THE UNIVERSITY OF CALGARY

Offline/Realtime Network Traffic Classification Using Semi-Supervised Learning

by

Jeffrey Erman

A THESIS
SUBMITTED TO THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE
DEGREE OF MASTER OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

CALGARY, ALBERTA

August, 2007

THE UNIVERSITY OF CALGARY

FACULTY OF GRADUATE STUDIES

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies for acceptance, a thesis entitled "Offline/Realtime Network Traffic Classification Using Semi-Supervised Learning" submitted by Jeffrey Erman in partial fulfillment of the requirements for the degree of MASTER OF SCIENCE.

Supervisor
Dr. Anirban Mahanti
Department of Computer Science
University of Calgary

Dr. Carey Williamson
Department of Computer Science
University of Calgary

Dr. Zongpeng Li
Department of Computer Science
University of Calgary

Dr. Subhabrata Sen
AT&T Labs
New Jersey

April 16, 2007

Date

Abstract

Identifying and categorizing network traffic by application type is challenging because of the continued evolution of applications, especially of those with a desire to be undetectable. The diminished effectiveness of port-based identification and the overheads of deep packet inspection approaches motivate us to classify traffic by exploiting distinctive flow characteristics of applications when they communicate on a network.

This thesis proposes a new machine learning approach for the classification of network flows using only flow statistics. Specifically, a *semi-supervised* classification method that allows classifiers to be designed from training data consisting of only a few labelled and many unlabelled flows. This thesis considers pragmatic classification issues such as longevity of classifiers and the need for retraining of classifiers. At the network core, only unidirectional flow records are available due to routing asymmetries. This thesis develops and validates an algorithm that can estimate the missing statistics from a unidirectional packet trace. The offline and realtime classifiers developed can achieve high flow and byte classification accuracy (i.e., greater than 90%).

Acknowledgments

In this section, I would like to thank the people who have helped me along the way in completing this M.Sc. thesis.

Firstly, I would like to thank my supervisor Professor Anirban Mahanti for his guidance, insights, patience, and encouragement which made this thesis possible. From him I have learned a great deal and most importantly he has taught me how to research. His enthusiasm and strives for excellence has made working with him a joy.

I also would like to thank Martin Arlitt. His help and expertise were very instrumental to my research. He collected the Internet traces from the University of Calgary that I used to properly evaluate my classifiers. He also assisted me with using Bro. His attention to detail and valuable comments have greatly improved this thesis and the papers we have written for publication.

I also would like to thank Ira Cohen and Professor Carey Williamson for their feedback and assistance in my research.

I thank my wife: Sherri. She is my best friend and has helped keep me sane at times. She has always given me her love and support. She has also been understanding when research has consumed all of my time. In addition, she has tirelessly listened to my many presentation rehearsals and proofread much of this document. Thanks Sherri!

I thank also my parents for always being supportive of education and encouraging me to pursue my ambitions.

Finally, I would like to give special thanks to my friend and mentor: John Quesnel. He has taught me much about computers, life, and writing proper English. I am grateful for the guidance and friendship he has given me over the years.

This thesis is dedicated to my wife, Sherri,
and to the fulfillment of our dreams..

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

The demand for bandwidth management tools that optimize network performance and provide quality-of-service (QoS) guarantees has increased substantially in recent years, in part, due to the phenomenal growth of bandwidth-hungry Peer-to-Peer (P2P) applications. Going by recent measurement studies in the literature and estimates by industry pundits, P2P now accounts for $50 - 70\%$ of the Internet traffic [9,68]. It is, therefore, not surprising that many network operators are interested in tools to manage traffic, such that traffic critical to business or traffic with realtime constraints is given higher priority service on their network. Critical for the success of any such tool is its ability to accurately, and in realtime, identify and categorize network flow by the application responsible for the flow. This task of mapping flows to the network applications that generate the traffic is called *traffic classification.*

1.1 Motivation

Identifying network traffic using port numbers was the norm in the recent past. This approach was successful because many traditional applications use port numbers assigned by or registered with the Internet Assigned Numbers Authority. The accuracy of this approach, however, has been seriously dented because of the evolution of applications that do not communicate on standardized ports [9,41,67]. Many current generation P2P applications use ephemeral ports, and in some cases, use ports of well-known services such as Web and FTP to make them indistinguishable to the port-based classifier. For example, KaZaA is known to use port 80 which is reserved for Web traffic.

Techniques that rely on inspection of packet contents [12, 35, 42, 49, 52, 67, 78] have been proposed to address the diminished effectiveness of port-based classification. These approaches attempt to determine whether or not a flow contains a characteristic signature of a known application. Studies show that these approaches work very well for today's Internet traffic, including P2P flows [35, 67]. In fact, some commercial bandwidth management tools use application signature matching to enhance robustness of classification [9, 58].

Nevertheless, packet inspection approaches pose several limitations. First, these techniques only identify traffic for which signatures are available. Maintaining an up-to-date list of signatures is a daunting task. Recent work on automatic detection of application signatures partially addresses this concern [35, 49]. Second, these techniques typically employ "deep" packet inspection because solutions such as capturing only a few payload bytes are insufficient or easily defeated (See Section 3.6 for empirical evidence of this.). Deep packet inspection places significant processing and/or memory constraints on the bandwidth management tool. On our network, for example, we have observed that during peak hours, effective bandwidth is often limited by the ability of the deployed commercial packet shaping tool to process network flows. Finally, packet inspection techniques fail if the application uses encryption. Many BitTorrent clients such as Azureus, μtorrent, and BitComet allow use of encryption.

The diminished effectiveness of the port-based and payload-based techniques motivates use of flow statistics for traffic classification [41, 51, 53, 65, 75]. These classification techniques rely on the fact that different applications typically have distinct behaviour patterns when communicating on a network. For instance, a large file transfer using FTP would have a smaller interarrival time between packets and larger average packet size than an instant messaging client sending short occasional messages to other clients. Similarly, some P2P applications such as BitTorrent [8]

can be distinguished from FTP data transfers because these P2P connections typically are persistent and send data bidirectionally; FTP data transfer connections are non-persistent and send data only unidirectionally. Although obfuscation of flow statistics is also possible, they are generally much harder to implement. There has been much work on scalable techniques for flow sampling and estimation (e.g., see [24, 25, 32, 43]), and furthermore, the logistics for collecting flow statistics is already available in many commercial routers (e.g., Cisco's NetFlow [13] solution).

1.2 Thesis Objectives

The three primary objectives of this thesis are:

- To propose a methodology that classifies network flows by application using *only* flow statistics.
- To apply this methodology to both offline and realtime classification, and evaluate the effectiveness of these classification approaches.
- To facilitate "rich" traffic classification at the network edge and at ingress/egress points of the network core and enable support for QoS provisioning of application specific guarantees.

The specific contributions of this thesis [27, 29–31] are elaborated upon in the following section.

1.3 Thesis Contributions

This thesis proposes a methodology that *classifies* (or equivalently, identifies) network flows by application using *only* flow statistics. Based on machine learning

principles, this methodology consists of two key components: a *learner* and a *classifier*. The goal of the learner is to discern a mapping between flows and applications from a training data set. Subsequently, this learned mapping is used to obtain a classifier. Traditionally, learning is accomplished using a fully labelled training data set, as has been previously considered in the traffic classification context [53, 65]. Obtaining a large, representative, training data set that is fully labelled is difficult, time consuming, and expensive. On the contrary, obtaining unlabelled training flows is inexpensive.

In this thesis, we develop a technique that enables us to build a traffic classifier using flow statistics from both labelled and unlabelled flows. This *semi-supervised* [10] approach to learning a network traffic classifier is one key contribution of this thesis. To the best of our knowledge, this is the first work to propose and explore semi-supervised classification for the network traffic classification problem. There are three main advantages to the proposed semi-supervised approach:

- Fast and accurate classifiers can be obtained by training with a small number of labelled flows mixed with a large number of unlabelled flows.
- This approach is robust and can handle both previously unseen applications and changed behaviour of existing applications. Furthermore, this approach allows iterative development of the classifier by allowing network operators the flexibility of adding unlabelled flows to enhance the classifier's performance.
- This approach can be integrated with solutions that collect flow statistics, such as Cisco's NetFlow [13] and Bro [59] (as done in this work). Furthermore, this approach can leverage recent work on techniques for flow sampling and estimation (e.g., [25, 32, 43]).

As a proof of concept, an implementation of prototype offline and realtime classification systems was done. A distinguishing aspect of this thesis is the implementation

of a realtime classifier in the Bro [59] Intrusion Detection System (IDS). Note that determining the application type while a flow is in progress is harder than offline identification because only partial information is available. This problem is addressed by designing a layered classifier that classifies flows at specific packet milestones using flow statistics that are available then.

Recent traffic classification efforts, including those that leverage flow statistics, are developed and evaluated assuming that the point-of-observation is the network edge where packet transmissions along both directions of a flow can possibly be observed. At egress/ingress points of a network core, observing both directions of a flow may not be possible because of routing asymmetries. This poses two challenges. First, statistics necessary for the satisfactory classification of a flow may not be available. Second, classification can only use per-flow information and cannot rely on additional information such as communication patterns between hosts.

To address these challenges, we study the influence directionality has on the predictive capability of different unidirectional flow statistics (e.g., packets originating only from the client, server, and combinations of both). Our observations lead us to develop and verify an algorithm capable of estimating the flow statistics for the unseen portion of a flow such as the number of bytes, and the number of packets.

We also consider the longevity of classifiers (i.e., how long they remain accurate in an operational network). To facilitate retraining, we present a heuristic for detecting retraining points. We expect this retraining detection heuristic to be used with the realtime classifier such that once retraining is deemed necessary, collection of additional flows for use as training data can be automatically initiated.

1.4 Overview of Results

Our evaluation of the classification accuracy of our approach is facilitated by recent full-payload packet traces from the University of Calgary's Internet link. We collected approximately one terabyte of traces during a 6-month period. Using a multi-pronged, semi-automated approach that consisted of payload-based identification, heuristics, and manual classification, the applications corresponding to individual flows were identified. These pre-classified traces were used as *base truth* to evaluate the accuracy of the classifier.

Using our offline and realtime classification systems, we find that flow statistics can indeed be leveraged to identify, with high accuracy, a variety of different applications, including Web, P2P file sharing, email, and FTP. In our evaluations, flow accuracies as high as 98% and byte accuracies as high as 93% were achieved.

We find that larger training data sets consistently achieve higher classification accuracies. While larger training data sets may appear to make the task of labelling the training data set time-consuming and difficult, we find that, in practice, a priori labelling of only a fraction of the training data is sufficient.

Our experiments with long-term Internet packet traces suggest that classifiers are generally applicable over reasonably long periods of time (e.g., on the order of weeks) with retraining necessary when there are significant changes in the network usage patterns including introduction of new applications.

In our evaluation of the influence of directionality of flow statistics in classifying traffic, we find that flow statistics along the server-to-client path of TCP connections achieve, on average, significantly higher byte accuracies compared to flow statistics along the client-to-server path; directionality appears to not have any significant impact on flow accuracy, with both directionalities attaining high flow accuracy. Based on our results, we hypothesize that statistics along the server-to-client path

have a greater ability to discriminate between flows than statistics along the client-to-server path. We believe this to be the case because for many common network applications the flow of application payload data is greater in the server-to-client path.

1.5 Thesis Organization

The remainder of this thesis is organized as follows. Chapter 2 provides background on the Internet's TCP/IP architecture. It also reviews prior work on Internet traffic classification problem that is most relevant to this work. Chapter 3 describes the data sets used in this work. Chapter 4 presents the proposed semi-supervised classification framework. Chapter 5 describes and analyzes different clustering algorithms that could be used in this framework. Chapter 6 discusses our offline and realtime classification results. In addition, it discusses the history of the traffic classification problem, longevity, and retraining point detection. Chapter 7 provides our classification results for unidirectional statistics. It describes our flow statistics estimation algorithm, its validation, and the classification results obtained with estimated statistics. Chapter 8 presents conclusions and directions for future work.

Chapter 2

Background and Related Work

This chapter provides background on the Internet protocol suite and describes different traffic classification techniques in the literature. Section 2.1 describes the TCP/IP architecture, which is the protocol suite used to transfer data on the Internet. Sections 2.2-2.5 describe different approaches to traffic classification. Section 2.6 presents other related research of interest to this thesis. Section 2.7 contrasts previous traffic classification approaches to the classification approach proposed in this thesis.

2.1 TCP/IP Architecture

The TCP/IP Architecture of the Internet is explained in this section to give the reader an understanding of how network traffic is transmitted across the Internet. This understanding is fundamental to the work in this thesis because most prior classification approaches including the approach advocated in this thesis rely on information obtained from the protocols that are used to transmit the messages for the user's applications.

The Internet is based upon the concept of *packet switching* [45]. Instead of transmitting a message between two hosts as a single large message, the message is broken up into smaller pieces called *packets*. These packets are then separately delivered to the other host. This allows the message to be transmitted and not require all the communication links along the sender-to-receiver path to be reserved during message transmission. The use of packet switching has several advantages including: increasing the throughput of the communication links, increasing the robustness of

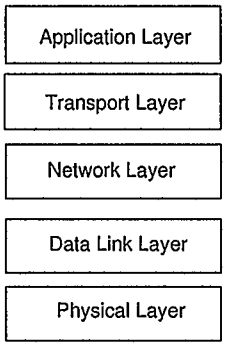


Figure 2.1: TCP/IP Layered Network Model

the communication, and reducing the latency.

Protocols can be defined as a set of rules that govern the transfer of data between two hosts. The protocols used in the Internet form a "protocol suite". In this protocol suite, the two most important protocols are the Transmission Control Protocol (TCP) and the Internet Protocol (IP). These two protocols together ensure that when data is transfered it is delivered to the correct end host reliably and in the order it was sent. This is to overcome the fact that the communication links that the data is traveling over are unreliable due to losses, delays, and communication errors that occur.

The protocols used for the Internet are organized into a layered network model. The lower layers of the network model provide services to the higher layers. This abstraction allows the individual layers to be developed concurrently and reduces the complexity of transferring data. The TCP/IP Architecture can be divided into five logical layers as presented in Figure 2.1 [45].

The bottom two layers of the protocol stack, the physical layer and the data link layer, deal with preparing the packet and transmitting the actual bits across the physical medium to between two hosts. Most relevant to the work in this thesis

are the top three layers of the network protocol stack, namely, the network layer, the transport layer, and the application layer. These layers determine how packets are routed through the network, how data can be transferred reliably, and how applications or protocols transfer data.

These will be explained in more detail in the following subsections.

2.1.1 Network Layer

The network layer is responsible for sending individual packets between hosts. The protocol that is responsible for this is called the Internet Protocol (IP). In IP, each packet is independent, and thus, requires full addressing information to be included in each packet header. IP provides a "best effort" delivery service, which means that it does not guarantee that a packet reaches its end destination.

One main service that IP provides is 32-bit addressing in IP version 4 [62]. This will be upgraded to 128-bit addressing when IP version 6 [20] is fully deployed. When a packet is sent using IP, the addressing in the packet header is used by intermediate routers to determine which path to send the packet. All IP packets are assigned a "Time-To-Live" value. If a packet does not get delivered and circles through too many routers it is eventually dropped when its Time-To-Live expires. This ensures that undelivered packets do not travel the network indefinitely. Other services that IP provides are fragmentation, type of service, and optional fields [45].

2.1.2 Transport Layer

As the network layer handles each packet individually, the logical communication between two end hosts is provided by the transport layer to the application layer. This transparency can include ensuring all data is delivered, providing error recovery, flow control, and congestion control.

The most common transport layer protocol used is the Transmission Control

Protocol (TCP) [63]. This protocol provides *reliable* transfer of data for applications. Another transport layer protocol is the User Datagram Protocol (UDP) [61], which provides more limited services.

In the following subsections, the multiplexing/demultiplexing of flows provided by the transport layer will be discussed. In addition, the functionality of TCP and UDP will be further elaborated upon.

Multiplexing and Demultiplexing Flows

The transport layer allows hosts to have multiple simultaneous data transfers to other hosts (or even the same host). This is facilitated by the transport layer using special fields called port numbers. In TCP and UDP, there is a source port number field and a destination port number field that tracks the port numbers used by each host for a flow. The port numbers are used at the host to determine which flow an arriving packet should be assigned to. Flows can be uniquely identified using the 5-tuple of source and destination IP addresses, source and destination port numbers, and the transport layer protocol.

The port number fields in TCP and UDP are 16 bits, which allows the port numbers to range from 0 to 65535. The port numbers from 0 to 1023 are typically reserved for well-known applications or protocols. These well-known port numbers are generally used by the host to receive incoming connections from other hosts. For example, a Web server normally uses port 80 for incoming HTTP connections.

The ephemeral port numbers range from 1024 to 65535. Generally, these are dynamically assigned. Ephemeral port numbers are used when the host does not care what the assigned port number is such as when the host is establishing a connection to a remote service. In continuing with the above example, a host (the client) would use an ephemeral port number such as 1024 to establish a HTTP connection to the Web server running on port 80.

The well-known port numbers used by a server have in the past been a strong indicator of the application type of a flow; however, recently they are increasingly becoming ineffective. We discuss port-based classification further in Section 2.2.

Overview of TCP

The TCP is connection-oriented and provides reliable transparent transfer of data to the application layer [45, 63]. In addition, TCP is full-duplex, which means that both end hosts can send data. The reliable data transfer TCP provides ensures that data is delivered to the application in-order, that data has minimal errors, that duplicate data is discarded, and that lost/discarded packets are resent. TCP includes congestion control mechanisms that modulate a sender's transmission rate such that network resources along the path from the sender to the receiver are shared fairly with other competing flows.

TCP header data is attached to each packet to provide these services. Unlike IP, the TCP header data is not used by any of the intermediate routers along the network path and is instead only used by the end hosts running TCP. The TCP header is depicted in Figure 2.2 [45].

The congestion window field in the TCP header is used to specify the maximum receive buffer size of a host. This is used to provide flow control so that a host's receive buffer is not overwhelmed by arriving data.

There are special bit flags in the TCP header that can be set to signal information to the other host. For example, the SYN flag is used for the establishment of connections, the FIN and RST flags are used for the termination of connections, and the ACK flag is used to provide reliability. These are further described in the following paragraphs. The TCP header also has URG and PSH flags but these are seldom used.

The TCP connection is established using a three-way handshake. This is a special

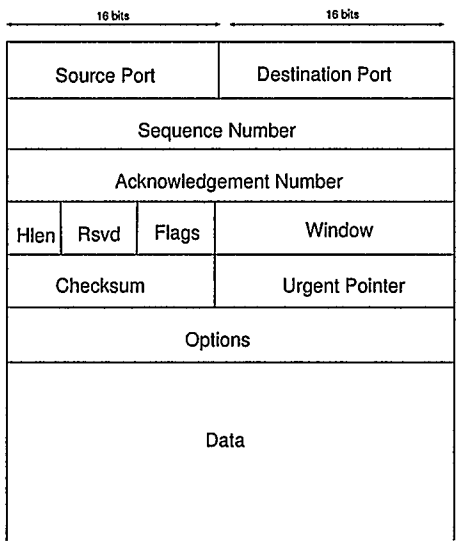


Figure 2.2: TCP Header

sequence of packets that are sent to ensure both hosts have established the connection. The host establishing the connection first sends a packet with the SYN flag set. The other host responds with a packet with the SYN and ACK flags set. Concluding when the original host acknowledges that it received the SYN/ACK packet sent by sending a packet with the ACK flag set. To terminate a connection, packets with FIN or RST flags set are used. We used these flags (e.g., SYN, FIN, ACK) to help differentiate the start and end of flows when the 5-tuple (source address, source port, destination address, destination port, transport protocol) is identical between flows.

Reliable transfer functionality in TCP is provided most importantly by using the sequence and acknowledgment number fields in the TCP header. The sequence numbers allow the payload data in the TCP packets to be reassembled in-order. The assigned sequence number of a packet corresponds to the Nth byte of the data stream and not the packet number. The acknowledgment numbers are used to cumulatively acknowledge that up to the Nth byte has been successfully received. Packets

acknowledging received data have the ACK flag set as well. If an acknowledgment is not received for a packet within a set period of time (e.g., a timeout occurs) or if multiple duplicate ACK's are received, then it is assumed that a packet has been lost and the packet assumed to be lost is resent by the sender.

The sequence and acknowledgment numbers in TCP are extensively used in Chapter 7 to develop our algorithm to estimate flow statistics from unidirectional flows at the network core. Different variants of TCP exist like TCP Reno, TCP New Reno, and TCP Vegas, and have varying acknowledgment policies such as for when to sent acknowledgment packets and how many unacknowledged packets can be outstanding. These policies allow TCP to have functionality like fast recovery of packet losses. We further elaborate on these differences in Chapter 7 when discussing our estimation algorithm.

Overview of UDP

UDP [61] is an alternative to the aforementioned TCP protocol. Compared to TCP, UDP provides limited services for messages that are exchanged using it [45].

UDP is a connectionless protocol that provides "best effort" delivery much like IP. UDP is lightweight compared to TCP because it does not have any connection setup (e.g., three way handshake) or tear down costs, and does not incorporate any mechanisms for reliable data delivery. This is advantageous to applications that are more sensitive to the timely delivery of data than reliability or to applications wishing to send small messages where the overhead of TCP would be significant. Examples of applications that typically use UDP are streaming media and DNS, respectively.

UDP does provide port numbers for the multiplexing and demultiplexing of multiple user requests. In addition, UDP provides basic error detection through an optional checksum that can be used.

For applications that require some reliability and error-correction, these features

may be built into the application layer of the applications using UDP.

In this thesis, UDP is not considered in the evaluation of our classification technique for three reasons. First, due to the stateless nature of UDP, clear identification of flows is not possible as is with TCP. Second, UDP accounts for a negligible amount of traffic in the traces used in this work. Third, there is a low-level of diversity amongst applications using UDP in our traces making classification in some cases trivial. However, while each UDP packet is independent, many UDP-based applications conceptually behave much like flows. For example, streaming media applications sent a continual flow of UDP datagrams between hosts. In another example, query-based applications send requests and receive responses in several datagrams in short succession. We further elaborate in Section 3.3 on how the message exchanges of these applications could be identified as flows and we discuss how the classification technique in this thesis could be applied to these UDP "flows".

2.1.3 Application Layer

The application layer allows the user's applications to communicate on the network. Many common applications communicate on the Internet with standard protocols that are used to provide compatibility amongst all applications of that type. For example, web browsers all communicate with an application-layer protocol called HTTP.

In the early days of the Internet, only a few applications were prevalent. These included electronic mail and simple file transfers. However, as the Internet has continued to evolve, the number of applications prevalent has grown substantially. Table 2.1 non-exhaustively lists some of the major application types available and the common application-layer protocols associated with them. The identification of these applications and application-layer protocols communicating on a particular flow is the main focus and goal of the approach proposed in this thesis.

Table 2.1: Internet Applications

Application Types	Application-Layer Protocols
Web	HTTP, HTTPS
Bulk File Transfer	FTP
Chat	MSN Messenger, AOL, IRC
Email	SMTP, POP3, IMAP
Remote Computing	Telnet, SSH
Peer-to-Peer (P2P)	BitTorrent, KaZaA, Gnutella, eDonkey

In the following sections, different approaches to identify the applications or their application-layer protocols will be discussed. These include approaches that use port numbers of well-known applications in Section 2.2, approaches that look for payload signatures inside packets in Section 2.3, and finally, approaches that are behavourial and machine learning based in Sections 2.4 and 2.5, respectively.

2.2 Port-based Classification

Historically, traffic classification techniques used well-known port numbers to identify Internet traffic. This was successful because many traditional applications use fixed port numbers assigned by or registered with the Internet Assigned Numbers Authority (IANA) [38]. Table 2.2 shows a partial list of the IANA port assignments for some selected applications and protocols. For example, email applications commonly use the Simple Mail Transfer Protocol (SMTP) on port 25 to send email and the Post Office Protocol version 3 (POP3) on port 110 to receive email.

Port-based classification has been shown to be ineffective because many recently developed applications do not communicate on standardized ports [9, 41, 67]. The current generation of P2P applications, many of which intentionally try to obfuscate their traffic, use ephemeral ports or use the port numbers of well-known applications to make the traffic indistinguishable to port-based classification and filtering. For

Table 2.2: IANA Assigned Port Numbers for Selected Applications and Protocols

Application	Port Numbers
FTP Data Channel	20
FTP Control Channel	21
SSH	22
Telnet	23
SMTP	25
DNS	53
HTTP	80
POP3	110
IRC	113
NNTP	119
SOCKS	1080

example, Madhukar *et al.* conducted a longitudinal study over a 2-year period on the effectiveness of port-based classification using empirical Internet traces taken from the University of Calgary [50]. The authors compared port-based classification with a classification technique that relies on a set of transport layer heuristics (discussed in Section 2.4). Their trace only had SYN, FIN, and RST packets due to the longitudinal nature of their trace, and thus, validation of their classification results using payload-based techniques (discussed in Section 2.3), for example, was not feasible. They found that 30% to 70% of the traffic is classified as unknown with port-based analysis. In addition, they found that the amount of unknown traffic was typically from 10% to 30% in the September 2003 to April 2004 portion of their trace. It has since increased from 30% to 70% by the Spring of 2005. They provide strong circumstantial evidence that this increase in unknown traffic is highly correlated to the increase in P2P traffic found with their transport-layer heuristic.

The ineffectiveness of port-based classification has spurred researchers and commercial vendors to find more effective methods of identifying network traffic. We discuss these approaches in the following sections.

2.3 Payload-based Classification

Another approach to Internet traffic classification that avoids port-based identification is analysis of packet payloads and is sometimes referred to as "Deep Packet Inspection". In this approach, the packet payloads are analyzed to see whether or not they contain characteristic signatures of known applications. These approaches have been shown to work very well for Internet traffic including P2P traffic [35, 52, 67]. However, these techniques also have drawbacks. First, these techniques typically require increased processing and storage capacity. Second, these approaches are unable to cope with encrypted transmissions. Finally, these techniques only identify traffic for which signatures are available, and are unable to classify previously unknown traffic. The payload-based approach has been well researched and the work presented here represents the current state-of-the-art for commercial traffic classification products.

One example of a study integrating payload-based analysis into a classification approach is the work by Moore *et al.* [52]. They describe a content-based methodology to classify network traffic. The first step of their classification methodology uses IANA assigned port numbers to create an initial classification. Then, using an iterative procedure, they use increasingly more information at later steps. This approach allows the traffic to be classified with increased confidence. The last step concludes the process by relying on manual analysis of the traffic for any remaining unclassified traffic.

Moore *et al.* compared the effectiveness of port-based classification to this content-based approach [52]. To facilitate this comparison the authors collected a 24-hour trace of the traffic generated from approximatley 1,000 users. This comparison found that approximately 30% of the bytes in the trace are either misclassified or unclassified when using just the IANA port assignments. However, with the content-based

approach 99.9% of the traffic was identified confidently.

In the remainder of this section, the research identifying payload signatures will be described. As well, some research conducted to address the aforementioned concerns such as the automatic detection of signatures and decreasing the processing requirements of deep packet inspection will also be outlined.

2.3.1 Identifying Payload Signatures

In [67], Sen *et al.* develop an approach to accurately identify P2P applications. Their approach is based on application-level payload signatures. The focus of their research is to identify signatures that are highly accurate, that are scalable for analysis of large volumes of traffic, and that are robust to variable network dynamics such as packet loss, asymmetric routing, and packets arriving out of order. Their work focused on the five most predominant P2P applications: Gnutella, eDonkey, Direct Connect, BitTorrent, and KaZaA.

The authors [67] implemented their signatures and found that signatures with a fixed-offset are trivial to implement and have a low computational overhead; while, variable-offset signatures are much more computationally expensive[1]. The method is validated on two full packet traces both collected in November 2003 that contain 120 Gigabytes and 1.8 Terabytes of data, respectively. They found that by examining a few packets in each flow over 99% of the P2P traffic could be identified. The authors also analyzed port-based identification and found that 30% to 70% of the traffic for KaZaA and Gnutella use non-standard port numbers whereas only 1% to 4% of the traffic for BitTorrent and eDonkey use a non-standard port.

Karagiannis *et al.* [41] uses a similar payload-based methodology to Sen *et al.* [67] for identifing P2P applications. Karagiannis *et al.* [41] later extends these signatures

[1]Examples of both fixed-offset and variable-offset signatures are given in Chapter 3 when we discuss our payload-based approach used to obtain "base truth".

to encompass all application types [42]. In both [41] and [42], Karagiannis *et al.* use the payload analysis to provide "base truth" to compare new behavioural-based traffic classification methods that they propose. These behavioural-based methods are further discussed in Section 2.4.

In earlier work, Dewes *et al.* look at the network traffic dynamics of Internet Chat Systems [22]. The authors focus on IRC and web-based chat systems. Their paper describes a port and payload-based methodology for identifying the chat flows and filtering out non-chat traffic. Their approach uses well-known port numbers to filter out traffic that is most likely non-chat such as Gnutella traffic on port 6346. After this filtering has taken place they use payload signatures to separate the web-based chat flows from the regular non-chat traffic.

2.3.2 Automated Detection of Payload Signatures

One of the concerns of payload-based analysis of network traffic is the identification of characteristic signatures for use in deep packet inspection. Haffner *et al.* address this problem by attempting to automatically learn the application signatures using three machine learning algorithms [35]. The algorithms studied include Naive Bayes, AdaBoost, and Regularized Maximum Entropy. The approach uses a binary feature vector to train the algorithms, which is obtained from the first n-bytes of a flow's payload. The flow's payload is encoded into binary vectors so that for each of the n bytes of payload, the binary vector has 256 elements corresponding to this byte. Each of these elements is initialized to 0 first and then the element whose number corresponds to the value contained in this byte is set to 1.

The authors validate their approach using *training* and *test* data[2] collected in 2004 and 2005 [35]. These algorithms are tested upon FTP, SMTP, POP3, IMAP,

[2]In supervised machine learning training data is used to learn a function that can be used to predict the class labels of test data. This is discussed in more detail in Chapter 4.

HTTPS, HTTP, and SSH applications. The "base truth" for these applications is obtained using port-based analysis. Overall, the algorithms are shown to have a high accuracy identifying specific applications in the case where only the first 64 to 256 bytes of the payload are used in constructing the feature vectors. However, none of these applications identified contain signatures that have variable-length offsets such as the Gnutella P2P application.

Haffner *et al.* relied on training the classifiers with each specific application it wanted the classifiers to identify [35]. Recently, Ma *et al.* extend this work by proposing an unsupervised approach to the detection of application signatures [49]. This allows similar flows (most likely from the same protocol) to be grouped together. These groups (clusters) are then labelled in a later step to create a classification of the current and future flows placed into that group. The authors achieve this by using a generic classification framework and compare the use of three different methods: product distributions of byte offsets, Markov models of byte transitions, and common substring graphs. The authors evaluate methods to determine if flows from the same protocols are grouped together and that a new protocol is placed into a separate group when it is introduced. The misclassification rate varied between 2% to 10% with their various methods.

2.3.3 Speeding up Deep Packet Inspection

Sen *et al.* [67] found that payload analysis is much more computationally expensive when the payload signatures use a variable-length offset instead of a signature based on a fixed-length offset. Kumar *et al.* address this problem by proposing algorithms to increase the speed of deep packet inspection of regular expressions [44]. The authors propose a new method of representing regular expressions that condenses the transition state space and reduces the previously large memory requirements for regular expression matching. The method is evaluated using regular expressions

obtained from several popular Intrusion Detection Systems such as Snort [64] and Bro [59]. The evaluations show that, with a careful implementation, regular expression matching of full-packet payloads can be successfully achieved at Gbps link speeds.

2.4 Behavioural-based Classification

Karagiannis *et al.* in [40, 41] classify P2P traffic and report on trends in the usage of P2P file sharing. The authors analyze data from a tier 1 ISP; however, they are limited by having only 16 bytes of payload data available and only 4 bytes in some of their older traces. This would limit the effectiveness of an analysis and evaluation using only payload-based classification. Instead, the authors develop a non-payload based method, specifically two transport layer heuristics to classify P2P.

One of the heuristics looks for IP addresses that are concurrently using both TCP and UDP. This heuristic works on the basis that most P2P applications typically send control information by UDP and transfer data by TCP. Flows using port numbers of well-known UDP applications such as DNS on port 53 are excluded to reduce false positives. The second heuristic looks at the ratio of the number of unique IP addresses to unique port numbers to which a host is connected. If this ratio is roughly equal then the flows from this host are classified as P2P. A higher ratio would tend to indicate a non-P2P type of flow such as HTTP because multiple concurrent flows are generally spawned from a web server to decrease the response times when a web page with multiple objects is requested. Karagiannis *et al.* validate these heuristics by creating a "base truth" using well-known port numbers of P2P applications, payload signatures, and a heuristic where if a IP address and port number pair had previously been used for a P2P flow in the last five minutes then future unlabelled IP/port pairs would also be classified as P2P. The transport layer heuristics were shown to be able to identify 90% of the total P2P bytes and 99% of the P2P flows. In addition,

the transport layer heuristics were able to identify P2P traffic that was previously unidentified with the payload analysis method used to establish the "base truth".

Karagiannis *et al.* more recently have developed a classification approach based on the analysis of communication patterns of hosts [42]. This system leverages information obtained from the social, the functional, and the application layers to identify the application classes of particular flows from a host. The social level information is information such as the popularity of a host and the communities with which the host communicates. The functional level attempts to determine if the host's communication paradigm is client/server or collaborative (e.g., P2P). The application layer uses the communication patterns of application protocols referred to by the authors as "graphlets" to identify the applications. Constantinou *et al.* propose a similar technique that looks at the connection graph of hosts [15].

Concurrent to [42], Xu *et al.* [74] developed a methodology, based on data mining and information-theoretic techniques, to discover functional and application behavioural patterns of hosts and the services used by the hosts. They subsequently use these patterns to build general traffic profiles, for example, "servers or services", "heavy hitter hosts", and "scans or exploits".

2.5 Machine Learning based Approaches

Another promising approach to traffic classification is the use of machine learning. This approach relies on the premise that a set of *features* for objects would be similar when objects are of the same class. In general, a feature can be any attribute that is relevant to the prediction of the target set of classes. In the case of traffic classification, the objects dealt with are flows and the classes are the different applications or traffic types the flow is attempted to be classified as.

Generally, in machine learning there are two stages when developing a classifier.

The first stage "learns" a mapping between the objects and the desired classes. This mapping is done using a labelled training data set. Subsequently, in the second stage this learned mapping is used by the classifier to label new objects. This framework is elaborated upon in Chapter 4.

In the following subsections, Section 2.5.1 presents related research that analyze different candidate features for use in network traffic classification and evaluate their ability to separate flows into distinct groups. Section 2.5.2 describes different approaches that build a machine learning based classifier for network traffic classification.

2.5.1 Analysis and Selection of Features

Obtaining a set of relevant features is a difficult problem in machine learning [46]. As such, the focus of much of the prior work using machine learning techniques has been on demonstrating the ability of algorithms to group together flows according to application type and *not* on classifying traffic (e.g., [27, 37, 51, 65, 75, 76]). These techniques generally use only features obtained from a single flow such as packet sizes, interarrival times, or aggregate statistics. These approaches do not consider the application labels of the flows when forming the groups. In the machine learning literature, this can be characterized as "unsupervised" learning because the labels are not used [23, 39].

Clustering algorithms[3] [23, 39] are the most common type of unsupervised machine learning algorithms used in this research topic. Many of the following studies use clustering algorithms.

Hernandez-Campos *et al.* study, using an abstract model, how to represent application level communications [37]. Their abstract model represents the communication patterns of a flow in "epochs" that store the amount of data traveling to both

[3]Clustering algorithms are further discussed in Chapter 5.

the sender and receiver, and the idle time between exchanges. The feature vectors for a flow are extracted from these epochs. Hernandez-Campos *et al.* then use hierarchical clustering [23, 39] to group the flows based on similarity. They found when 5,000 flows were clustered that many of the clusters corresponded roughly to a single application. For example, one of their clusters contained web flows and another contained flows from mail protocols.

Roughan *et al.* [65] classified flows into four predetermined traffic classes (interactive, bulk data transfer, streaming, and transactional) using the Nearest Neighbor and the Linear Discriminate Analysis classification techniques. Roughan *et al.* show that it is possible to successfully separate the flows of different traffic classes using only flow statistics and give explanations to why their chosen flow statistics (average packet size, and flow duration) would work for the different traffic classes they studied.

McGregor *et al.* analyzed packet sizes and interarrival times of different application types to determine whether different applications exhibit different packet size and interarrival characteristics [51]. In analyzing plots of packet sizes and interarrival times, they found that while there were some distinguishing characteristics between applications it would be difficult to do rich traffic classification. McGregor *et al.* then proposed a methodology to use Expectation Maximization (EM) clustering that will group flows using flow statistics including byte counts, connection durations, and packet size statistics. The authors conducted a preliminary analysis using cluster visualization to examine the clusters and find that many of the clusters correspond to a single type of traffic class such as bulk data transfers and DNS traffic.

Zander *et al.* [75, 76] extend the aforementioned work. Specifically, they look at maximizing intra-cluster homogeneity (or cluster purity) by investigating which set of features separate the flows from different applications with greatest accuracy. The traces used in this analysis are from a publicly available archive of traces and

port-based analysis was used to establish the "base truth". The authors have continued this work and recently used the C4.5 supervised machine learning algorithm to estimate the traffic trends in archival traces [77].

In our own research, we have investigated the ability of clustering algorithms to group together flows by application type using flow statistics as features [27]. We considered three different clustering algorithms (K-Means, DBSCAN, and AutoClass), and showed that these algorithms can form clusters where each cluster largely consists of applications of a single type. Our investigations are presented in Chapter 5.

We conclude this discussion by noting that two distinct types of features have been used for traffic classification. The first type of features use "aggregate" flow statistics such as mean packet sizes, and flow durations. The second type of features use the "individual" packet sizes and interarrival times. In the following subsection, we discuss classification approaches based on each of these distinct feature types. Note, we provide a discussion of the advantages and disadvantages between these types of features in Section 2.7.

2.5.2 Classification Approaches

In this section, we discuss the classification approaches that use machine learning to build a classifier. These classifiers predict the application labels of new flows to accomplish the traffic classification. As previously mentioned, these approaches consist of two stages: a learning stage and a classification stage. In the learning stage, labelled *training* data is used to create the flow to application mappings. This can generally be considered as "supervised" learning because the training flows must be labelled *before* the learning occurs [23, 39]. The approach taken in this thesis is semi-supervised because we only require a small portion of the flows in the training data to ever be labelled. This is explained in Chapter 4.

We first describe classification approaches that use aggregate flow statistics as their features. The semi-supervised classification technique proposed in this thesis also utilizes aggregate flow statistics.

Moore *et al.* extensively study the suitability of a Naïve Bayes classifier for Internet traffic classification [53,79]. A similar use of Naïve Bayes was first proposed in [1]. The Naïve Bayes algorithm is one of the simpler supervised machine learning algorithms available. The algorithm is built on the assumption that features are independent and identically distributed. The Naïve Bayes method estimates the Gaussian distribution of the features for each class based on labelled training data. A new flow is classified based on the conditional probability of the connection belonging to a class given its attribute values. The probability of belonging to the class is calculated for each attribute using the Bayes rule:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)},$$

where A is a given class and B is a fixed attribute value. These conditional probabilities are multiplied together to obtain the probability of an object belonging to a given class A.

Moore *et al.* [53] use a large hand classified trace to evaluate this approach. In addition, they outline an exhaustive list of 248 flow features [54] and find that their classifier's performance suffers when redundant or irrelevant flow features are used. To overcome this problem, feature reduction is used to reduce this list to the 12 most frequently used features. In the evaluation, the feature reduction and refinements made to the Naïve Bayes algorithm allow the classification accuracy of flows to increase from 65% to over 95%. This work is further continued in [47] with the goal of adapting this approach to realtime traffic classification.

Our semi-supervised classification methodology complements prior efforts of Moore *et al.* [47,53,79]. Our approach is amenable to introduction of new applications and

behavioural changes of existing applications. We presented a preliminary comparison of our approach to Naïve Bayes in [28].

Recently, Williams *et al.* [72] compared five supervised learning algorithms and four different methods of feature reduction. Unfortunately, this study has some drawbacks. First, the evaluations relied on older data sets from 2000 and 2001 and used port-based analysis to obtain the "base truth". Second, the traffic classes tested did not include P2P which is much more elusive and important to classify. We believe that the results from this preliminary comparison are inconclusive and do not show any significant advantages of using a more complex supervised machine learning algorithms over the simpler Naïve Bayes algorithm used in prior work [47, 53].

Nguyen *et al.* continue the work of Williams *et al.* in [56, 57]. They study using "sub-flows" for traffic classification. In this work, the sub-flows are a window of packets from which flow statistics are calculated from. Nguyen *et al.* test their approach by attempting to identify from an online game called Wolfenstein Enemy Territory with a test data set containing interfering traffic. They find that a sub-flow as small as 25 packets allows for accurate detection of the Wolfenstein Enemy Territory online game traffic.

The classification approaches we describe in the sequel use per-packet statistics such as packet sizes of individual packets, and a sequence of packet inter-arrival times.

Dedinski *et al.* investigate a technique to identify P2P traffic on a network [19]. The authors collect traces from an isolated network where a P2P application (eDonkey) and a FTP application are downloading a 600 MB file. The packet size distributions and interarrival times are analyzed for each type of flow using wavelet analysis. Overall, the author's preliminary results found that it was possible to distinguish between FTP and eDonkey. However, no qualitative results were given.

Bernaille *et al.* [5,6] further explore traffic classification using the sizes of the first

P packets of a TCP session. The classifier proposed by Bernaille *et al.* is the closest work to our own classification approach presented in Chapter 4. Their proposed classifier similarly uses the K-Means algorithm and a minimum distance measure to assign a flow to an application label. Their empirical study shows that flow accuracy up to 98% can be achieved for some applications. However, they were unsuccessful in classifying application types with variable-length packets such as Gnutella.

Bernaille *et al.* further test their classification approach on encrypted traffic [4]. A test data set for their experiments is created by replaying unencrypted flows that had previously been labelled by payload signatures over a SSH tunnel. The new trace now contains only encrypted traffic. When the classifier is trained with a training data set of 500 unencrypted flows for several application types, and tested on the encrypted trace the flow accuracy is 85%. However, neither of these studies by Bernaille *et al.* [4,6] assess byte accuracy, which makes direct comparison to our work difficult.

In [17,18], Crotti *et al.* present an approach to traffic classification that is similar to Bernaille *et al.* [6]. This approach uses packet sizes, interarrival times, and arrival order of the first N packets as features for their classifier. The authors construct *protocol fingerprints*, which are histograms of the observed variables for a flow. The author's approach, which is noted to be susceptible to noise, incorporates a Gaussian filter into the *anomaly score* used to determine the likelihood of a flow belonging to a traffic class of the protocol fingerprint. The results that have been presented with this approach are preliminary and focus on only HTTP, POP3, and SMTP. This precludes direct comparisons to our work, which is more holistic in its evaluation of all the issues surrounding traffic classification.

2.6 Other Classification Approaches

While not directly related to traffic classification the following research provides some insight into the work done in this thesis.

Saddi *et al.* propose an approach to estimate the amount of mice (short flows) and elephants (long flows) in a trace [66]. The authors present the idea that larger packets tend to indicate elephant flows. This is an interesting finding as it indicates aggregate packet size as a relevant discriminator for application type as elephant and mice flows are generally associated with different types of applications. We confirm this in Section 6.1.3 when we find that packet size is good feature to use in our classification approach.

In [70], Sun *et al.* use statistical information to identify encrypted web browsing traffic. A similarity-based Nearest Neighbour algorithm is used. The features used include the size of the web page and the sizes of the objects the a web page that are inferred from the TCP connection's packet exchanges. The authors explore countermeasures such as padding, mimicking, and morphing, and their associated cost. These counter measures are the same that could be used to disguise a flow in the traffic classification problem.

2.7 Discussion

This section discusses our proposed semi-supervised classification technique in comparison to the aforementioned traffic classification approaches in the literature. We believe the approach presented in this thesis offers some distinct advantages that overcome some of the drawbacks of previous approaches.

The traditional classification techniques such as those based on well-known port numbers or payload analysis are either no longer effective for all types of network traffic or otherwise have several drawbacks such as the inability to classify encrypted

traffic and increased processing overhead. Our proposed classification technique, as well as the behavioural and machine learning based approaches, overcome these issues by avoiding the use of port numbers and packet payload information in the classification process.

The behaviour-based approaches proposed by Karagiannis *et al.* [42] to classify network traffic have some limitations and drawbacks. At the heart of their approach is the use of *graphlets* to identify the connection patterns of the different traffic types. However, to add a new graphlet that can uniquely distinguish itself from all other graphlets would be a difficult task. Another limitation of the graphlets is that some of the most predominant traffic classes require heuristics (tunable parameters) to distinguish themselves from each other. The optimum settings for these heuristics, for instance, to successfully discriminate between HTTP and P2P seem to be network and application dependent; P2P users can change their application settings and possibly evade detection. Another drawback is the need to store information across multiple flows. As Roughan *et al.* discuss "*multi-flow* features are more complex and computationally more expensive to capture than flow or connection data alone" [65]. In contrast, our work relies upon and advocates using aggregate flow statistics that can easily be computed from a single flow that *do not* require per-packet information to be stored, and achieve comparable or better accuracies when classifying traffic, including traffic originating from P2P applications.

We believe that our semi-supervised approach offers some distinct advantages over supervised machine learning approaches. One of the main benefits of our semi-supervised approach over supervised machine learning is that new applications can be identified by examining the flows that are grouped to form new clusters. The supervised approach cannot discover new applications and can only classify traffic for which it has labelled training data.

Another advantage occurs when the flows are being labelled. The labelling of the

entire data set that is representative of all applications for the supervised approaches is difficult, time consuming, and expensive. Our semi-supervised approach is accurate with a training data set that has only a few labelled flows mixed with many easily obtainable unlabelled flows.

One main difference between the work of Bernaille *et al.* [5, 6] and Crotti *et al.* [17, 18] and the work in this thesis is the choice of features. Bernaille *et al.* explore the potential of classifying traffic using the size of the first P packets of a TCP session. Conceptually, their approach is similar to payload-based approaches that look for characteristic signatures during protocol handshakes to identify the application and is unsuccessful in classifying application types with variable-length packets in their protocol handshakes such as Gnutella. As noted by Bernaille *et al.*, "the main challenge to traffic classification techniques in general is evasion. For instance, an 'attacker' could easily evade our method by padding packet payloads in order to modify sizes" [6]. Our approach is much more robust to this type of attack and as we discuss in Section 6.3.1 would require a crippling amount of overhead for an attacker to defeat it.

Neither of these studies [5,6,17,18] assesses the byte accuracy of their approaches which makes direct comparison to our work difficult. Our evaluation suggests that achieving a high flow accuracy is relatively easy. The more difficult problem is obtaining a high byte accuracy as well. Our work concentrates on achieving both high flow and byte accuracy.

Some prior studies focused on traces with a limited number of application: 10 in [6] and only 3 in [17]. In this thesis, we use traces that span several months, and furthermore, we try to classify all traffic in our traces because we find that some applications with only a few flows can still contribute substantially to the amount of bytes transferred in the network. In addition, we address many of the challenges outlined in their research including selection of training data sets, classifier longevity,

automatic detection of retraining points, an ability to leverage unlabelled training data. We have found that realtime classification using a hierarchy of classifiers substantially improved the classification accuracy.

2.8 Summary

In this chapter, we presented an overview of the TCP/IP protocol suite. In addition, we described different approaches to network traffic classification. The historical approach of traffic classification using port-based analysis is ineffective. Payload-based approaches using characteristic signatures currently provide accurate classifications; however, they have many challenges including the inability to identify encrypted traffic. Recent research has focused on overcoming the challenges of traffic classification through approaches that are based either on the behaviour of hosts or through approaches that use machine learning.

The next chapter describes the methodology we used to collect our network traces, how base truth was established, and presents an overview of the data sets to provide empirical motivations for our work.

Chapter 3

Methodology

This chapter describes the data sets used in this thesis. Section 3.1 outlines our trace collection methodology. Section 3.2 presents high-level summary statistics of the collected traces. Section 3.3 describes the extraction of flow statistics from the traces. Section 3.4 describes the method used to establish the base truth of the flow to application mappings for collected traces. An overview of the data sets is provided in Section 3.5. Section 3.6 provides some empirical observations as additional motivation for our work.

3.1 Traces and Collection Methodology

To facilitate our work, we collected traces from the Internet link at the University of Calgary. Depending on the specific subnets traced, the collected traces are categorized as Campus, Residential, and Wireless LAN (WLAN).

Although our classification approach uses only flow statistics, application-layer information is helpful for training the classifiers and required for validating the results. Thus, we needed packet traces that contain relevant application-layer information. In addition, this work needed traffic traces that span an extended period of time to facilitate assessment of the longevity of the classifiers. These requirements introduce several challenges. First, to capture application-layer information necessary for establishing a base truth of the flow to application mapping, packet traces should have the relevant application-layer headers. The application-layer header lengths differ from one application to another. Thus, for simplicity of trace capture, we obtained full payload traces. Second, full payload packet traces require a substantial amount

of storage space. Two additional issues we had to consider were the limited storage space on our network monitor, and that we cannot move a trace off of our monitor quickly enough to sustain continuous full packet tracing.

We have a network monitor deployed on our campus Internet link. Our monitor is configured with two 1.4 GHz Intel Pentium III processors, 2 GB of memory, and 70 GB of disk space for traces. Traffic from the campus Internet connection (a 100Mb/s full-duplex Ethernet link) is forwarded via port mirroring to our monitor over a 1 Gb/s half-duplex Ethernet link.

Initially, we planned to use `tcpdump` to collect the traces, but during testing we found that it dropped many packets (typically more than 1.5%). We instead used a tool called `lindump`, which we found dropped far fewer packets [48].

To address the trace issues identified above while still meeting the requirements of this research, we collected forty-eight 1-hour traces, over a span of six months, of traffic from the campus to outside the campus network, and vice versa only (*i.e.*, traffic to and from the public Internet is captured). Specifically, we collected eight 1-hour traces each week, for five consecutive weeks in the spring of 2006 (April 6 to May 7) and also an additional week in the fall of 2006 (September 28). The traces were collected Thursdays, Fridays, Saturdays, and Sundays from 9-10 am and 9-10 pm on each of these days. Our reasoning for this collection scheme is as follows. First, we expected there to be noticeable differences in usage between the morning and evening hours. Second, we expected there to be noticeable differences in usage between work days and non-work days. Third, the collection period spanned several important transitions in the academic year: the busy final week of the semester (April 6-9), a break before final examinations (April 11-16), the final examination period (April 17-28), the start of summer break for students (April 29-May 7), and a week in the fall semester (September 28-October 1).

Based on these observations, we expected that these traces would capture any

substantial changes that occurred in the traffic during the collection period, while substantially reducing the volume of data we needed to collect. We call this set of forty-eight traces the Campus traces; these contain traffic to and from all academic units on Campus. The network infrastructure uses a signature-based bandwidth management tool to actively limit all identifiable P2P traffic. In addition, user accounts are actively policed for the presence of non-academic content.

The Residential trace was collected on October 20, 2006 from midnight to 10 am from a specific set of subnets corresponding to the student residence network of the university. The student residence network is of interest because it is not actively policed. Instead, there is a "soft" limit on the bandwidth available to each user, and in addition, the total bandwidth usage of this network is limited during work hours.

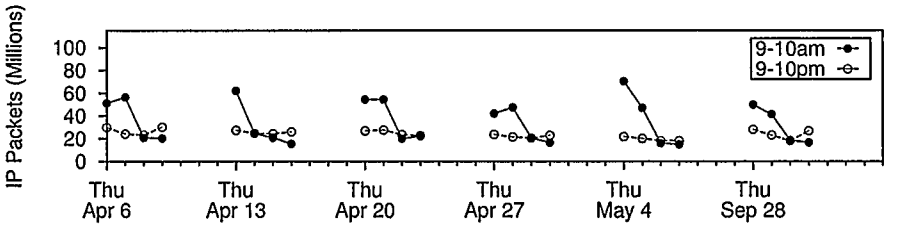
The Wireless Local Area Network (WLAN) trace is a 1-hour trace, collected from the campus WLAN from 9 am to 10 am on September 28, 2006. The WLAN covers many of the buildings on campus, and is open to faculty, staff, and students.

In addition to the above trace, we used data from two other empirical packet traces to analyze the clustering algorithms discussed. One is a publicly available packet trace called Auckland IV[1] that contains the traffic going through the University of Auckland's link to the Internet. We used a subset of the Auckland IV trace from March 16, 2001 at 06:00:00 to March 19, 2001 at 05:59:59. The other trace is an additional full packet trace that we collected from the campus portion of the university's Internet link.

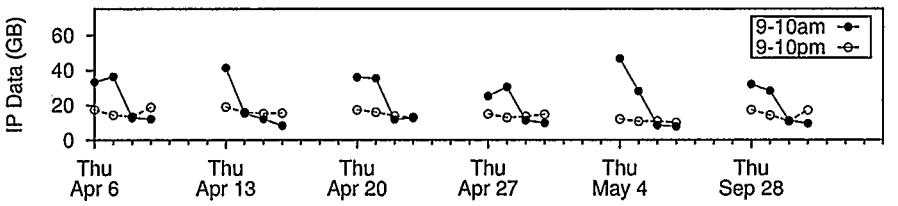
3.2 High-level Statistics of the Traces

Figure 3.1 provides some high-level statistics of the Campus traces. Figures 3.1(a) and (b) reveal some expected trends. First, both the number of packets and volume

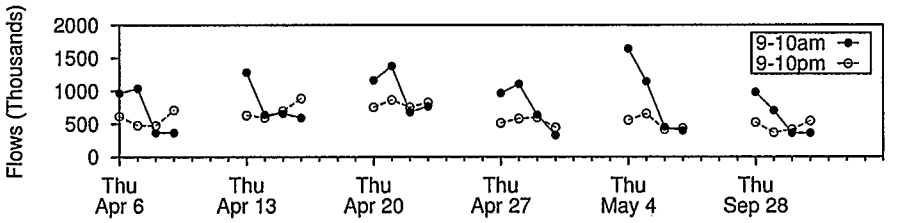
[1]Available at: http://www.wand.net.nz/wand/wits/auck/



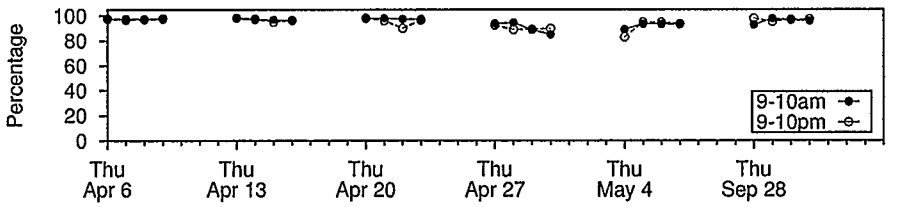
(a) Total IP packets in 1-hour traces



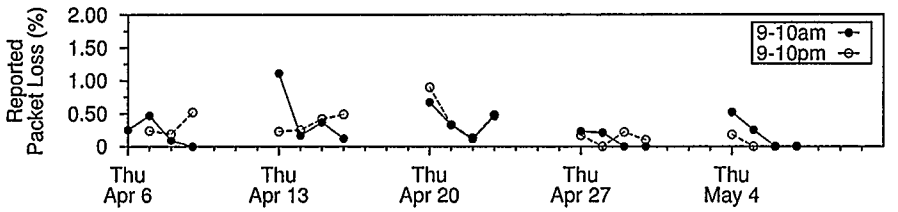
(b) Total IP bytes in 1-hour traces



(c) Total TCP flows in 1-hour traces



(d) Percentage of bytes corresponding to TCP packets



(e) Packet loss rate of the traffic monitor

Figure 3.1: Statistics of the Campus Traces

of data transferred are higher during the morning than in the evening on work days. Second, the number of packets and the volume of data are higher in the evening than in the morning on non-work days. Third, the level of network activity decreased as classes ended and exams began, and decreased further still as the semester ended and students left for the summer.

In total, 1.39 billion IP packets containing 909.2 GB of data were collected. Of this, 89.0% of the packets and 95.1% of the bytes were transferred using TCP and 10.2% of the packets and 4.7% of the bytes were transferred using UDP. Figure 3.1 (c) and (d) show the number of flows and the percentage of bytes in each trace that correspond to TCP packets, respectively.

Figure 3.1 (e) provides the packet loss statistics `lindump` reported for each trace. The reported packet loss was typically below 0.50% with many of the traces being zero. However, the worst packet loss experienced was 1.11% for the 9-10am trace on Thursday April 13.

The 10-hour Residential trace contains 97.5 million IP packets and 58.3 GB of data. Of this, 85.1% of the packets and 83.2% of the bytes are TCP and 14.5% of the packets and 16.6% of the bytes are UDP. The WLAN trace contains 18.4 million IP packets and 11.6 GB of data. Of this, 95.7% of the packets and 98.3% of the bytes are TCP and 1.8% of the packets and 3.6% of the bytes are UDP.

3.3 Flow Characteristics

In this thesis, we focus exclusively on classifying TCP traffic. As discussed above, our traces also had non-TCP traffic (e.g., UDP and ICMP). There are two main reasons for our focus on TCP flows. First, TCP traffic accounts for a significant fraction of the overall traffic. Classifying this traffic accurately allows determination of the robustness of our approach. Second, if flow characteristics from other protocols

were collected, it would likely be advantageous to have a separate classifier for the non-TCP traffic. Classification of UDP traffic is a fruitful avenue for future work.

To collect the statistical characteristics necessary for our classification system, the flows must be identified within the traces. Bro [59], an open source Network Intrusion Detection System, was used for extracting the flow statistics.

The start of a TCP flow is determined by SYN/SYNACK packets being sent. Flows are (typically) terminated when either the FIN or RST packets are received. In addition, we specified in Bro that a flow be considered terminated if it is idle for more than 900 seconds. After determining the flows in the traces, we are able to calculate the required flow statistics. For flows in progress when we started our trace collection we also calculated flow statistics based on the packets we observed in the traces.

3.4 Establishing Base Truth

We established base truth for the traces using an automated process that consists of payload-based signature matching, heuristics, and HTTPS identification. The details of this process are discussed next.

The payload-based classification step uses Bro [59], which has a signature matching engine that generates a match event when the packet payload matches a regular expression specified for a particular rule. We used many of the same methods and signatures described by Sen *et al.* [67] and Karagiannis *et al.* [41], but augmented some of their P2P signatures to account for protocol changes and some new P2P applications.

For the BitTorrent P2P protocol, Sen *et. al* [67] propose using the signature:

```
^19BitTorrent protocol
```

We found that this signature by itself identified the majority of BitTorrent traffic in

```
signature bittorrent_id {
payload /.*(BT_CHOKE|BT_UNCHOKE|BT_UNINTERESTED|BT_INTERESTED
|BT_HAVE| BT_BITFIELD|BT_REQUEST|BT_PIECE|BT_CANCEL
|BT_KEEP_ALIVE|AZ_PEER_EXCHANGE|AZ_HANDSHAKE
|AZ_TORRENT_(SYN|ACK|SESSION_SYN|SESSION_ACK))/
event "BitTorrent"
}
```

Figure 3.2: BitTorrent Payload Signature

our traces. However, we augmented this signature to find some additional BitTorrent traffic in our trace which we were missing with the signature used by Sen *et al.*. Figure 3.2 shows the extra payload signatures we used to identify BitTorrent traffic. The additional signatures for BitTorrent come from the ten primary types of packets that are shared between peers [7]. These packets contain a specific term used to represent the command sent between the peers. These commands are preceded by BT_. In addition, the BitTorrent client Azureus has some of its own specific commands with the AZ_ prefix.

Details about the payload-based signatures for all applications we identified can be found in Appendix A.

For our payload-based classification, if a packet in a flow matches the regular expression pattern specified for a particular application then the entire flow is labelled as being this application. This leaves a possibility for more than one label to be given to a flow in the case where more than one signature was matched. In analyzing the cases where this did occur we found that this typically only happened with flows labelled as HTTP. This occurred with HTTP because some applications such as Gnutella-based P2P applications also use the HTTP protocol. To handle these possible misclassifications, if a flow had already been labelled as HTTP but another signature was matched such as one for Gnutella, then the flow was reclassified as

Gnutella. However, for application labels other than HTTP, once a flow was classified it was not reclassified even if more than one rule was matched.

Some P2P applications are now using encryption. For example, BitTorrent is using a technique called Message Stream Encryption and Protocol Encryption (MSE/PE). The MSE/PE technique uses a Diffie-Hellman exchange that is combined with the *infohash* of the *torrent* to establish the key for the connection [8]. After this exchange has occurred, the clients use RC4 to encrypt the data packets. Some popular BitTorrent clients such as μtorrent and Azureus allow the users to optionally fall back to plaintext if a client does not support or is not using encryption. To identify some of this encrypted P2P traffic, we used a heuristic. Specifically, we maintain a lookup table of (IP address, port number) tuples from flows that have recently (i.e., within 1-hour) been identified as using P2P. If a flow is unlabelled and there is a match in our P2P lookup table, we label it as possible P2P. This mechanism works on the basis that some P2P clients use both encryption and plaintext.

We also analyzed unlabelled traffic on port 443, to establish whether or not this traffic is indeed HTTPS. This verification was done using an experimental version of Bro that has this detection capability. In addition, automated random checks were performed to determine whether or not flows labelled as HTTPS involved at least one host that was a Web server.

The publicly available Auckland IV traces are anonymized, and thus include no payload information. Thus, to determine the flow's "base truth", port numbers are used. While port-based identification is becoming increasingly ineffective we feel this should still provide accurate results for the Auckland IV traces used in this thesis. This is because the emergence of dynamic port numbers in P2P traffic did not happen until late 2002 [14]; the Auckland traces were collected in 2001.

Table 3.1: Application Breakdown (Campus Traces)

Class	Flows	% Flows	Bytes	% Bytes
HTTP	9,213,424	39.5%	334.4 GB	38.7%
P2P	620,692	2.7%	310.9 GB	36.0%
EMAIL	1,123,987	4.8%	42.5 GB	4.9%
FTP	23,571	0.1%	20.3 GB	2.3%
P2P Possible	35,620	0.2%	12.3 GB	1.4%
STREAMING	3,396	0.0%	7.4 GB	0.9%
DATABASE	3,057,362	13.1%	3.0 GB	0.3%
CHAT	26,869	0.1%	1.0 GB	0.1%
OTHER	51,298	0.2%	32.1 GB	3.7%
UNKNOWN	990,492	4.2%	70.1 GB	8.1%
UNKNOWN (443)	1,409,707	6.0%	29.7 GB	3.4%
UNKNOWN (NP)	6,765,214	29.0%	1.0 GB	0.1%
Total	23,321,632	100.0%	864.6 GB	100.0%

3.5 Overview of the Data Sets

Table 3.1 summarizes the applications found in the forty-eight 1-hour Campus traces. Application breakdowns for the 10-hour Residential trace, the 1-hour WLAN trace and the Auckland IV trace are shown in Table 3.2, Table 3.3, and Table 3.4, respectively.

Over 29 different applications were identified. These applications include: BB, BitTorrent, DirectConnect, eDonkey, FTP, Gnutella-based P2P programs (e.g., LimeWire, BearShare, Gnucleus, Morpheus, FreeWire), GoToMyPC, HTTP, ICQ, IDENT, IMAP, IMAP SSL, JetDirect, KaZaA, MySQL, MSSQL, MSN Messenger, MSN Web Cam, NNTP, POP3, POP3 SSL, RTSP, Samba, SIP, SMTP, SOAP, SpamAssassin, SSH, SSL, VNC, and Z3950 Client. To simplify the presentation, we group the applications by category. For example, the P2P category includes all identified P2P traffic from protocols including BitTorrent, Gnutella, and KaZaA. P2P flows identified using heuristics are labelled P2P Possible. The OTHER category constitutes various applications that were identified but did not belong to a larger group and did not

Table 3.2: Application Breakdown (Residential Trace)

Class	Flows	% Flows	Bytes	% Bytes
P2P	297,781	17.6%	38.52 GB	79.3%
HTTP	118,485	7.0%	3.37 GB	6.9%
P2P Possible	39,943	2.4%	0.34 GB	0.7%
EMAIL	1,159	0.1%	0.12 GB	0.2%
STREAMING	29	0.0%	0.07 GB	0.1%
CHAT	1,207	0.1%	0.05 GB	0.1%
OTHER	190	0.0%	0.03 GB	0.1%
UNKNOWN	91,275	5.4%	5.88 GB	12.1%
UNKNOWN (443)	4,833	0.3%	0.06 GB	0.1%
UNKNOWN (NP)	1,135,242	67.2%	0.13 GB	0.3%
Total	1,690,144	100.0%	48.56 GB	100.0%

Table 3.3: Application Breakdown (WLAN Trace)

Class	Flows	% Flows	Bytes	% Bytes
P2P	61,603	15.9%	6.90 GB	60.3%
HTTP	145,177	37.5%	2.94 GB	25.7%
P2P Possible	7,842	2.0%	0.13 GB	1.2%
CHAT	2,928	0.8%	0.05 GB	0.5%
EMAIL	695	0.2%	0.02 GB	0.1%
FTP	157	0.0%	0.00 GB	0.0%
STREAMING	13	0.0%	0.00 GB	0.0%
OTHER	374	0.1%	0.01 GB	0.1%
UNKNOWN	16,100	4.2%	1.16 GB	10.1%
UNKNOWN (443)	8,581	2.2%	0.22 GB	2.0%
UNKNOWN (NP)	143,631	37.1%	0.02 GB	0.1%
Total	387,101	100.0%	11.4 GB	100.0%

Table 3.4: Application Breakdown (Auckland IV Trace)

Class	Flows	% Flows	Bytes	% Bytes
HTTP	3,092,009	81.2%	36.24 GB	68.6%
DNS	75,513	2.0%	0.07 GB	0.1%
SOCKS	69,161	1.8%	0.24 GB	0.4%
IRC	53,446	1.4%	0.01 GB	0.0%
FTP (control)	50,474	1.3%	0.03 GB	0.1%
POP3	37,091	1.0%	0.22 GB	0.4%
Gnutella	10,784	0.3%	0.51 GB	1.0%
NNTP	9,442	0.2%	1.25 GB	2.4%
FTP (data)	5,018	0.1%	2.13 GB	4.0%
UNKNOWN	404,501	10.6%	12.14 GB	23.0%
Total	3,807,439	100.0%	52.83 GB	100.0%

account for a significant proportion of flows. The tables also list three categories of UNKNOWN flows. There are UNKNOWN (NP) flows that have no payloads. Most of these are failed TCP connections, while some are port scans. The UNKNOWNS (443) are flows on port 443; these are likely to be HTTPS traffic. The third category is simply labelled as UNKNOWN to reflect the fact that we have not identified the applications that generated this traffic. The unknown flows are not used in our analysis. General observations from these data sets follow.

Figure 3.3 shows the breakdown of the traffic for different applications in the Campus traces. For clarity, this figure reports HTTP, P2P, OTHERS, and UNKNOWN, wherein the OTHERS category includes all classified traffic that is not HTTP or P2P. Also we have excluded flows that do not have any payloads. In this figure the diurnal patterns of the traffic can be seen. During the weekdays there is more traffic than the weekends and during the mornings there is more traffic than at night (note April 14 was a holiday). The large increase of UNKNOWNS on May 4 at 9 am is due to a port scan. One IP address made 621,429 connections to one of the University of Calgary's subnets scanning every port from 1 to 1024. These were single packets sent of only 60 bytes and so there is not a similar spike in the bytes

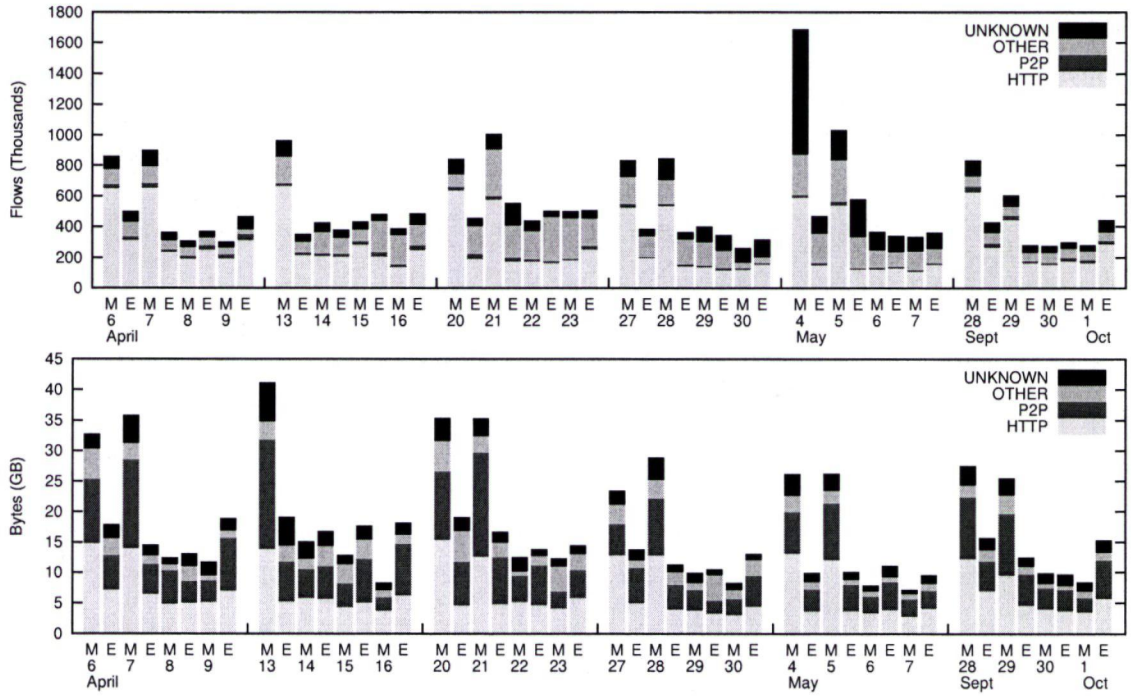


Figure 3.3: Application Class Breakdown of Campus Trace (M: Morning; E: Evening)

graph for the same trace.

On the campus network (Table 3.1), HTTP, DATABASE, and EMAIL traffic contribute a significant portion of the total flows. On this network, P2P contributes only 2.7% of the flows. However, P2P still accounts for a considerable portion, approximately 36%, of the bytes. In contrast, the traffic from the residential network (Table 3.2) exhibits comparatively less diversity in the usage of applications, with HTTP and P2P being the dominant applications. In the 10-hour Residential trace, P2P has a significant presence, both in terms of number of flows and number of bytes. We attribute this difference, somewhat speculatively, to the network use policies in place and the profile of the network users. As mentioned earlier, the campus network is used by faculty, staff, and students, and is actively regulated for non-academic content. Furthermore, the network infrastructure uses signature-based

Table 3.5: P2P Breakdown (Residential Trace)

Application	Flows	% Flows	Bytes	% Bytes
BitTorrent	286,794	96.3%	22.00 GB	57.1%
Gnutella-based	10,066	3.4%	16.47 GB	42.7%
eDonkey	921	0.3%	0.05 GB	1.4%
Other	161	0.1%	0.01 GB	0.4%
Total	297,942	100.0%	38.5 GB	100.0%

Table 3.6: P2P Port Usage (Residential Trace)

Application	Non-Standard Port (% Flows)	Non-Standard Port (% Bytes)
BitTorrent	91.7%	84.0%
Gnutella-based	82.1%	99.1%
eDonkey/eMule	89.1%	99.0%

identification to severely throttle P2P traffic. In contrast, the residential network is used exclusively by students, is not actively policed, and only applies a soft limit on the bandwidth available to each user.

Table 3.5 shows that BitTorrent and Gnutella-based P2P applications such as BearShare, LimeWire, Morpheus, and Gnucleus are prevalent on the residential network. KaZaA was hardly seen in the traces.

3.6 Empirical Motivation for this Research

We supplement our trace data analysis with three empirical observations that further motivate our traffic classification work. These observations concern port numbers, amount of variable-length offset bytes, and encryption.

Table 3.6 shows that use of non-standard ports is prevalent[2]. Approximately 92% of the BitTorrent flows used non-standard ports. This contrasts starkly with the study by Sen *et al.* [67] in 2004 where they found only 1% of the BitTorrent flows used

[2]Default port numbers used were BitTorrent (6881-6889,32459), Gnutella-based (6346), eDonkey/eMule (4661,4662,4711,4712).

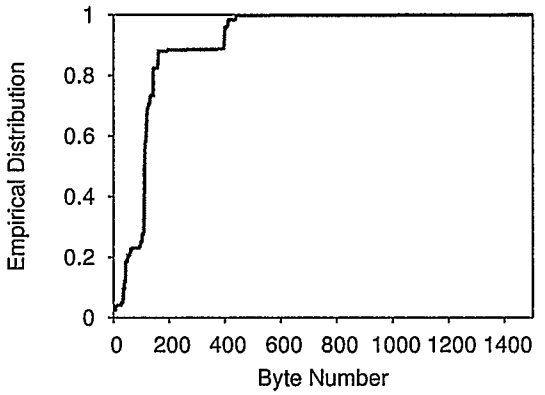


Figure 3.4: Variable-Length Offset of Gnutella Signature

non-standard ports. This provides further evidence on the declining effectiveness of port-based classification.

Figure 3.4 shows the empirical distribution of variable-length offsets in Gnutella before the characteristic payload signature is found. We found that signature matching using only the initial 64 bytes of the payload bytes will allow approximately only 25% of the Gnutella flows to be identified. Over 400 payload bytes of each packet would need to be captured to increase the number identified to 90%. Furthermore, an application could easily make the length greater than 400 bytes if it helped avoid detection.

Finally, our base truth establishment process indicates the presence of encrypted traffic, most of which is likely to be from P2P applications. We have labelled these as P2P Possible in Tables 3.1-3.3. We believe that as P2P applications evolve, encryption will become the norm, and in that case, packet inspection techniques will likely fail.

3.7 Summary

This chapter discussed the data sets used in this thesis. The majority of the traces used in this study were collected from the Internet link at the University of Calgary. These University of Calgary traces were collected over a 6-month time period. In total, over 1 terabyte of data was collected and we identified flows based on their 5-tuple. We established a "base truth" classification for our traces using a payload-based approach. We present a breakdown of the traffic based on the application types we identified. In addition, we confirm that port-based classification is ineffective and provide additional empirical motivation for our proposed approach.

The next chapter presents our semi-supervised classification approach.

Chapter 4

Semi-Supervised Classification Framework

Many of today's network monitoring solutions operate on the notion of network *flows*. A flow is defined as a series of packet exchanges between two hosts, identifiable by the 5-tuple (source address, source port, destination address, destination port, transport protocol), with flow termination determined by an assumed timeout or by distinct flow termination semantics. For each flow, network monitors can record statistics such as duration, bytes transferred, mean packet interarrival time, and mean packet size. This chapter outlines our classification method that can map flows (characterised by a vector of flow statistics) to applications (or traffic classes), with high accuracy and in realtime.

4.1 Terminology

We now introduce notations and terminology to describe the problem formally. Let $\mathbf{X} = \{X_1, \cdots, X_N\}$ be a set of flows. A flow instance X_i is characterised by a vector of attribute values, $\mathbf{X_i} = \{X_{ij} | 1 \leq j \leq m\}$, where m is the number of attributes, and X_{ij} is the value of the j^{th} attribute of the i^{th} flow. In the traffic classification context, examples of attributes include flow statistics such as duration, bytes transferred, and total number of packets. The terms *attributes* and *features* are used interchangeably in the machine learning literature, and often $\mathbf{X_i}$ is referred to as a feature vector. Also, let $\mathbf{Y} = \{Y_1, \cdots, Y_q\}$ be the set of traffic classes, where q is the number of classes of interest. The Y_i's can be classes such as "HTTP", "Streaming", and "Peer-to-Peer". Our goal, therefore, is to learn a mapping from a m-dimensional variable X to Y. This mapping forms the basis for classification models, also referred to as

classifiers in the machine learning literature.

Traditional learning methods of classifiers use a training data set that consists of N tuples $(\mathbf{X_i}, \mathbf{Y_i})$ and learn a mapping $f(\mathbf{X}) \rightarrow \mathbf{Y}$. The goal is to find a mapping that (correctly) generalizes to previously unseen examples. Such learning methods are referred to as *supervised* learning methods [23]. Supervised machine learning techniques have previously been applied for classifying network flows. Roughan *et al.* [65] classified flows into four predetermined traffic classes (interactive, bulk data transfer, streaming, and transactional) using the Nearest Neighbor and the Linear Discriminate Analysis classification techniques. Moore *et al.* [53] evaluated the suitability of a Naïve Bayes classifier for the Internet traffic classification problem. Recently, Williams *et al.* [72] presented a preliminary comparison of five supervised learning algorithms.

In designing our classification method, we are interested in overcoming two main challenges faced by supervised techniques:

1. Labelled examples are scarce and difficult to obtain. With few labelled examples, traditional supervised learning methods often produce classifiers that do not generalize well to previously unseen flows.

2. Not all types of applications generating flows are known *a priori*, and new ones may appear over time. Supervised methods force a mapping of each flow into one of q known classes, without the ability to detect new types of flows.

To address these challenges, we designed a method that combines unsupervised and supervised methods. Our classification method consists of two steps. We first employ a machine learning approach called *clustering* [23] to partition a training data set that consists of scarce labelled flows combined with abundant unlabelled flows. Clustering partitions the training data set into disjoint groups ("clusters") such that flows within a group are similar to each other whereas flows in different groups

are as different as possible. Second, we use the available labelled flows to obtain a mapping from the clusters to the different known q classes (Y). This step also allows some clusters to remain unmapped, accounting for possible flows that have no known labels. The result of the learning is a set of clusters, some mapped to the different flow types. This method, referred to as *semi-supervised* learning [3, 10, 16], has received considerable attention, recently, in the machine learning community.

We note that our application of semi-supervised learning is novel in that we leave some of the clusters unlabelled. This is different from the traditional application of semi-supervised learning; in the traditional application of this approach, all classes are known a priori, and unlabelled flows are used to improve precision of the classifier. In the traffic classification problem, however, not all classes are known a priori, and thus, we use the unlabelled clusters to represent new or unknown applications. In effect, unlabelled flows are used to improve precision and handle unknown applications. The remainder of this chapter discusses the details of the classification method.

4.2 Model Building: Clustering

The first step in training our classifier is to leverage all available training flows and group them into clusters. In the machine learning paradigm, clustering is an example of an *unsupervised* learning algorithm [23] because the partitioning of the flows in the training data is guided only by the similarity between the flows and not by any predetermined labelling of the flows. A key benefit of the unsupervised learning approach is the ability to identify hidden patterns. For example, new applications as well as changed behaviour of existing applications can be identified by examining flows that form a new cluster.

Clustering algorithms use a measure $d(\mathbf{x_i}, \mathbf{x_j})$ of similarity between feature vectors

$\mathbf{x_i}$ and $\mathbf{x_j}$, and find a partition that attempts to place similar examples in the same cluster, and dissimilar examples in different clusters. There are various similarity metrics that can be used. Without loss of generality, in this thesis we use the Euclidean distance as the similarity measure:

$$d(\mathbf{x_i}, \mathbf{x_j}) = \left[\sum_{k=1}^{m}(x_{ik} - x_{jk})^2\right]^{1/2}. \tag{4.1}$$

There are many different clustering algorithms in the machine learning literature. Although the proposed classification approach is *not* specific to any particular clustering algorithm, our offline and realtime implementations use the K-Means algorithm [23]. In Chapter 5, we analyze three different clustering algorithms namely, K-Means, DBSCAN, and AutoClass; this analysis provides insight into the models produced by these clustering algorithms and the type of clusters formed, and furthermore, provides rationale for our choice of using the K-Means algorithm.

4.3 Classifier: Mapping Clusters to Applications

The output of the K-Means clustering algorithm is a set of clusters, represented by their centroids, γ_k. Given a flow feature vector $\mathbf{x}$, we assign it to one of the clusters by finding the nearest centroid to $\mathbf{x}$, using:

$$C_k = \arg\min_k d(\mathbf{x}, \gamma_k), \tag{4.2}$$

where $d(\cdot, \cdot)$ is the distance metric chosen in the clustering step. For K-Means with Euclidean distance, this step amounts to the maximum likelihood cluster assignment solution. In the machine learning literature, this form of classification is known as a "distance-based" classifier [26].

However, knowing to which cluster a flow feature vector most likely belongs does not provide the actual classification to one of the application types. Therefore, we

need a mechanism to map the clusters found by the clustering algorithm to the different application types.

We use a probabilistic assignment to find the mapping from clusters to labels: $P(Y = y_j|C_k)$, where $j = 1, ..., q$ (q being number of application types) and $k = 1, ..., K$ (K being the number of clusters). To estimate these probabilities, we use the set of flows in our training data that are labelled to different applications $(\mathbf{x_i}, \mathbf{y_i})$, $i = 1, ..., L$, where L is the total number of different labelled applications. $P(Y = y_j|C_k)$ is then estimated by the maximum likelihood estimate, $\frac{n_{jk}}{n_k}$, where n_{jk} is the number of flows that were assigned to cluster k with label j, and n_k is the total number of (labelled) flows that were assigned to cluster k. To complete the mapping, clusters that do not have any labelled examples assigned to them are defined as "Unknown" application types, thus allowing the representation of previously unidentified application types.

Finally, the decision function for classifying a flow feature vector $\mathbf{x}$ is the maximum *a posterior* decision function:

$$y = \arg \max_{y_1,...,y_q} (P(y_j|C_k)), \tag{4.3}$$

where C_k is the nearest cluster to $\mathbf{x}$, as obtained from Eq. 4.2. Our approach uses *hard* clustering. However, labelling using *soft* clusters can easily be accommodated into our framework. For instance, the confidence of a flow's label could be based on $P(y_j|C_k)$ and labels below a certain threshold could be considered "Unknown". Exploration of soft clustering and its potential benefits is left for future work.

4.4 Summary

In this chapter, we formally described the proposed semi-supervised classification framework. The classification framework contains two primary steps: model build-

ing and classification. This framework forms the basis of our offline and realtime classification systems discussed in Chapter 6.

The next chapter presents a description and analysis of clustering algorithms that can be used in the aforementioned classification framework.

Chapter 5

Clustering Analysis

This chapter describes and analyzes the potential of clustering algorithms for use in our semi-supervised approach to traffic classification. This analysis provides insight into the clustering models used by the semi-supervised classification framework presented in the preceding chapter. Section 5.1 describes the three clustering algorithms considered. Section 5.2 presents analysis where the algorithms are compared based on their ability to generate clusters that consist primarily of a single application type. In Section 5.3, we describe how a classifier for each clustering algorithm can be developed for our framework and why K-Means is selected as the clustering algorithm used to build our offline and realtime classifiers.

5.1 Clustering Algorithms

We restrict our attention to three popular clustering algorithms, namely K-Means [39], DBSCAN [33], and AutoClass [11]. Each of these algorithms is based on a different clustering principle: K-Means is partition-based, DBSCAN algorithm is density-based, and AutoClass is probabilistic model-based. The data mining literature contains many well-understood clustering algorithms developed during the past three decades [23, 39, 73]. A possible future work direction is to consider other clustering algorithms.

5.1.1 K-Means

There are a variety of partition-based clustering algorithms available [23,39]. The K-Means algorithm [23] , shown on page 56, is selected because it is one of the quickest

Algorithm 1 The K-Means Algorithm

Input : Training Data Set $D = \{x_1, \cdots, x_n\}$ and number of clusters K.
Output: Clusters $C_1, \cdots, C_K$ such that $D = \cup_{i=1}^{K} C_i$ and $C_i \cap C_j = \emptyset, \forall i \neq j$.
for *each* k **do**
  let γ_k be a randomly chosen object from D;
end
repeat
  for *each object* $x_i, i \in \{1, \cdots, n\}$ **do**
    $k = \arg\min_j d(x_i, \gamma_j)$ assign x_i to cluster C_k
  end
  for *each cluster* C_k **do**
    compute new cluster centroid γ_k
  end
until *convergence criterion satisfied* ;

and simplest. The K-Means algorithm partitions the feature vectors in the training data set into a fixed number of spherical-shaped clusters by minimizing the total mean square error between feature vectors and the cluster centroids. Starting with an initial partition (random or other), the algorithm iteratively assigns each vector to the cluster whose centroid is nearest, and recalculates the centroids based on the new assignments. This process continues until membership within clusters stabilizes. The complexity of the algorithm is $O(lKn)$ where l is the number of iterations [23]. For the data sets used in this thesis, the algorithm converges within a few iterations.

5.1.2 DBSCAN Clustering

Density-based algorithms regard clusters as dense areas of objects that are separated by less dense areas [2, 33]. Unlike algorithms such as K-Means, these algorithms are not limited to finding spherical shaped clusters but can find clusters of arbitrary shapes. We choose the Density Based Spatial Clustering of Applications with Noise (DBSCAN) algorithm as a representative of density-based algorithms [33]. The DBSCAN results are obtained from the implementation available in the WEKA software suite [73].

The DBSCAN algorithm takes two inputs: epsilon (eps) and minimum number of points (minPts). The algorithm uses these input parameters to define the concepts of eps-neighbourhood, core object, density-reachability, and density-connectivity. The eps-neighbourhood of an object p is defined as the set of all objects that are within eps distance of p. An object q is described as a core object if the number of objects within its eps-neighbourhood is at least minPts. An object p is said to be density-reachable from a core object q provided there exists a finite sequence of core objects between p and q, with each of these core objects being in the eps-neighbourhood of its immediate predecessor. Finally, objects p and q are said to be density-connected if an object o exists from which both p and q are density-reachable.

The DBSCAN algorithm defines a cluster as the set of objects in a data set that are density-connected to a particular core object. Any object that is not part of a cluster is categorized as noise. This is in contrast to the K-Means and AutoClass algorithms which give every object a cluster assignment.

The DBSCAN algorithm works as follows. Initially, all objects in the data set are assumed to be unassigned. The DBSCAN algorithm chooses an arbitrary unassigned object p from the data set. If DBSCAN finds p is a core object, it finds all the density-connected objects based on eps and minPts. It assigns all these objects as being from a new cluster. If DBSCAN finds p to be not a core object, then p is considered to be noise and the DBSCAN algorithm moves onto the next unassigned object in the data set. Once every object is assigned, the algorithm stops.

5.1.3 AutoClass

Probabilistic model-based clustering is another powerful clustering technique. We use an implementation of a probabilistic model-based clustering technique called AutoClass [11]. This algorithm allows for the automatic selection of the number of clusters and the soft clustering of the data. Soft clusters allow the data objects to

be fractionally assigned to more than one cluster. In the analysis in Section 5.2, we use the most probable assignment as the object's assignment.

To build the probabilistic model, the clustering algorithm must determine the number of clusters and the parameters that govern the distinct probability distributions of each cluster. To accomplish this task, AutoClass uses the Expectation Maximization (EM) algorithm [21].

The EM algorithm has two steps: an expectation step and a maximization step. The initial expectation step guesses what the parameters are using pseudo-random numbers. Then in the maximization step, the mean and variance are used to re-estimate the parameters continually until they converge to a local maximum. These local maxima are recorded and the EM process is repeated. This process continues until enough samples of the parameters have been found (we use 200 cycles in our experimental results).

AutoClass uses a Bayesian information criterion (BIC) to determine the best set of parameters to use for the probabilistic model. BIC is based on intra-cluster similarity and inter-cluster dissimilarity. Also, BIC penalizes the score of models with more clusters to minimize potential over-fitting.

5.2 Clustering Evaluation

In this section, the overall effectiveness of each clustering algorithm is evaluated. The algorithms are compared based on their ability to generate clusters that have a high predictive power of a single application type. We believe that in order to build an accurate classifier, a good classification model must be used.

The clustering analysis is done using two empirical packet traces: the Auckland IV trace (discussed in Section 3.1) and a campus trace collected on March 10, 2006 from 1 to 2pm. We refer to this particular campus trace as the Calgary trace for the

remainder of this chapter.

The majority of flows in both traces carry HTTP traffic. This unequal traffic sample does not allow for the fair testing of different traffic classes (i.e., HTTP would dominate the data set such that producing a single cluster containing all flows would still give a high cluster purity). To address this problem, the Auckland data sets used for this test consist of 1000 random samples of each of the following traffic classes: DNS, FTP (control), FTP (data), HTTP, IRC, Gnutella, NNTP, POP3, and SOCKS. The Calgary data sets used 2000 random samples of each of the following traffic classes: HTTP, P2P, SMTP, and POP3. The size of the data sets were limited to 8000 flows because this was the upper bound that the AutoClass algorithm could cluster within a reasonable amount of time (4-10 hours). To achieve greater confidence in our results, we repeated our tests using 10 different data sets generated from each trace. We report the minimum, maximum, and average results from the data sets of each trace.

The flow statistics considered in this test include: total number of packets, mean packet size, mean payload size excluding headers, number of bytes transfered (in each direction and combined), and mean inter-arrival time of packets. Our decision to use these features is based primarily on the previous work done by Zander *et al.* [75]. Due to the heavy-tailed distribution of many of the features and our use of Euclidean distance as our similarity metric, we found that the logarithms of the features gives much better results for all the clustering algorithms [60, 73]. We undertake an extensive feature selection process later in Section 6.1.3 when evaluating the design of our offline classifier.

5.2.1 Algorithm Effectiveness

The clustering algorithms are evaluated using a metric called *cluster purity*. This cluster purity measurement determines how well the clustering algorithm is able to

create clusters that contain only a single traffic class. Note that for these experiments the traffic class a flow belongs to is known. We are interested in determining whether flows of the same type form distinct clusters.

The traffic class that makes up the majority of the flows in a cluster is used to label the cluster. When the flow label matches with its corresponding cluster label, we have a True Positive (TP); otherwise, a False Positive (FP) occurs. Any flow that has not been assigned to a cluster is labelled as noise. The cluster purity is calculated as:

$$cluster\ purity = \frac{\sum TP\ for\ all\ clusters}{total\ number\ of\ flows}. \tag{5.1}$$

In the following subsections, the effectiveness of the K-Means, DBSCAN, and AutoClass algorithms are presented.

5.2.2 K-Means Clustering

The K-Means algorithm has an input parameter of K. This input parameter as mentioned in Section 5.1.1, is the number of disjoint partitions generated by K-Means. In our data sets, we would expect there would be at least one cluster for each traffic class. In addition, due to the diversity of the traffic in some classes such as HTTP (e.g., browsing, bulk download, streaming) we would expect even more clusters to be formed. Therefore, based on this, the K-Means algorithm was evaluated with K initially being 10 and K being incremented by 10 for each subsequent clustering. The minimum, maximum, and average results for the K-Means clustering algorithm are shown in Figure 5.1.

Initially, when the number of clusters is small the cluster purity of K-Means is approximately 49% for the Auckland IV data sets and 67% for the Calgary data sets. The cluster purity steadily improves as the number of clusters increases. This continues until K is around 100 with the cluster purity being 79% and 84% on average, for the Auckland IV and Calgary data sets, respectively. At this point,

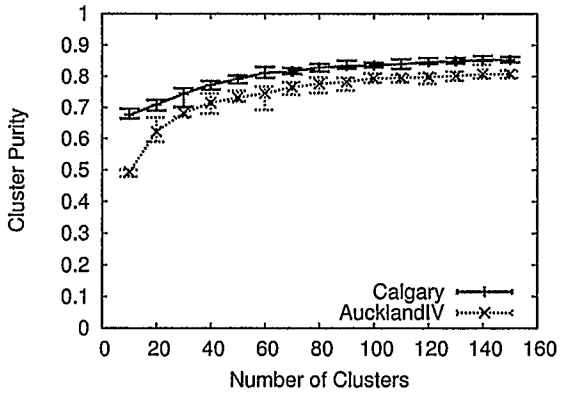


Figure 5.1: Purity using K-Means

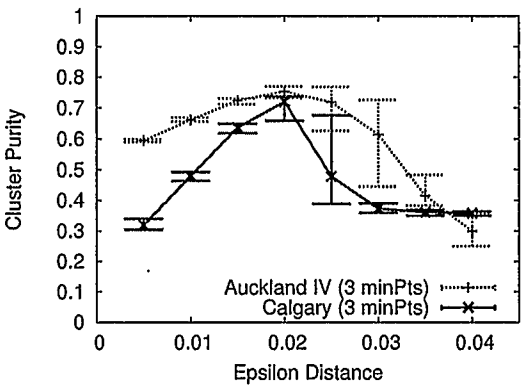


Figure 5.2: Accuracy using DBSCAN

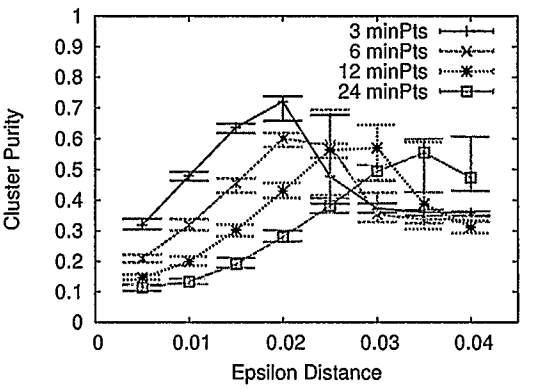


Figure 5.3: Parametrization of DBSCAN

the improvement is much more gradual with the cluster purity only improving by an additional 1.0% when K is 150 in both data sets. When K is greater than 150, the improvement is further diminished with the cluster purity improving to the high 80% range when K is 500. However, large values of K increase the likelihood of over-fitting.

5.2.3 DBSCAN Clustering

The purity results for the DBSCAN algorithm are presented in Figure 5.2. Recall that DBSCAN has two input parameters (minPts, eps). We varied these parameters, and in Figure 5.2 report results for the combination that produces the best clustering results. The values used for minPts were tested between 3 and 24. The eps distance was tested from 0.005 to 0.040. Figure 5.3 presents results for different combinations of (minPts, eps) values for the Calgary data sets. As may be expected, when the minPts was 3 better results were produced than when the minPts was 24 because smaller clusters are formed. The additional clusters found using three minPts were typically small clusters containing only 3 to 5 flows.

When using minPts equal to 3 while varying the eps distance between 0.005 and 0.020 (see Figure 5.2), the DBSCAN algorithm improved its cluster purity from 59.5% to 75.6% for the Auckland IV data sets. For the Calgary data sets, the DBSCAN algorithm improved its cluster purity from 32.0% to 72.0% as the eps distance was varied with these same values. The cluster purity for eps distances greater than 0.020 decreased significantly as the distance increased. Our analysis indicates that this large decrease occurs because the clusters of different traffic classes merge into a single large cluster. We found that this larger cluster was for flows with few packets, few bytes transfered, and short durations. This cluster contained typically equal amounts of P2P, POP3, and SMTP flows. Many of the SMTP flows were for emails with rejected recipient addresses and connections immediately closed after connecting to the SMTP server. For POP3, many of the flows contained instances where no email was in the users mailbox. Gnutella clients attempting to connect to a remote node and having its "GNUTELLA CONNECT" packets rejected accounted for most of the P2P flows.

Table 5.1: Purity using AutoClass

Data Set	Average	Minimum	Maximum
Auckland IV	92.4%	91.5%	93.5%
Calgary	88.7%	86.6%	90.0%

5.2.4 AutoClass Clustering

The results for the AutoClass algorithm are shown in Table 5.1. For this algorithm, the number of clusters and the cluster parameters are automatically determined. Overall, the AutoClass algorithm has the highest purity. On average, AutoClass is 92.4% and 88.7% pure in the Auckland IV and Calgary data sets, respectively. AutoClass produces an average of 167 clusters for the Auckland IV data sets, and 247 clusters for the Calgary data sets.

5.2.5 Discussion

For the traffic classification problem, the number of clusters produced by a clustering algorithm is also an important consideration as more clusters increase the computational processing of the classifier. The reason being that once the clustering is complete, each of the clusters must be labelled. Minimizing the number of clusters is also cost effective during the classification stage.

One way of reducing the number of clusters to label is by evaluating the clusters with many flows in them. For example, if a clustering algorithm with high accuracy places the majority of the flows in a small subset of the clusters, then by analyzing only this subset a majority of the flows can be identified. Figure 5.4 shows the percentage of flows represented as the percentage of clusters increases, using the Auckland IV data sets. In this evaluation, the K-Means algorithm had 100 for K. For the DBSCAN and AutoClass algorithms, the number of clusters cannot be set. DBSCAN uses 0.02 for eps, 3 for minPts, and has, on average, 190 clusters. We selected this point because it gave the best cluster purity for DBSCAN. AutoClass

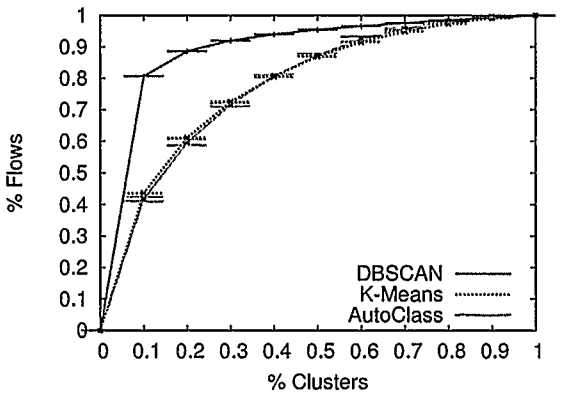


Figure 5.4: CDF of Cluster Weights

has, on average, 167 clusters.

As seen in Figure 5.4, both K-Means and AutoClass have more evenly distributed clusters than DBSCAN. The 15 largest clusters produced by K-Means contain only 50% of the flows. In contrast, for the DBSCAN algorithm the five largest clusters contain over 50% of the flows in the data sets. These five clusters identified 75.4% of the NNTP, POP3, SOCKS, DNS, and IRC flows with a 97.6% cluster purity. These results are unexpected when considering that by only looking at five of the 190 clusters, one can identify a significant portion of traffic. (Qualitatively similar results were obtained for the Calgary data sets.) However, the DBSCAN algorithm is the only algorithm considered that can create non-spherical shaped clusters. This allows these larger clusters to form. The K-Means and AutoClass algorithms can approximate these same areas using multiple clusters which explains why more clusters are needed with K-Means to represent 50% of the flows.

Another noteworthy difference among the clustering algorithms is the time required to build the models. On average, to build the models, the K-Means algorithm took less than 1 minute, the DBSCAN algorithm took 3 minutes, and the AutoClass

algorithm took 4.5 hours. Clearly, the model building phase of AutoClass is time consuming. We believe this may deter system developers from using this algorithm even if the frequency of retraining for the model is low.

5.3 Designing an Efficient and Effective Classifier

The semi-supervised classification framework proposed in this thesis uses the K-Means algorithm for its offline and realtime implementations. One reason for this choice is that the more complex clustering algorithms required significantly longer learning time than K-Means (e.g., hours versus minutes). We find that with K-Means, large data sets can be leveraged to provide many benefits such as improving classifier precision and allowing the classifier to handle unknown applications. That notwithstanding, it is possible to design classifiers for the DBSCAN and AutoClass algorithms as described below. During the course of this research work, we did in fact build each of these classifiers.

A classifier for the DBSCAN algorithm can be developed using an approach similar to the distance-based approach used for K-Means (discussed in Section 4.3). However, the non-spherical shapes that the DBSCAN clusters can form cannot be adequately represented by only calculating a single centroid of the cluster, thus necessitating additional points to represent the cluster. To classify a new flow, the distance between each of the points representing a cluster and the candidate flow is calculated. A flow is assigned to the cluster that has the lowest distance measurement overall to any of the cluster's points. This DBSCAN classifier would be much slower than K-Means because of these additional calculations per cluster.

AutoClass predicts the cluster to which a new flow belongs using the probabilistic model developed during clustering. The probability of the new flow belonging to each cluster is calculated. The cluster assignment can then be made using the most likely

cluster. Note, that using a probabilistic assignment like this could be one method of incorporating *soft clustering* into our framework mentioned in Section 4.2.

In addition to the fast clustering possible using K-Means, the simplicity and ease of implementation of this algorithm prompted its use in the offline/realtime systems developed in this work. The K-Means classifier has the least amount of computational overhead because the data structures representing the clusters allow fast computations of distance (i.e., $d(\mathbf{x_i}, \mathbf{x_j})$ in Section 4.2). For example, we found the DBSCAN algorithm would have required upto 10 times as many points to represent its clusters as K-Means. Also, K-Means can generate clusters that largely consist of a single application type. The other clustering algorithms investigated in some cases provided more pure clusters, however, once converted into classifiers the difference in classification accuracy was negligible. In the case of DBSCAN, we found in some preliminary tests that the flows discarded as noise significantly impacted the accuracy of the classifier. Finally, the K-Means algorithm converges to a well-understood probabilistic model: the Gauss Mixture Model [23]. Exploration of other clustering algorithms for use with the semi-supervised method is left for future work.

5.4 Summary

In this chapter, we described and analyzed several clustering algorithms for potential use in the semi-supervised framework proposed in this thesis. We found that the clustering algorithms largely produce clusters that have a high predictive power of a single traffic class. The results showed AutoClass produced the most pure clusters. However, we also found that the K-Means algorithm is a more suitable choice for us to use as a classifier; K-Means clusters are only marginally less pure, but K-Means is much faster at clustering flows. This allows substantially larger training data sets to be leveraged in the rest of our work.

In the next chapter, we present classification results using the offline and realtime classifiers developed based on the K-Means algorithm.

Chapter 6

Offline and Realtime Classification

This chapter presents the offline and realtime classification systems. Section 6.1 evaluates the design alternatives for offline classification, and Section 6.2 introduces and evaluates the realtime classifier. The history of the traffic classification problem, the longevity of the classifier, and the detection of when the classifier requires retraining are discussed in Section 6.3.

6.1 Offline Classification

We implemented a prototype offline classification system, incorporating both steps of the classification methodology, in approximately 3,000 lines of C++ code. In this section, we discuss the design considerations that affect the performance of the classifier. The design considerations are:

- Composition of the training data set: There are two related considerations, the fraction of the training flows that are labelled, and the methodology used to select flows for the training; these issues are discussed in Sections 6.1.1 and 6.1.2, respectively. Unless stated otherwise, we assume that all training flows are labelled.

- The features used to characterise the flows: Feature selection is discussed in Section 6.1.3.

- The number of clusters K generated in the clustering step of the classification method: This parameter can be used to tune our classifier to achieve better accuracy, however, at the cost of additional computation for the classifier. Unless

stated otherwise, we assume $K = 400$. We explore this factor in Section 6.1.4.

Our primary performance metrics are flow and byte accuracy. Flow accuracy is the number of correctly classified flows to the total number of flows in a trace. Byte accuracy is the number of correctly classified bytes to the total number of bytes in the trace. In our results, we report for a given test data set the average results and the 95% confidence interval from 10 runs each with a different training set of feature vectors. Unless stated otherwise, the training data set of 8,000 samples is selected from the test data set used for evaluation. In all our experiments the test data set is a factor of 10 to 100 larger than the training data set.

6.1.1 Semi-Supervised Learning

Labelling of training feature vectors is one of the most time-consuming steps of any machine-learning classification process, especially because many Internet applications purposefully try to circumvent detection. We expect a vendor to achieve labelling of flows using a variety of orthogonal approaches, including payload analysis, port-based analysis, experimentation, expert knowledge, or a combination thereof. Clearly, it is an advantage if high classification accuracy is achieved by labelling only a small number of flows.

Recall that our approach allows clustering to use both labelled and unlabelled training flows, and then relies on only the labelled flows to map clusters to applications. This semi-supervised approach to training the classifier leverages the fact that clustering attempts to form disjoint groups, wherein each group consists of objects that bear a strong similarity to each other. Thus, the hypothesis is that if a few flows are labelled in each cluster, we have a reasonable basis for creating the cluster to application type mapping.

To test the aforementioned hypothesis, we conducted a number of experiments. The first experiment considers the possibility of the entire training data set being

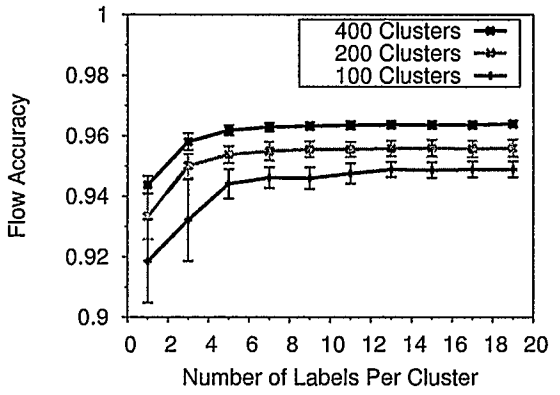
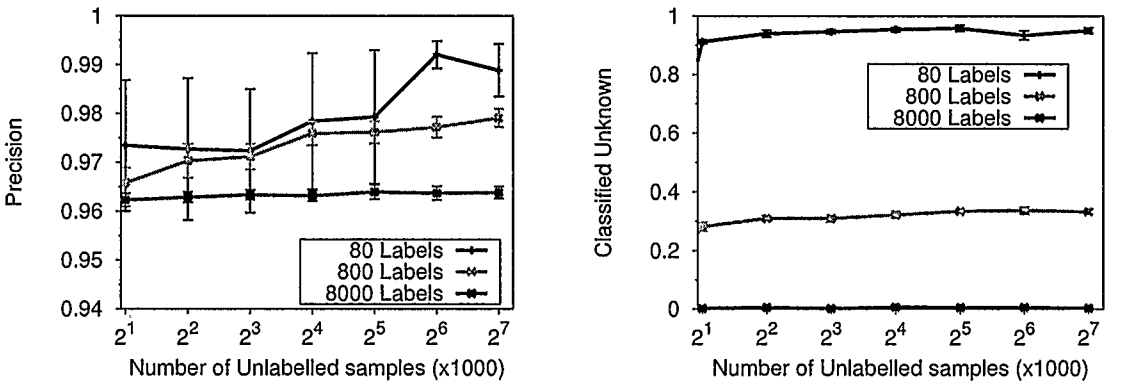


Figure 6.1: Impact of Selective Labelling of Flows after Clustering

unlabelled. In this case, we can selectively label a few flows from each cluster and use these labelled flows as the basis for mapping clusters to applications. The hypothesis here is that the clustering step produces "pure" (in the sense of application types) clusters; in Chapter 5, we provided empirical evidence of this hypothesis. Figure 6.1 presents results from this experiment. We assume that we are provided with 64,000 unlabelled flows. Once these flows are clustered we randomly label a fixed number of flows in each cluster. Interestingly, the results show that with as few as two labelled flows per cluster and $K = 400$, we can attain 94% flow accuracy. The increase in classification accuracy is marginal once five or more flows are labelled per cluster.

For the second set of experiments, results of which are shown in Figure 6.2, we utilized 80, 800, and 8,000 labelled flows, and mixed these labelled flows with varying numbers of unlabelled flows to generate the training data set. Both labelled and unlabelled flows were randomly chosen from the April 6, 9 am Campus trace. These training flows were used to learn the flow to application mapping, with $K = 400$ in the clustering step, and we tested the resulting classifier on the same Campus trace. Note that there are 966,000 flows in this trace.



(a) Precision (b) Fraction Labelled as Unknown

Figure 6.2: Impact of Training with a Mix of Labelled and Unlabelled Flows

Figure 6.2 reports the precision of the classifier. Precision is calculated as the number of correctly labelled flows to the total number of labelled flows, with those labelled "unknown" excluded from the calculation. We observe that for a fixed number of labelled training flows, increasing the number of unlabelled training flows increases our precision. This is an important empirical result because unlabelled flows are relatively inexpensive to obtain and the penalty for incorrect labelling of a flow might be high (e.g., assigning lower priority to business critical traffic). Thus, by simply using a large sample of unlabelled flows, the precision rate can be substantially increased. This experiment further demonstrates the potential of the semi-supervised learning method.

The semi-supervised classifier makes it possible to start with a few labelled flows, and over time incrementally label more training flows so as to improve the classification performance. The results in Figure 6.2 show that even when a very small fraction of flows are labelled, the precision of the classifier remains high. As additional labels are added, the precision remains high, albeit decreasing slightly, but has the accompanying effect of significantly reducing the amount classified as unknown. Further reductions in unknown classifications can be hastened by "cherry picking"

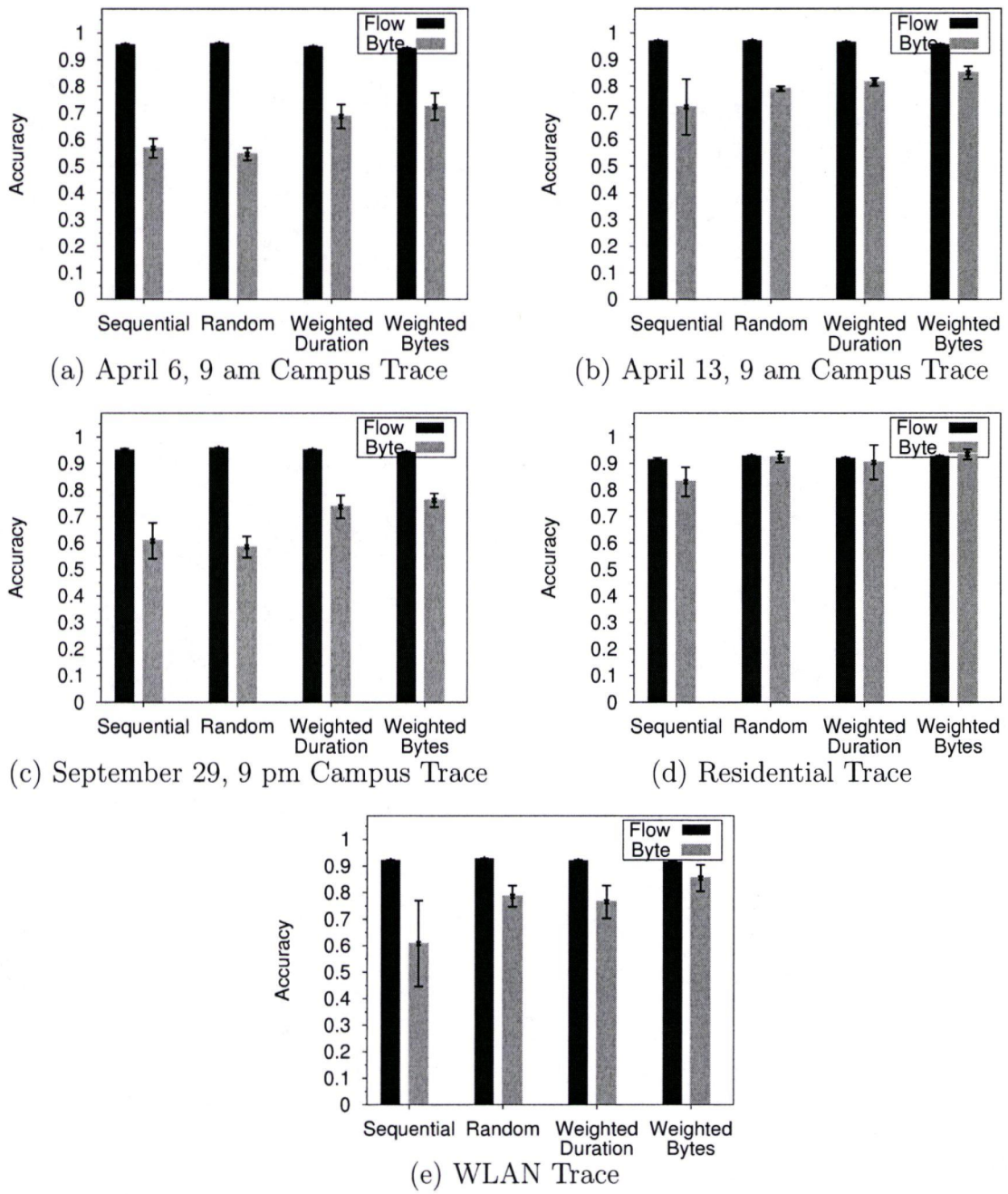
which flows to label; specifically, obtaining a few labels corresponding to highly used clusters can substantially reduce the number of unknowns.

6.1.2 The Dichotomy of Elephant and Mice Flows

The presence of elephant and mice flows in Internet traffic is well documented (see [55] and the references therein). Without proper representation of both types of flows in the training data set, we run the risk of producing a classifier that may, for example, have a high flow accuracy but a low byte accuracy. In this section, we investigate how sampling methodology influences the selection of both elephant and mice flows in the training data set.

We considered both sequential and random sampling techniques. For sequential sampling, we generated each of the ten training data sets needed for the experiments by randomly picking a point to begin sequential selection of flows. Along with simple random sampling, we also considered weighted random sampling techniques that bias selection of samples according to the transfer size of a flow or according to the duration of a flow. Our weighted sampling policy takes 50% of the flows from below and 50% of the flows from above the 95^{th} percentile of the flow transfer sizes or of the flow durations for the weighted bytes and duration policies, respectively. We believe this weighted scheme allows additional clusters to be formed to better represent elephant flows.

Figure 6.3 shows classification results from three single Campus traces (April 13, 9 am Campus trace is our largest), the Residential trace, and the WLAN trace. We observe very high flow accuracies, in excess of 95% with the Campus traces and around 90% with the Residential and WLAN traces, irrespective of the sampling technique. However, the corresponding byte accuracies are lower and they vary across the traces and with the sampling strategy. Depending on the sampling strategy, byte accuracies between 50% and 85% are attained with the Campus traces, whereas byte



(a) April 6, 9 am Campus Trace

(b) April 13, 9 am Campus Trace

(c) September 29, 9 pm Campus Trace

(d) Residential Trace

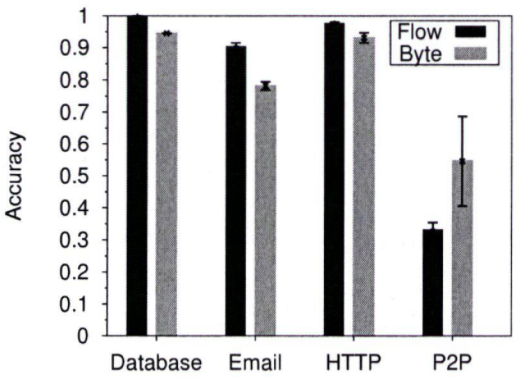
(e) WLAN Trace

Figure 6.3: Impact of Sampling Methodology on Classification Accuracy

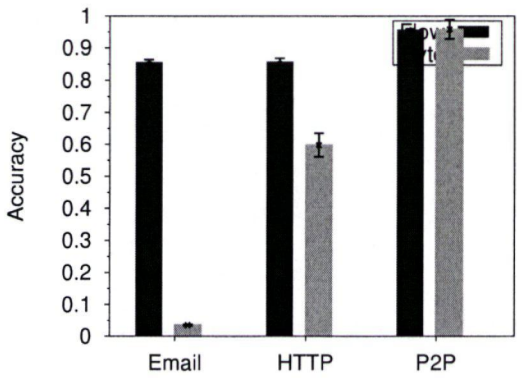
accuracies between 80% and 93% and between 60% and 85% are obtained for the Residential and WLAN traces, respectively.

Our experiments and the results in Figure 6.3 also show that sequential sampling for selecting training flows performs poorly in comparison to the random and weighted random sampling techniques. For example, in the WLAN trace, on average, byte accuracy of 61% is achieved with sequential sampling whereas byte accuracy of 86% is achieved with weighted byte sampling. The weighted byte sampling technique results in a 41% improvement of the byte accuracy compared to that with sequential sampling. Similar improvements in byte accuracies are observed in experiments with the remaining Campus traces. The byte accuracies with the Residential trace are generally higher; yet, a modest improvement of 13% can be achieved by switching from sequential to weighted byte sampling. In general, the weighted bytes sampling technique achieves the best byte accuracies when classifying traffic. We attribute this improved classification performance to the increased probability of forming more representative clusters for infrequently occurring elephant flows. Finally, it is worth noting that the large improvement in byte accuracy is possible with only a marginal reduction in flow accuracy.

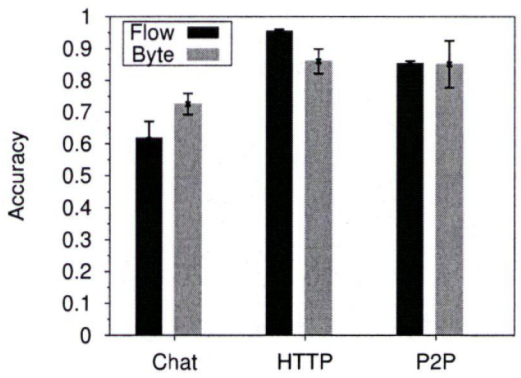
We conclude this section with a discussion of classification accuracy by application type. Figure 6.4 shows the classification accuracies for applications that contribute at least 0.5% of the flows or bytes in the traces. The results are from the weighted byte sampling experiments shown in Figures 6.3. Overall, our approach is able to classify any type of traffic, including P2P traffic, provided there are enough samples in the training data set from which the mapping between flows and applications may be learned. For the Campus trace considered (Figure 6.4(a)), we find that the classification accuracy for P2P traffic is lower than that for other traffic because P2P flows account for only a small percentage, typically less than 3% of the total flows, and therefore, our sampling techniques are unable to capture enough of the



(a) April 6, 9 am Campus Trace



(b) Residential Trace



(c) WLAN Trace

Figure 6.4: Classification Accuracy by Application

P2P dynamics to learn the flow to application mapping. It is the misclassification of P2P flows that results in the overall lower byte accuracy seen in Figure 6.3(a). As can be seen in Table 3.1, P2P accounts for a small fraction of the flows but a large fraction of the total bytes. When P2P is prominent (Figures 6.4(b) and (c)), as in the WLAN and the Residential traces, we achieve flow and byte accuracies near 90% for this type of traffic.

6.1.3 Feature Selection

Another important design choice in training our classifiers is the set of features used in the classifier. Many flow statistics (or features) can be calculated from a flow; however, not all features provide good discrimination between the classes. Using such features can decrease the accuracy of the classifier. We started with 25 candidate features. To find a subset of discriminating features we employ a feature selection method. In general, the feature selection task is exponentially hard; however, efficient methods for feature selection are widely used [34].

We use a backward greedy feature selection method [34]. The method works as follows. Given n features, we train a classifier with all features and compute its accuracy. We then find the single feature to remove such that the classifier with $n-1$ features has the highest accuracy. This process is continued until we find the maximum number of features to remove such that the resultant classifier has the best accuracy.

To choose a subset of features to use in all of our experiments, we perform the backward greedy search with the various data sets. We then find which subset of the features were chosen most often in the different experiments. The eleven flow features that were chosen are: total number of packets, average packet size, total bytes, total header (transport plus network layer) bytes, number of caller to callee packets, total caller to callee bytes, total caller to callee payload bytes, total caller to callee header bytes, number of callee to caller packets, total callee to caller payload bytes, and total callee to caller header bytes. In the rest of this Chapter we use this set of features as a basis for our classifiers[1].

Interestingly, we found that flow features that have a time component such as

[1]Caller is the host that initiates a flow (e.g., the host that sends the SYN packet during TCP connection establishment); callee is the host that reacts to the initiation request (e.g., the host that responds with a SYNACK packet during TCP connection establishment).

duration, interarrival time, and flow throughput were found not to be useful by the feature selection algorithm. In general, selection of time-oriented features should be avoided as they are less likely to be invariant across different networks.

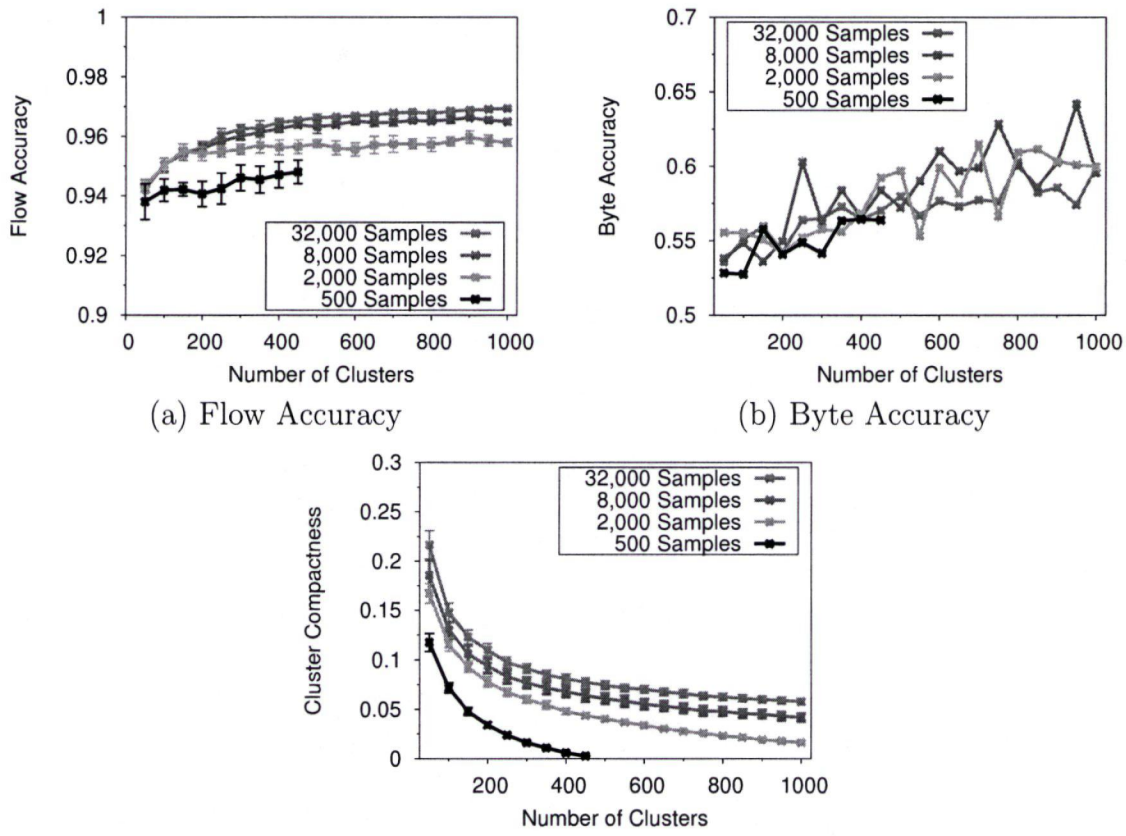
Internet flow features, in general, exhibit a high degree of skewness [60]. We found it necessary to transform the flow features to obtain higher classification accuracies. Experimentation with several commonly used transforms indicated that logarithmic transformations yield the best results. In general, transformation of features is often necessary in most machine learning applications.

6.1.4 Tuning the Classifier

The number of clusters (K) impacts the quality of clustering (and thus the quality of classification), the time complexity of building the classifier, and the runtime performance of the classifier. To determine a suitable K, we varied both the number of clusters and the number of labelled training flows. Figure 6.5 shows the results from experiments where we varied K from 50 to 1,000, and varied the number of vectors in the training data sets from 500 to 32,000 flows. The training flows were selected using a simple random sampling (See Section 6.1.3.).

Several observations can be made from the flow accuracy results in Figure 6.5(a). First, flow accuracies in excess of 95% are achieved when using training data sets with 2,000 or more labelled flows. Second, although having more flows in the training data set improves flow accuracy, the percentage improvement shows diminishing returns. Third, as K increases, we observe that the flow accuracy also increases. For example, for training data sets with 8,000 or more flows, a large K ($\geq 4,000$) can facilitate flow accuracies around 97.5%. However, having such large values for K is not practical as this increases the time complexity of the classification step.

Figure 6.5(b) shows the byte accuracy results. The byte accuracies, on average, ranged from 52% to 62%. We did not find any clear relationship between number of



(a) Flow Accuracy

(b) Byte Accuracy

(c) Cluster Compactness

Figure 6.5: Parameterizing the Classification System (April 6, 9 am Campus Trace)

flows in the training data set and the corresponding byte accuracy. Byte accuracy is very sensitive to a few large elephant flows in network traffic. In general, a simple random selection of training flows from the traces is unlikely to capture enough elephant flows in the training data sets, especially because the training data sets consist only of a few thousand flows. For example, there are 58 FTP data transfers that account for 6.5% of the bytes in the April 6, 9 am Campus trace, and these are rarely captured in the (randomly chosen) training data set. Thus, these large FTP flows are typically misclassified. Increasing the number of clusters K typically

improves byte accuracy, albeit marginally, because the likelihood of forming clusters for the elephant flows when they are selected in the training data set increases. The use of more sophisticated sampling techniques such as weighted bytes policy (discussed in Section 6.1.2) can substantially improve the byte accuracies. Another solution we found for classifying "rare" applications of interest is to specifically add flows of this type to the training data set. This makes it possible for the classifier to have clusters representing this application as well.

Figure 6.5(c) shows *cluster compactness* [36]. Cluster compactness measures the degree of homogeneity within the clusters formed; a low compactness measure indicates more homogeneity among flows in the clusters. Clearly, if each flow in the training set is assigned its own independent cluster, then cluster compactness will reach zero. We see this trend in the graph wherein the larger K becomes, the lower compactness becomes. However, we also see a plateau effect for $K \geq 400$, wherein compactness decreases slowly with increases in K.

Choosing parameter values for the clustering step presents a tradeoff between accuracy and classification overhead. Our results show that a larger training data set improves the flow accuracy, and a larger K improves flow accuracy, byte accuracy, and cluster compactness. A large value for K, however, increases the classification overhead and some caution must be emphasized when choosing K. Because our semi-supervised learning does not require all flows to be labelled, we advocate using a large training data set with as many labelled flows as possible, and a K value that achieves the desired tradeoff between accuracy and computation overhead. Essentially, the size of the training data set and the value for K are tuning parameters that can be adjusted depending upon the application.

6.2 Realtime Classification

In this section we discuss the design, implementation, and performance of a prototype realtime classification system we developed using our classification framework.

6.2.1 Design Considerations

A fundamental challenge in the design of the realtime classification system is to classify a flow as soon as possible. Unlike offline classification where all discriminating flow statistics are available a priori, in the realtime context we only have partial information on the flow statistics.

We address this challenge by designing a layered classification system. Our layers are based upon the idea of *packet milestones.* A packet milestone is reached when the count of the total number of packets a flow has sent or received reaches a specific value. We include the SYN/SYNACK packets in the count. Each layer is an independent model that classifies ongoing flows into one of the many class types using the flow statistics available at the chosen milestone. Each milestone's classification model is trained using flows that have reached each specific packet milestone.

To classify flows in realtime we track the flow statistics of each ongoing flow. When a flow reaches the first packet milestone, it is classified using the first layer's classification model. When the flow reaches further packet milestones it is then reclassified using the appropriate layer's model. When a flow is reclassified, any previously assigned labels are disregarded.

This layered approach allows us to revise and potentially improve the classification of flows. The memory overhead of our approach is linear with respect to the number of flows because we use the same feature set at all layers.

An alternative approach would be to classify at points that are significant in the transport-layer protocol. For example, the first layer could classify with just the

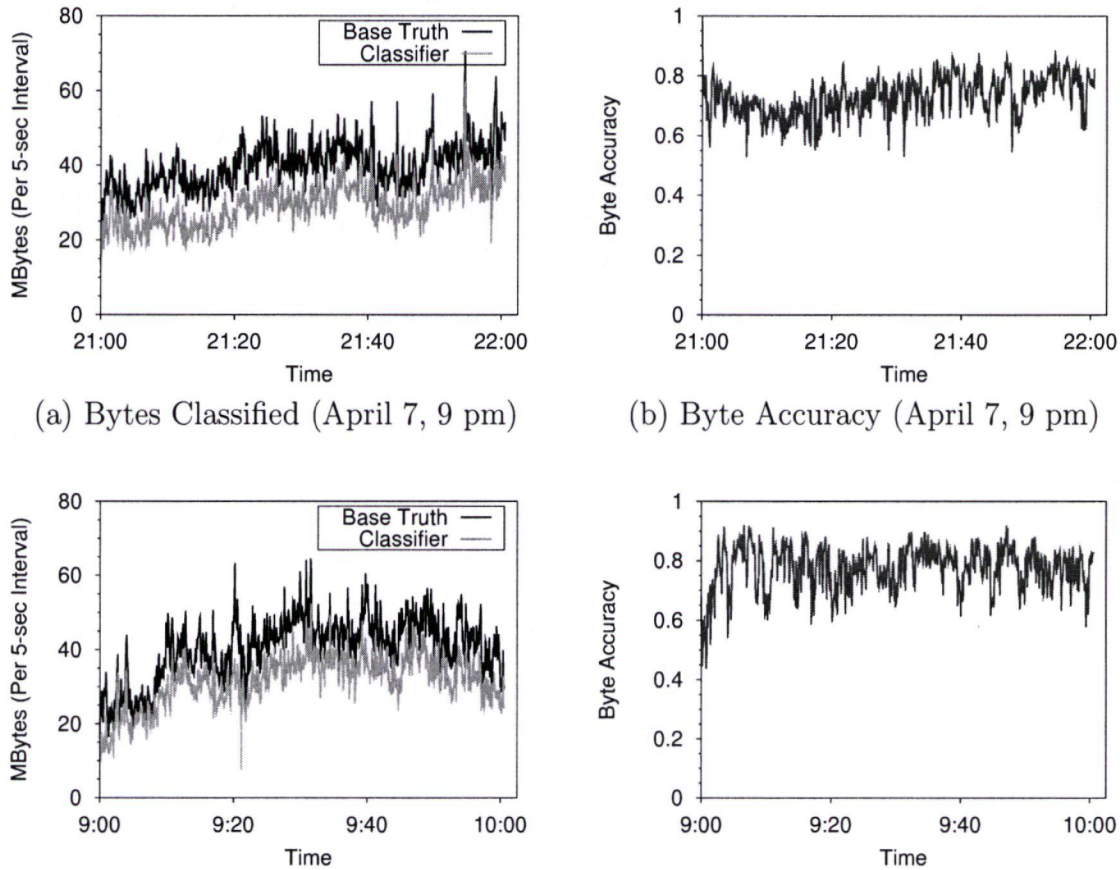
transport protocol and port number when the very first packet is seen. For TCP connections, the next layer could be when the first data packet is seen (i.e., following the connection establishment phase). We defer this approach for future work.

The prototype was built using an existing IDS system called Bro [59]. Bro is an ideal candidate for our prototyping effort because by design it performs the realtime analysis of network traffic. We added two scripts to Bro 0.9a (unmodified) to enable our realtime classifier. The first script tracks the flow feature set. When a flow reaches a specific packet milestone, the script calls a classification function in our second Bro script. The second Bro script contains a classification function for each specific milestone at which we reclassify our flows. This second Bro script was generated by a C++ program that reads in the training flows and generates the mapping from flows to applications. We use the same features as in Section 6.1 with one obvious exception; we do not use *total number of packets.*

6.2.2 Classification Results

For these experiments, we trained the classifier using flows from the April 6, 9 am trace with 966,000 flows. For each of N layers we created models using 8,000 training flows, using $K = 400$. In our implementation, we use thirteen layers and separate our packet milestones exponentially $(8, 16, 32, \cdots)$. For layers eleven and higher (packet milestones greater than 4,096), fewer than 5% of flows in the trace reached these milestones. Therefore, for these layers we trained with all available flows in the trace (always more than 500). We do not test our model on the same trace from which we generated the training data to avoid biasing our results.

We calculated realtime byte accuracy as follows. When a packet arrives for a given flow we use the current label assigned by our classifier to determine if the bytes for this packet have been correctly classified. Byte accuracy in a given time interval is simply the fraction of bytes that were assigned the correct labels. Note that the



(a) Bytes Classified (April 7, 9 pm)

(b) Byte Accuracy (April 7, 9 pm)

(c) Bytes Classified (April 13, 9 am)

(d) Byte Accuracy (April 13, 9 am)

Figure 6.6: Performance of Realtime Classifier

system may reclassify a flow several times and could therefore assign multiple labels to the flow during its lifetime. Thus, we report only byte accuracy in a moving time window and do not report flow accuracy.

Figure 6.6 presents example results by using the April 7, 9 pm and April 13, 9 am campus traces (April 13, 9 am is our largest 1-hour campus trace). We see that the classifier performs well with byte accuracies typically in the 70% to 90% range. Quantitatively similar results were obtained when tested on the other traces.

Another aspect we considered was the effect of adding additional layers to our

Table 6.1: Real-Time Byte Accuracy with Number of Layers Varied

Layer	Packet Milestone	Byte Accuracy
1	8	40.0 %
2	16	45.8 %
3	32	48.9 %
5	128	49.5 %
10	4096	49.7 %
13	16384	77.5 %

classification system. For the April 13, 9 am trace shown in Table 6.1, 78% of the flows had correct labels after classification at the first layer (8 packets). If this were the only layer used in our system, this would result in 40% of the bytes being correctly classified. This low value occurs because many of the elephant flows are incorrectly classified at the early stages. Using five layers improves the byte accuracy to 50%. Finally, with thirteen layers, byte accuracy reaches 78% as we are correctly classifying the elephant flows. We also note that the last label given to a flow is correct 82% of the time.

Some of the intermediate layers appear to provide little or no improvement in byte accuracy. These additional layers can be removed and still allow our classification system to achieve similar byte accuracies while reducing overhead.

6.3 Discussion

In this section we discuss three topics: the *arms race* occurring between network operators and users/application developers, the longevity of our classifier, and the ability of our methodology to determine when retraining is required.

6.3.1 The Classification Arms Race

To fully comprehend the traffic classification problem, one needs to understand its history. For many years, traffic classification was trivial, as applications tended to

abide by well-known port numbers. Application developers had little motivation to deviate from this. Over time though, things changed; network bandwidths increased, new applications emerged, and the Internet became available to a much larger audience. In the late 1990's, the exchange of high fidelity music (and later video) became feasible and accessible to a large audience. The increased bandwidth consumption contributed to the creation of the traffic classification problem.

What ensued can best be described as an arms race involving at least four parties - content owners, ISPs, users, and application developers. The race started slowly. Initially ISPs could identify these file sharing applications using well known ports. The ISPs could then control or block the offending applications. In September 2002 KaZaA escalated the race by introducing dynamic ports, effectively bypassing blocked ports. Since that time, the two sides have gone back and forth numerous times.

One important observation is that file sharing users have little loyalty to the applications. If an application is blocked or impeded by an ISP, users will quickly migrate to an application that can provide them with access to the content they want. It is, therefore, important for a traffic classifier to overcome current countermeasures, and also be able to function with the countermeasures that may come in the future. For example, encryption is currently not widely used by file sharing applications, even though some of these applications already support it. If required, users could easily start encrypting their traffic. This would immediately prevent content-based classifiers from properly identifying file-sharing traffic.

We believe our classifier based on flow statistics will be difficult to circumvent. This is because it is very hard for applications to disguise their behaviour without adding large amounts of overhead. Consider the *average packet size* feature. To disguise this feature, an application would need to modify flows so the *average packet size* across all its flows appear random. This would involve adding significant

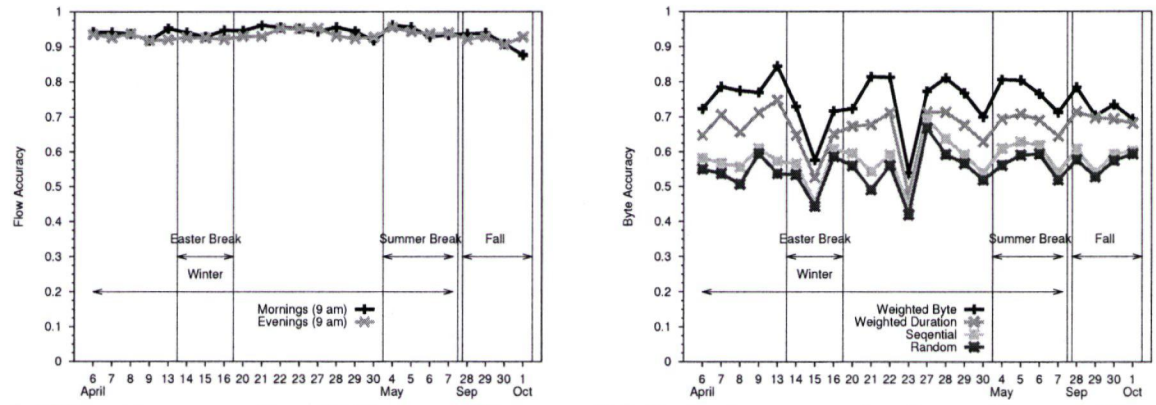
overhead because sometimes either padding would need to be added to packets to increase the packet size or full packets broken up into several smaller packets when sent to decrease packet size. Similarly, changing the ratio of data sent between hosts could also require substantial amounts of extra data transfer. Ultimately, to defeat the classifier the overhead required would be crippling. Nevertheless, if new applications originate or old applications change behaviour, we would like the classification system to adapt accordingly.

In light of the above discussion, we can identify (at least, to first order) two important considerations. One, a classification system should be robust and be able to maintain high classification accuracy in the presence of transient changes in network/application usage patterns; our hope would be that classifiers have a reasonably long shelf life. Two, when there are substantial changes, for example, owing to introduction of new applications, or owing to behavioural changes of existing applications, the classifier should automatically detect the need for retraining; our intent in this case is to keep up with the arms race. These two issues are further discussed in Sections 6.3.2 and 6.3.3, respectively.

6.3.2 Longevity

To experimentally evaluate the long-term predictive value of classifiers, we tested the classifiers that were built by sampling from the April 6, 9 am Campus trace (see Section 6.1.2) across the forty-eight Campus traces. Figure 6.7 presents sample results from our experiments.

Figure 6.7 (a) shows the classification accuracy as a function of time. The results shown are for classifiers trained using labelled flows sampled by the weighted bytes technique. Qualitatively similar results were obtained for other sampling techniques (we do not show them on the graph to avoid line "crowding"). Our results show that the classifier retained a high flow accuracy throughout the 6-month period. Flow



(a)Flow Accuracy for all Campus Traces (b) Byte Accuracy for 9 am Campus Traces

Figure 6.7: Longevity of Classifier

accuracies close to 95% are consistently achieved in the traces we tested, including major transitions such as end of winter semester, summer break, and beginning of fall semester. For example, the student population substantially dwindles during the summer months. Also, during the summer months, the number of Database flows (MSSQL) substantially increased from the 5% originally seen in the training data sets to over 25% during during this period. However, our classifier is still able to classify the new database traffic correctly. There is no substantial loss in classification accuracy.

In Figure 6.7 (b), we present the byte classification accuracies for the 9 am Campus traces. The results for the 9 pm Campus traces are qualitatively similar. The byte accuracy trend is similar to the flow accuracy trend but shows more variability. We also find that the weighted bytes approach for selecting training flows consistently achieves higher accuracies than the random and sequential selection techniques because more P2P traffic is successfully classified by the former. We further investigated why the byte accuracy drops significantly on April 15 and April 23. The drop in byte accuracy was due to misclassification of FTP flows as either P2P or HTTP.

In general, FTP is not captured well by any of the sampling techniques because it accounts for only a small fraction ($< 0.01\%$) of the flows, and thus, is unlikely to be captured in a small-sized training data set. Typically, FTP accounts for less than 5% of the bytes but on those days it accounted for 21.6% and 26.6% of the bytes, respectively.

6.3.3 Retraining

The results above show that our classifiers remained fairly robust over time and for different traces. While encouraging, a mechanism for updating the classifiers is still required. An update of the classifier can be in the form of re-clustering, re-labelling of clusters, or both. The ideal way to determine if an update is required is to track and measure the classification accuracy as new flows are classified. However, measuring the accuracy is not possible, as the flow labels are not known. There are, however, two indirect measures for measuring reliability of the classifiers. The first is to build classifiers using a mix of labelled and unlabelled flows, as discussed in Section 6.1.1. Then, we can track the number of flows that are not assigned any label. If this number increases, it indicates the need for labelling some of those unknown flows so that their corresponding clusters are also labelled. The semi-supervised approach makes it possible over time that this type of an update would capture under-represented flow types and allow the accuracy of the classifier to improve.

Alternatively, a statistical measure could be used to detect changes in the quality of the clustering model. We propose using the average distance of new flows to their nearest cluster mean; a significant increase in the average distance indicates the need for an update. Formally, this measure corresponds to the likelihood function of the clustering model in representing the new flows. The measure is easy to compute and track, as it does not require knowledge of the flow labels. While an indirect measure of accuracy, the *clustering likelihood* measure is correlated to the accuracy of the

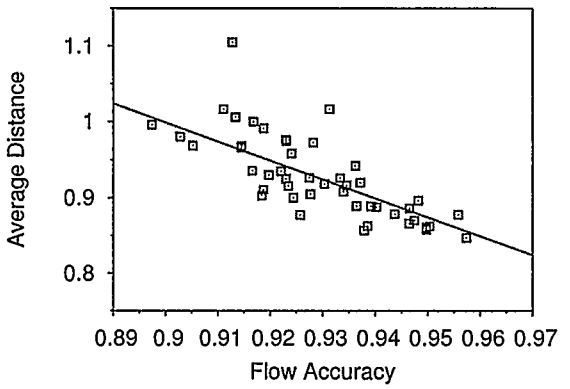


Figure 6.8: Correlation between Average Distance and Flow Accuracy

classifier. Recall from Section 4.3 that new flows are mapped to clusters using the distance metric as a measure of similarity. Thus, it is expected that average distance between flows and cluster centres is negatively correlated with accuracy.

Figure 6.8 shows a scatter plot of flow accuracy and average distance for all forty-eight Campus traces for one of the classifiers used in Figure 6.7. These sample results show that when the average distance to the cluster centres is higher, the flow accuracies are typically lower, and vice versa. We repeated the above test for the 9 remaining weighted bytes classifiers we built by sampling from the April 6, 9 am Campus trace and found similar results. The correlation between average distance and accuracy ranged from -0.57 to -0.75 in the models we tested; the average correlation was -0.69.

In practice, the clustering likelihood can be easily used as an indicator of when our classification models need to be retrained. As previously demonstrated, the classification model is fairly robust and would not need to be retrained frequently. The average distance could be recorded for large intervals such as on an hourly or a daily basis. The average distance obtained during the interval just after retraining

could be used as a baseline as this most likely is when the model is most accurate. If the hourly or daily average distance increases, and stays generally above a certain threshold (e.g., 50% above the baseline), then this may be treated as an indicator for retraining. The detection threshold can be adjusted to accommodate different amounts of variation in flow accuracy.

Once the need for retraining is detected there are various approaches to retraining that can be employed to update the classification model besides the simple and extreme one of retraining the models completely from "scratch" using new labelled and unlabelled flows. While we do not evaluate these approaches, we note some approaches to retraining that do not require completely rebuilding the model. One approach is to create new clusters using new flows that were far from their means. This would be followed by selectively querying the labels of flows from these uncertain clusters. In the machine learning literature, this is known as *active learning* [71]. Another approach is to sample new flows and randomly replace only a fraction of the existing flows in the training data set and then rebuild the classifier.

6.4 Summary

In this chapter, we evaluated the proposed semi-supervised classification framework using offline and realtime prototypes. We found that both high flow and byte accuracy can be achieved in both cases and we can successfully classify a variety of applications such as P2P, HTTP, FTP, and email. The classifiers are robust to transient changes in the network. The detection of non-transient changes such as introduction of new applications or behavioural changes to existing applications can be facilitated using the proposed detection of retraining points.

In the next chapter, we address the problem of applying our framework at the network core where only unidirectional traces are available.

Chapter 7

Classification at the Network Core

This chapter considers the problem of traffic classification at the network core. Specifically, the offline classification framework developed in Chapter 6 is extended and applied for classifying network traffic as may be observed at egress/ingress points of the network core. At egress/ingress points at the network core, observing both directions of a flow may not be possible because of routing asymmetries. This poses two challenges. First, important statistics for the satisfactory classification of a flow may not be available. Second, classification can only use per-flow information and cannot rely on additional information such as communication patterns between hosts.

In light of the above, we study in Section 7.1 the influence of "directionality" of flow statistics in classifying traffic. Our results show that flow statistics for the server-to-client direction of TCP flows achieve better classification accuracies. Server-to-client statistics of a flow may not always be available at the network core, thus, we develop a flow statistics estimation algorithm in Section 7.2.

7.1 Classification Results using Unidirectional Flows

The empirical traces at our disposal have both directions of a flow. Our goal is to study how the directionality (i.e., client-to-server or server-to-client) of a flow impacts classification results. We generated from each empirical trace a "server-to-client" data set and a "client-to-server" data set that for each flow in the trace records only the packets seen in the server-to-client direction or the client-to-server direction, respectively. To represent the typical case of traffic seen at the network core, we selected for each flow in an empirical trace either the client-to-server direction packets

or the server-to-client direction packets. We refer to this third category of data sets as "random directionality" in this chapter. We restrict our attention to the first week of the campus traces for this study.

7.1.1 Configuring the Classifier

We have found in our experimentation that increasing the sample size of our training data set between 2000 and 128,000 does improve our classification results, albeit, with diminishing returns when increasing the training data set sizes. This finding is expected and corresponds well with the results in Section 6.1.4. From each data set, we generated *training data sets*, each of which were generated by selecting 64,000 random flows using random sampling. As we advocated using a large training data set in Section 6.1.4, a sample size of 64,000 was chosen because we wanted to maximize the ability of the models to represent different applications and their dynamics within the limits of computational requirements of building the model. Furthermore, for the purpose of evaluation we assumed that all training flows are labelled. Much smaller training data sets or data sets with labelled and unlabelled flows could also be used for these experiments. However, as similar types of cases were already evaluated in the previous chapter, we do not explore them again and strictly focus on the impact that the directionality of the unidirectional flows has at the network core.

The selection of features plays an important role in machine learning. Although many statistics can be obtained from a flow as discussed in Section 6.1.3, in the case of unidirectional flows the number of available features is reduced. We experimented with feature selection algorithms and settled on the following features: total number of packets, mean packet size, mean payload size excluding headers, number of bytes transferred, flow duration, and mean inter-arrival time of packets. Due to the aforementioned skewed distribution of many of these features, we found that using logarithmic transformations yields the best results. In these experiments we have

restricted our focus to only unidirectional features of a single direction of a flow.

Recall that the K-Means algorithm takes the number of clusters K as input. In general, and as discussed in Section 6.1.4, K can be considered a tuning knob that can be adjusted based on the needs of the classifier. The evaluations in this chapter used K equal to 400 as this represented the best tradeoff between accuracy and computational overhead.

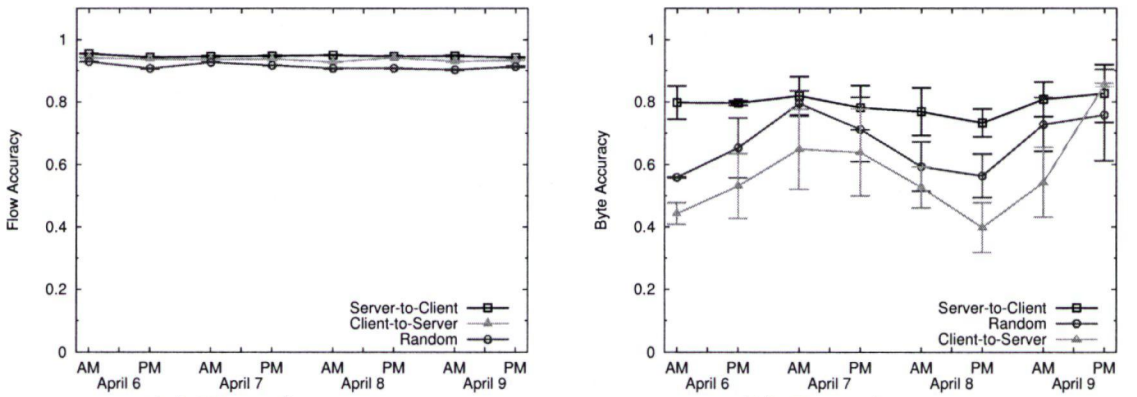
7.1.2 Experimental Methodology

From each data set, we generate 10 different training data sets. After the clustering was complete, we used each of these models in our classifier for classification of the entire respective trace. We report the average results and the 95% confidence intervals for the 10 models. The number of flows in a *test data set* typically ranges between 500,000 to 1,000,000.

7.1.3 Results

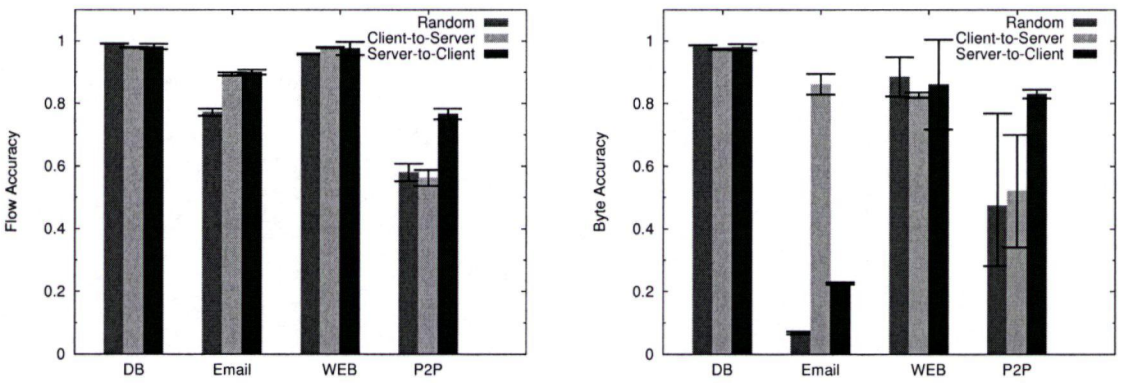
Figure 7.1 shows the classification accuracy results for data sets derived from each of the selected traces. Overall, we found that the server-to-client data sets consistently give the best classification accuracy achieving, on average, 95% and 79% in terms of flows and bytes, respectively. With the random data sets, the average flow and byte accuracy was 91% and 67%, respectively. The client-to-server data sets were able to correctly classify, on average, 94% of the flows with an average byte accuracy of 57%. In general, use of the client-to-server data sets resulted in the worst byte accuracies in all traces, except for the April 9, 9 pm trace.

Figure 7.2 shows the flow and byte accuracies achieved for the four most significant applications (in terms of number of flows). We found that all three types of data sets have a high flow accuracy for Database, Email, and Web traffic, with both client-to-server and server-to-client data sets achieving, on average, accuracies



(a) Flow Accuracy (b) Byte Accuracy

Figure 7.1: Classification Accuracy (Campus Traces)



(a) Flow Accuracy (b) Byte Accuracy

Figure 7.2: Classification Accuracy by Application (April 6, 9 am Campus Trace)

Table 7.1: Confusion Matrix with Server-to-Client Data Sets (April 6, 9 am Campus Trace)

	Classification				
Actual Class	Web	Email	P2P	DB	OTHER
Web	511375	5214	7284	520	1084
Email	6620	64732	3066	88	631
P2P	6886	3620	47716	199	254
DB	1262	420	872	41262	166
OTHER	1904	232	1018	103	5336

in excess of 90%. The application type that proved the most difficult to classify was P2P. The server-to-client data sets achieved a 77% flow accuracy; this is 20% greater than the accuracies with client-to-server and random data sets.

Table 7.1 shows the confusion matrix [73] with the classifier using a server-to-client data set to help further illustrate the accuracy of the classification by application. In this $m \times m$ matrix the data point $c_{i,j}$ indicates the number of flows from class i that were classified as class j. Obviously, we want values along the diagonal to be much larger than the others which is what we do find. By looking across the row of the confusion matrix at a given class i we can calculate the recall for that class. Likewise, by looking down a column at a given class j we can calculate the precision of that class.

The per-application byte accuracy for Database and Web is high with all three types of data sets. However, for Email and P2P flows the accuracies vary considerably between the different data sets. For Email flows, the client-to-server data sets provide 86% accuracy, but the random and server-to-client data sets have extremely low accuracies of 7% and 23%, respectively.

While it is difficult for us to make a generalization to encompass every model and trace, in the models where we did extensive analysis of the results, the reason why the client-to-server data sets classified Email so well was that SMTP flows were

being correctly classified. In the client-to-server models, SMTP flows were put into a few large clusters that classified most of the SMTP traffic, with one of these clusters normally capturing most of the large (in terms of bytes) SMTP flows. However, in the server-to-client models the SMTP clusters were more fragmented and generally formed many small clusters. The smaller clusters were generally for SMTP flows with few bytes transferred (less than 2000 bytes). The larger SMTP flows that accounted for most of the Email bytes generally did not form a cluster and were included in clusters labelled either as P2P or Web. The confusion matrix in Table 7.1 further confirms that these misclassifications with the server-to-client models are P2P and Web. In the random models, SMTP did not form many clusters, which resulted in SMTP being misclassified most of the time.

The server-to-client data sets are better able to classify the P2P flows than the other data sets. With the server-to-client data sets, byte accuracy of approximately 83% is achieved, which represents a 30% increase over client-to-server and random data sets. This higher classification accuracy is because 20% more P2P flows are correctly classified using the server-to-client data sets. This marked difference from the other data sets is one of the main reasons why server-to-client data sets achieve the best flow and byte accuracies in Figure 7.1.

If we were to employ this type of traffic classification system at the network core and, for example, tried to give a lower priority to P2P traffic or higher priority to Web traffic, we think we would be quite successful. Overall, in all our models Web has precision and recall values of 97%. P2P flows have a precision of 82% and a recall of 77%. In the case of lowering the priority of P2P traffic, these results indicate that 77% of the P2P would be correctly given a lower priority and at the same time less than 3% of the Web flows would be mistakenly given a lower priority. Such a system if deployed in real-time could greatly reduce the strain that P2P puts on many networks.

While we have advocated the discrimination of P2P and Web traffic in the above example we are, however, not limited to just these two types of applications. If reducing P2P was not the concern and instead prioritizing mission-critical business traffic was the focus then our classification system could be used just as successfully. Business-critical traffic from a Database achieves a high accuracy as well. The confusion matrix provides further evidence of this fact with a precision of almost 98% and a recall of 94% when classifying Database flows.

We investigated why the Database flows achieved high accuracies with all the statistics. The confusion matrix provides further evidence of the Database flows being accurately classified and the classifier having a precision of almost 98% when classifying Database flows. We found that in the client-to-server direction the Database flows generally sent 5 packets with 76 bytes of total payload data, and in the server-to-client direction the Database flows sent 4 packets with 63 total bytes of payload data. This very regular pattern exhibited by the Database flows allowed for highly accurate clusters to be formed in our models even though these clusters are at spatially different places in the client-to-sever and the server-to-client models. In models with the random flows, we found clusters were forming at both places which accounts for its high accuracy as well.

7.2 Classification Results using Flow Estimation

In this section we introduce and use our flow statistic estimation algorithm. This algorithm uses the packets of an unidirectional flow to estimate the flow statistics of the other direction of the flow it does not see in the trace. The estimation algorithm is based on the syntax and semantics of the TCP protocol and thus, would not work for other transport protocols such as UDP. The algorithm is introduced in Section 7.2.1. Section 7.2.2 discusses some of the assumptions made by the algorithm. Section

7.2.3 outlines exceptions that may influence the accuracy of the algorithm. In Section 7.2.4, we empirically verify the estimation algorithm's predictions. Finally, in Section 7.2.5, we test the classification accuracy using the estimated statistics.

7.2.1 Algorithm

The statistics of interest to us can be divided into three general categories: duration, number of bytes, and number of packets. After we obtain the data for these three general categories we can calculate other statistics such as average throughput, mean packet interarrival time, and packet average size.

The duration of a flow is the amount of time from when the first packet of a flow is sent until the last packet of the flow is sent. This statistic is fairly easy to calculate; we can use the first and last packet sent in the direction we observe of the flow as a good estimate of the duration. This works because typically in a well-behaving TCP connection every packet that is sent receives a corresponding acknowledgment from the other host. The packet exchanges typically occurring at the beginning and at the end of a flow have the SYN and FIN packets, respectively. In cases where we did not see the SYN and/or FIN exchange, such as when the traffic monitor drops packets, we calculate the duration with the first or last exchange of data packet and acknowledgment packets which may result in a less accurate estimate of the flow duration.

The second category of statistics is concerned with the number of bytes transmitted. Our approach for calculating the number of missing bytes is similar to the technique developed by Smith *et al.* [69]. In the TCP protocol, the host responds to reception of TCP segments (packets) by sending acknowledgments (ACKs) with the sequence number field (SEQ) in the TCP header set to indicate the next expected in-order byte. By using these ACK numbers we can estimate the amount of data that has been received by calculating the offset between the highest ACK number and the

lowest ACK number seen. This works fairly well for all TCP connections. We did, however, find one exception that caused our calculations to go astray. This occurred for connections that were closed using TCP resets (RST). For TCP RST packets, the ACK number may not correspond to the in-order byte sequence received. Instead, some TCP implementations let the field be a randomly assigned value. To combat this problem we exclude the ACK numbers from RST packets when we calculate the highest and lowest ACK numbers.

The last category of statistics, the number of packets sent, is the most difficult to estimate. We derive a set of heuristics that estimate, for each TCP flow, the number of packets that could potentially be received in the other direction between transmission of two successive packets. We assume that if a SYN packet is seen, then we are seeing the client-to-server packets of a flow. Otherwise, we assume we are seeing the server-to-client packets. Algorithm 2 shows the rules that we defined. We track the last sequence (*PrevSeq*) and acknowledgment numbers (*PrevAck*) seen in the flow; before a flow starts these values are set to zero. We also calculate the change in the sequence (*SeqChg*) and acknowledgment (*AckChg*) number between the packets that we see. In the event that we do not receive a SYN or a SYNACK packet at the beginning (or at all), our algorithm processes the first data packet with either our first (line 5) or second rule (line 7), and then works correctly afterward.

We explain the remainder of this algorithm using examples. Let us assume that we are seeing the client-to-server packets and the last packet (for the flow of interest) had a sequence number of 100 and an acknowledgment number of 200, and the next packet has a sequence number of 1560 and an acknowledgment number of 200. The increase in sequence number indicates that the most recent packet carried some payload data. However, since the acknowledgment number has not increased we infer that the missing server-to-client packets for this interval had no payload data and would most likely be ACKs corresponding to the payload in the last packets

Algorithm 2 Packet Estimation Algorithm

Input: Set of Unidirectional Flows
Output: Set of Estimated Flow Statistics

```
1  foreach TCP flow f do
2      PrevSeq = 0, PrevAck = 0, MissedAcks = 0, MissedData = 0
3      foreach Packet p do
4          Calculate (SeqChg, AckChg)
5          if SeqChg > 0 and AckChg = 0 and PrevSeq = 0 then
6              continue;                    ▷ SYN packet sent and nothing is missed
7          else if SeqChg > 0 and AckChg > 0 and PrevAck = 0 then
8              MissedAcks = MissedAcks + 1;          ▷ SYNACK or SYN missed
9          else if SeqChg > 0 and AckChg = 0 then
10             MissedAcks = MissedAcks + ⌈SeqChange/MSS⌉;
11         else if SeqChg = 0 and AckChg > 0 then
12             MissedData = MissedData + ⌈AckChange/MSS⌉;
13         else if SeqChg > 0 and AckChg > 0 then
14             MissedData = MissedData + ⌈AckChange/MSS⌉;
15         else if SeqChg ≤ 0 or AckChg ≤ 0 then
16             continue;          ▷ Nothing has been missed from last packet seen
17         end
18     end
19 end
```

sent. This case would be caught by our third rule (line 9) where we check to see if the sequence number has increased and the acknowledgment number has not. We calculate the number of ACKs missed as the sequence number change divided by the expected maximum segment size (MSS). Conversely, if the sequence number does not increase but the acknowledgment number does increase we infer that in this interval packets that were sent in the other direction contained a total payload size directly proportional to the change in the acknowledgment numbers. To calculate the number of data packets that should have been received we divide the acknowledgment number change by the MSS. This case is handled by our fourth rule (line 11). The fifth rule (line 13) handles cases where data is being sent simultaneously in both directions and the sixth rule (line 15) handles retransmissions and packets that are received out of order.

7.2.2 Assumptions

In our rules we make three general assumptions, the first pertaining to the expected MSS of packets, the second pertaining to the ACK-ing policy of the TCP stacks, and the last in regards to retransmissions and packet loss.

We use MSS in our calculations for the number of packets sent. The MSS can be estimated from the options field in the SYN/SYNACK packets of a connection. A MSS announcement is made by each host at the beginning of a TCP connection with the lowest value typically being used. In a unidirectional trace it would be possible on a per-flow basis to estimate MSS based on any announcements seen. However, to be more computationally efficient to determine the expected MSS, we analyzed the empirical distribution of MSS in our traces. Our analysis showed that 95% of the connections had a MSS of 1460 bytes. Approximately 5% had a MSS of 1380 and some other minor groupings at 512 and 1260. Therefore, we used 1460 bytes as the expected MSS in our verification and results.

How TCP acknowledges segments depends on the TCP stacks of both the client and the server. In some cases, an ACK is sent for every packet, while in other cases an ACK is sent for every other packet. Our heuristics assume a simple acknowledgment strategy of an ACK (with 40 bytes of header data and no payload) for every data packet in the flow. We realize that this may overestimate the number of ACKs.

We also assume there are no packet losses, and therefore, our statistics do not take into account any retransmissions. We make this assumption because retransmissions have more to do with the transmission media and congestion than application specific behaviour of the flows we want to classify. However, this does make our estimations lower than what the actual numbers should be but this has the positive effect of balancing the overestimation of the number of ACKs.

7.2.3 Sources of Errors

The largest source of error we found was not from faults in our algorithm, but were a byproduct of problems encountered in the flow collection and separation. The most significant errors we found in our results occurred when SYN packets (or other random packets) of different flows were wrongly put into the same flow. This affected our estimator because having a SEQ number or ACK number range much larger than it should be causes the estimated number of bytes to be very wrong (e.g., 1 GB instead of a few couple hundred bytes). We caught these errors by having a sanity check that determines if the average packet size was larger than 1500 bytes.

Another error we found was when there were large jumps in the ACK/SEQ numbers. The main cause of this anomaly was flow merging (i.e., the end of one flow merged into the start of the next flow). To handle these, when we saw a jump in the ACK/SEQ numbers greater than 100 packets worth of data we assumed flow merging had occurred. We closed the original flow and started a new one starting with the new sequence number. When we split the flow, however, if the new flow

did not receive any more packets we assumed these jumps must have been errors and remove these single packet flows. These jumps in ACK/SEQ numbers typically occurred in approximately 750 out of every 1 million flows. The value of 100 packets was chosen so that it would be larger than the largest typical bursty loss of packets. Originally, we had set this value at 10 but found a few large flows were being split because they had consecutive packet losses of 10 to 15 packets. Thus, we increased the number to 100 and found that this was large enough to handle *almost* all the bursty packet losses and flows were not inadvertently split.

7.2.4 Validation

Estimating flow duration is easy, and overall the error in the duration estimation is low. The average flow duration was 27.5 seconds, with an error of 7.3%. In our estimation results shown in Figure 7.3, we found that normally 90% of the flows had duration errors less than 1 msec. In most cases where there was a high error in the duration, we found that the error was caused by a RST or FIN packet being sent well after the rest of the flow's packets were sent.

Figure 7.4 shows a scatter plot of the actual number of bytes versus the estimated number of bytes for the random data set generated from the April 6, 9 am trace. The scatter plot shows strong agreement between the actual and estimated amount of bytes. For our traces, the algorithm was always within 0.4% and 1.4% of the actual number of bytes.

Figure 7.5 shows a scatter plot of the actual number of packets versus the estimated number of packets for the random data set generated from the April 6, 9 am trace. As seen in the scatter plot, the estimated number of packets closely follows the actual number of packets; the estimate accuracy appears to be somewhat lower when there are more packets transferred along the missing direction of the flow. Experiments with the remaining traces showed that the packet estimate was, on average,

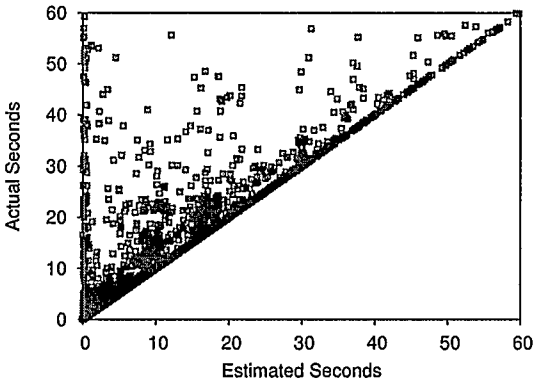


Figure 7.3: Estimation Results for Flow Duration (April 6, 9 am Campus Trace)

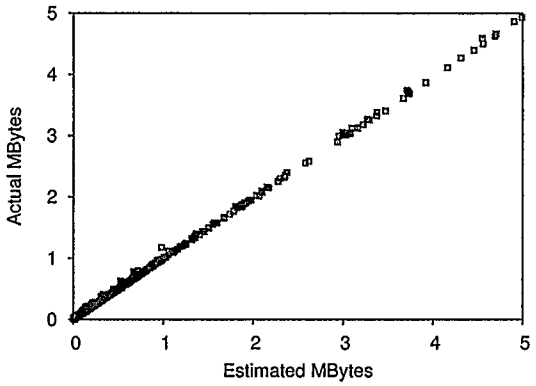


Figure 7.4: Estimation Results of the Number of Bytes (April 6, 9 am Campus Trace)

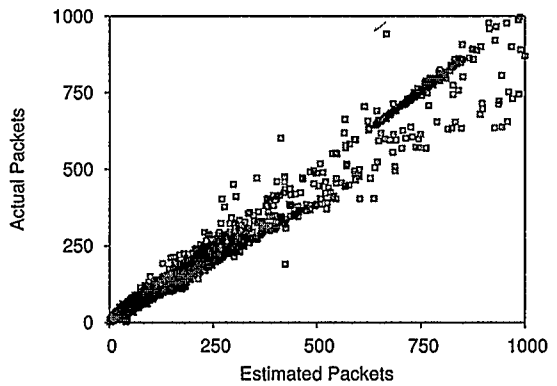


Figure 7.5: Estimation Results of the Flow Packets (April 6, 9 am Campus Trace)

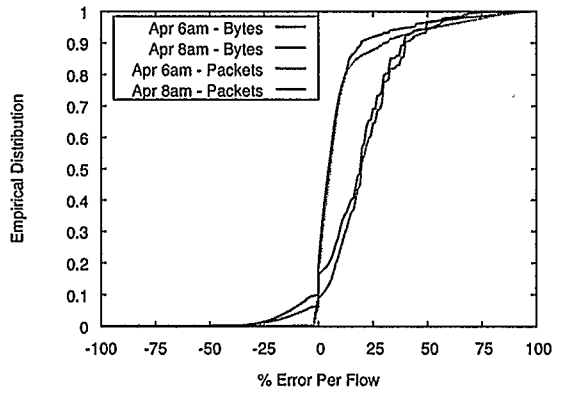
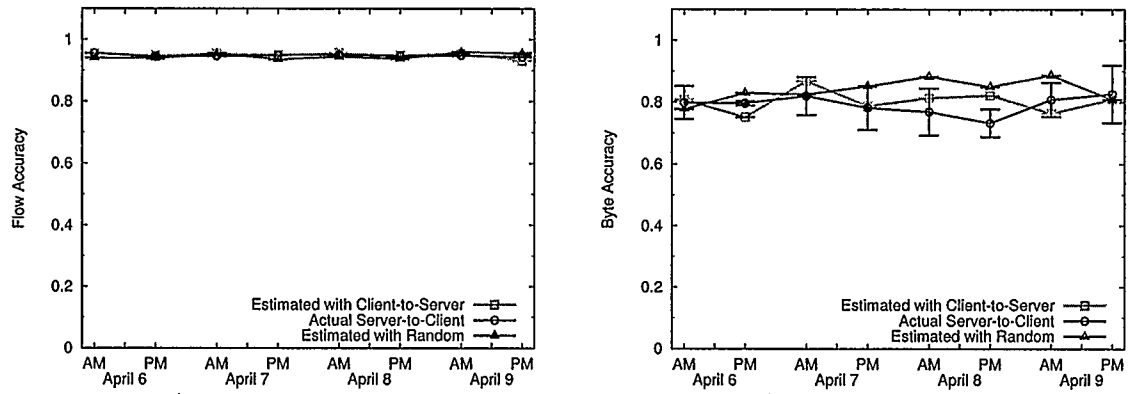


Figure 7.6: CDF of the Per-Flow Percentage Error.

within -5.3% and 1.6% of the actual number of packets.

On a per flow basis, Figure 7.6 shows the distribution of the per-flow percentage error for both packet and byte estimation. It shows that our estimate is within 30% of the actual number of packets for 80% of the flows, and within 20% of the actual number of bytes, for 90% of the flows.

Looking solely at the percentage error is somewhat misleading, since the high error cases often correspond to flows with few packet transmissions (fewer than 10). The main sources of inaccuracy are flows that after the TCP handshake had occurred saw a single reject or RST packet from the server. The clients in such cases attempts to send the initial data packet several times. This typically occurred in P2P connections that were refused. If our algorithm sees the server side of such flows it estimates it missed either 0 or 1 packets because we ignore RST packets. Otherwise, if it sees the client side of the flow it thinks it missed several ACKs because we assume all packets are acknowledged. In both cases, the algorithm is off by a few packets but the percentage error is large. We found that the overall average error per flow is 2.4 packets, and that 87% of flows are within 5 packets of the actual number. In terms



a) Flow Accuracy b) Byte Accuracy

Figure 7.7: Accuracy with Estimation Algorithm Estimating Server-to-Client Statistics

of bytes, the overall average error is 120 bytes, and 92% of the flows are within 500 bytes of the actual number.

7.2.5 Classification Using Estimated Statistics

We examine the accuracy of our classifier if we use the estimation algorithm described in the previous section to estimate server-to-client statistics for our traces when only the client-to-server or random flows are seen.

Figure 7.7 shows the classification accuracy. These experiments are similar to those reported in Figure 7.1, except that both model building and the subsequent classifier use the estimated statistics when necessary. As seen in this figure, we find that when we use the estimation algorithm to estimate the server-to-client statistics the flow accuracy and byte accuracy achieved using the client-to-server and random flows is very close to the actual accuracy we achieved when using the actual server-to-client statistics obtained from the empirical traces.

Interestingly, our classification accuracies are largely unaffected by the potential

errors in our estimated flow statistics. We think this robustness is due to the fact that we use the logarithm of the flow statistics (as mentioned in Section 7.1). The magnitude of difference of the flow statistics has a much greater impact in the classification than the small errors in the estimations. This makes us believe it is possible to use our estimation technique to calculate the different statistics which allows for the best classification even though only partial information is available.

7.3 Summary

This chapter considered the problem of classifying network traffic when only one direction of network flows are observed, as may be the case in the network core. We found that, in general, rich traffic classification using only unidirectional statistics is feasible, with our experiments showing accuracies of 95% in terms of flows and 80% in terms of bytes. We also found that better classification performance is achieved when statistics for the server-to-client direction are used than when statistics for the client-to-server direction are used. Because collection of the server-to-client statistics may not always be feasible, we developed and validated an algorithm that can estimate the missing statistics from a unidirectional packet trace.

The next chapter concludes this thesis and presents avenues for future work.

Chapter 8

Summary and Conclusions

This chapter summarizes the work done in this thesis. An overview of the thesis and its contributions are presented in Section 8.1 and Section 8.2, respectively. Section 8.3 discusses the conclusions we have drawn from our work. Finally, related open research problems are discussed in Section 8.4.

8.1 Thesis Summary

This thesis proposed a semi-supervised traffic classification framework that relies on using only flow statistics to classify traffic. We designed this approach to overcome the limitations and drawbacks of port-based and payload-based classification techniques. We provide an extensive evaluation of our proposed classification framework and of our design decisions; we believe these will be useful to future researchers and practitioners when applying machine learning techniques to this and other classification problems.

Chapter 1 presented the goals of the thesis.

Chapter 2 presented background and relevant prior work. Specifically, we discussed the TCP/IP protocol suite and the different traffic classification techniques in the literature. Historically, port [38] and payload [35, 52, 67] based approaches have been used for traffic classification in the literature and by commercial vendors. However, these approaches have several drawbacks such as being either increasingly ineffective or incurring high overhead, hampering their deployment. While these drawbacks have received some attention [35,44,49], they have spurred new traffic classification techniques to be developed based on using the behaviours of hosts [41,42,74]

and using machine learning [6, 17, 53, 72].

Chapter 3 described our data sets and our methodology of collecting traces. This research was facilitated by over 1 terabyte of full-payload packet traces collected over a 6-month time period from the University of Calgary. We described the establishment of a "base truth" classification of the flows in our data sets using a payload-based approach. We presented a breakdown of the application types we found predominantly in our base truth classification results of our traffic traces. We provided empirical evidence that re-affirmed the unreliability of port-based classification to classify traffic accurately.

The contributions of this work were presented in Chapters 4 - 7, and are summarized in the next section.

8.2 Contributions Summary

Chapter 4 formally described the semi-supervised classification framework proposed in this thesis. One of the specific contributions made is that we designed a flexible mathematical framework that leverages unlabelled flows and is not restricted to a specific clustering algorithm. This semi-supervised framework recognizes that obtaining labelled flows is hard and that not all applications are known *a priori*.

Chapter 5 investigated the potential of several clustering algorithms for use in the proposed semi-supervised framework. We showed that all the clustering algorithms considered can produce "pure" clusters and we described how a classifier for each algorithm can be designed. Furthermore, we discussed our decision to use K-Means in our classifier.

Chapter 6 evaluated the offline and realtime classifiers developed. Our results show that:

1. Both high flow and byte accuracy can be achieved;

2. A variety of applications, including P2P, HTTP, FTP, and email can be successfully classified; and,

3. Robust classifiers can be built that are immune to transient changes in network conditions.

Furthermore, to facilitate automatic detection of non-transient changes such as introduction of new applications or behavioural changes to existing applications, we proposed a retraining point detection mechanism. A salient feature of our work is the development of working prototypes.

Chapter 7 considered the problem of classifying network traffic when only one direction of network flows are observed, as may be the case when the point-of-observation is the network core. To address this problem, we applied our semi-supervised classification framework for classifying network traffic to only use unidirectional flow statistics.

The results show that, in general, rich traffic classification using only unidirectional statistics is feasible, with our experiments showing accuracies of 95% in terms of flows and 80% in terms of bytes. We also found better classification performance is achieved when statistics for the server-to-client direction are used than when statistics for the client-to-server direction are used. Because collection of the server-to-client statistics may not always be feasible, we developed and validated an algorithm that can estimate the missing statistics from a unidirectional packet trace.

8.3 Conclusions

This thesis proposed and evaluated a semi-supervised framework for classifying network traffic using only flow statistics. A key advantage of the semi-supervised approach is the ability to accommodate both known and unknown flows during development of the classifier. We show that our technique can achieve high classification

accuracy by using only a few labelled examples in combination with many unlabelled examples.

We concentrate on achieving both high flow accuracy and high byte accuracy, unlike many of the other proposed traffic classification approaches in the literature (e.g., [6, 17, 19, 72]). As we discussed, byte accuracy is specific to the problem of traffic classification wherein a minority of the flows (e.g., the "elephants") have a much greater impact on the network, and therefore, is an important aspect that cannot be disregarded. We found that achieving high flow accuracy is relatively easy. The more difficult problem is obtaining a high byte accuracy as well.

Our evaluations show that generic classifiers based on flow statistics can be developed. A vendor may train the classifier using a mix of labelled and unlabelled flows, where labelled flows may be obtained from operational or test networks. Our retraining point detection enables the possibility of discovery of network specific unknowns; these may be, at the discretion of the operator, delivered to the vendor for labelling. As and when required, vendors may distribute new classifiers along with a set of labelled flow feature vectors.

One challenge to our proposed approach is the selection of relevant features. This problem is not specific to our proposed classification framework; it is applicable to all machine learning problems in general. It is not possible to prove that a particular set of features is optimum. Some features might prove to be better discriminators for some application types than others; thus, depending upon the mixture of the traffic in the data sets it is possible for a different combination of features to provide better accuracy. However, as we have demonstrated, with appropriate selection of features, highly accurate classification using only flow statistics is possible. If application behaviours change or if the mix of applications on a network is different the optimum set of features may change. This is an open problem, but is nevertheless still manageable, because there exists a rich set of tools and methods in the machine

learning literature for feature selection [23,46].

8.4 Future Work

Many opportunities exist for future work, several of which have already been mentioned earlier in the thesis. These opportunities are further elaborated upon in this section.

Performance Comparisons

One potential area for future work is comparing the performance of our classification framework against existing classification techniques. In general, an extensive comparison between the existing approaches in the literature is missing and would be an interesting avenue to pursue. A study of this nature would be non-trivial for several reasons. First, publicly available data sets with a reliable "base truth" (i.e., not from port-based analysis) are not available due to privacy issues. This makes the reproduction of results impossible. Second, many of the current proposed approaches only attempt to classify a small subset of applications and this subset differs between studies. Third, the performance metrics used to evaluate techniques varies widely. For example, byte accuracy is ignored in the results of several studies [6,17,19,72]. Finally, many of the techniques have different tuning parameters (e.g., [41,42,74]) and use different features (e.g., [6,17,53,72]).

Exploring Different Design Choices

The development and evaluation of our semi-supervised classification framework focused on the use of K-Means to build our offline and realtime classifiers. Many clustering algorithms are available from the rich and diverse machine learning literature. For example, we built a classifier based on "hard" clustering where a flow is

always assigned to a most probable cluster; soft clustering techniques allow a flow to be probabilistically assigned to multiple clusters. In information theory, soft decisions have been shown to increase accuracy in certain situations. Another extension could be to evaluate different layering schemes for the realtime classifier.

Realtime Classification Performance

In our evaluation of our offline and realtime prototype classifiers we did not consider classification speed. Classification speed is most relevant to the problem of realtime classification as the timeliness of classification can affect the network's performance. This is not as applicable to offline classification because the flow has already ended. Our realtime classifier in terms of performance was slow and took approximately three hours to process a one hour trace. This aspect was not discussed in our results because we did not attempt to optimize the prototypes for performance. In fact, we added significant additional overhead to the realtime classifier as we measured its accuracy against the payload-based "base truth" and recorded its incremental classifications at each layer. By simply removing these measurements we could greatly increase the processing speed of the classifier.

We believe there exists several other opportunities to increase the speed of the classifier. First, collecting flow statistics in Bro accounted for the majority of the overhead in the classifier. Commercial products (e.g., Cisco's NetFlow [13]) optimized for collecting flow statistics could be integrated to reduce this overhead. Second, the hardware used in our evaluations was an IBM x335 series server with an Xeon 2.4GHz CPU and 1GB of RAM. This is not the most up-to-date hardware available and we think that Bro was constrained by the amount of RAM available on the machines. Finally, in our realtime prototype classifier our classification rules were generated in a Bro script that was interpreted by Bro as it ran. However, if we moved our classification rules into C code that is then compiled into Bro, then

we believe this could greatly improve the performance. In addition, the classification rules could be hierarchically structured to reduce the number of comparisons required.

Classifying UDP Traffic

The evaluation of our classification framework focused on TCP traffic. The framework can be extended, as discussed in Section 3.3, to classify UDP traffic. Flow statistics for UDP flows could be collected and a classifier built to handle this new type of flow. In addition, new features might need to be chosen to effectively discriminate between UDP-based applications.

Developing Generic Classifiers

Currently, we are investigating the applicability of classifiers developed using training data from one environment in successfully classifying traffic from another environment. Preliminary experiments with classifying the Campus traces using classifiers developed from the WLAN trace are encouraging. We are now trying to obtain traces from other environments to further validate our preliminary observations.

Bibliography

[1] A. A. Ali and R. Tervo. Traffic Identification Using Bayes' Classifier. In *Canadian Conference on Electrical and Computer Engineering*, Halifax, Canada, March 2000.

[2] M. Ankerst, M. M. Breunig, H. Kriegel, and J. Sander. OPTICS: Ordering Points to Identify the Clustering Structure. In *SIGMOD '99*, Philadelphia, USA, June 1999.

[3] S. Basu, M. Bilenko, and R. Mooney. A Probabilistic Framework for Semi-Supervised Clustering. In *KDD '04*, Seattle, USA, August 2004.

[4] L. Bernaille and R. Teixeira. Early Recognition of Encrypted Applications. In *PAM'07*, Louvain-la-neuve, Belgium, April 2007.

[5] L. Bernaille, R. Teixeira, I. Akodkenou, A. Soule, and K. Salamatian. Traffic Classification on the Fly. *Computer Communications Review*, 36(2):23–26, 2006.

[6] L. Bernaille, R. Teixeira, and K. Salamatian. Early Application Identification. In *CoNEXT'06*, Lisboa, Portugal, December 2006.

[7] BitTorrent. http://www.bittorrent.com.

[8] BitTorrent Protocol Encryptioin. http://en.wikipedia.org/wiki/protocol_encryption.

[9] Cache Logic. Peer-to-Peer in 2005. http://www.cachelogic.com/home/pages/research/p2p2005.php, 2005.

[10] O. Chapelle, B. Schölkopf, and A. Zien, editors. *Semi-Supervised Learning.* MIT Press, Cambridge, USA, 2006.

[11] P. Cheeseman and J. Strutz. Bayesian Classification (AutoClass): Theory and Results. *In Advances in Knowledge Discovery and Data Mining, AAI/MIT Press, USA*, 1996.

[12] T. Choi, C. Kim, S. Yoon, J. Park, H. Kim, H. Chung, and T. Jesong. Content-Aware Internet Application Traffic Measurement and Analysis. In *IEEE/IFIP NOMS'04*, Seoul, Korea, April 2004.

[13] Cisco NetFlow. http://www.cisco.com/warp/public/732/tech/netflow.

[14] C. Colman. What to do about P2P? *Network Computing Magazine*, 12(6), November/December 2003.

[15] F. Constantinou and P. Mavrommatis. Identifying Known and Unknown Peer-to-Peer Traffic. In *NCA'06*, Cambridge, USA, July, 2006.

[16] F. Cozman, I. Cohen, and M. Cirelo. Semi-Supervised Learning of Mixture Models. In *ICML'03*, Washington, USA, August 2003.

[17] M. Crotti, M. Dusi, F. Gringoli, and L. Salgarelli. Traffic Classification through Simple Statistical Fingerprinting. *Computer Communications Review*, 37(1):7–16, 2007.

[18] M. Crotti, F. Gringoli, P. Pelosato, and L. Salgarelli. A Statistics Approach to IP-level Classification of Network Traffic. In *ICC'06*, Istambul, Turkey, June 2006.

[19] I. Dedinski, H. De Meer, L. Han, L. Mathy, D. Pezaros, J. Sventek, and Z. Xiaoying. Cross-Layer Peer-to-Peer Traffic Identification and Optimization Based on Active Networking. In *7th International Working Conference on Active and Programmable Networks*, Sophia Antipolis, France, November 2005.

[20] S. Deering and R. Hinden. Internet Protocol, Version 6 (IPv6) Specification. RFC 2460 (Draft Standard), December 1998.

[21] A. P. Dempster, N. M. Paird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, 39(1):1–38, 1977.

[22] C. Dews, A. Wichmann, and A. Feldmann. An Analysis of Internet Chat Systems. In *IMC'03*, Miami Beach, USA, October 2003.

[23] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification.* Wiley, New York, USA, 2nd edition, 2001.

[24] N. Duffield, C. Lund, and M. Thorup. Properties and Prediction of Flow Statistics from Sampled Packet Streams. In *IMW '02*, Marseille, France, August 2002.

[25] N. Duffield, C. Lund, and M. Thorup. Flow Sampling Under Hard Resource Constraints. In *SIGMETRICS'04*, New York, USA, June 2004.

[26] M. Dunham. *Data Mining: Introductory and Advanced Topics.* Prentice Hall, New Jersey, 1st edition, 2003.

[27] J. Erman, M. Arlitt, and A. Mahanti. Traffic Classification using Clustering Algorithms. In *SIGCOMM'06 MineNet Workshop*, Pisa, Italy, September 2006.

[28] J. Erman, A. Mahanti, and M. Arlitt. Internet Traffic Identification using Machine Learning. In *GLOBECOM'06*, San Francisco, USA, November 2006.

[29] J. Erman, A. Mahanti, M. Arlitt, I. Cohen, and C. Williamson. A Semi-Supervised Approach to Network Traffic Classification. In *SIGMETRICS'07 (Extended Abstract)*, San Diego, USA, June 2007.

[30] J. Erman, A. Mahanti, M. Arlitt, I. Cohen, and C. Williamson. Offline/Online Traffic Classification Using Semi-Supervised Learning. Technical report, University of Calgary, 2007.

[31] J. Erman, A. Mahanti, M. Arlitt, and C. Williamson. Identifying and Discriminating Between Web and Peer-to-Peer traffic in the Network Core. In *WWW'07*, Banff, Canada, May 2007.

[32] C. Estan, K. Keys, D. Moore, and G. Varghese. Building a Better NetFlow. In *SIGCOMM '04*, Portland, USA, August 2004.

[33] M. Ester, H. Kriegel, J. Sander, and X. Xu. A Density-based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *KDD'96*, Portland, USA, August 1996.

[34] I. Guyon and A. Elisseef. An Introduction to Variable and Feature Selection. *Journal of Machine Learning Research*, pages 1157–1182, 2003.

[35] P. Haffner, S. Sen, O. Spatscheck, and D. Wang. ACAS: Automated Construction of Application Signatures. In *SIGCOMM'05 MineNet Workshop*, Philadelphia, USA, August 2005.

[36] J. He, A. Tan, C. L. Tan, and S. Y. Sung. *Clustering and Information Retrieval*, chapter On Quantitative Evaluation of Clustering Systems, pages 105–134. Kluwer, 2003.

[37] F. Hernàndez-Campos, F. D. Smith, K. Jeffay, and A. B. Nobel. Statistical Clustering of Internet Communications Patterns. *Computing Science and Statistics*, 35, July 2003.

[38] IANA. Internet Assigned Numbers Authority (IANA). http://www.iana.org/assignments/port-numbers.

[39] A. K. Jain and R. C. Dubes. *Algorithms for Clustering Data.* Prentice Hall, Englewood Cliffs, USA, 1988.

[40] T. Karagiannis, A. Broido, and N. Brownlee. Is P2P Dying or Just Hiding? In *GLOBECOM '04*, Dallas, USA, November 2004.

[41] T. Karagiannis, A. Broido, M. Faloutsos, and kc claffy. Transport Layer Identification of P2P Traffic. In *IMC'04*, Taormina, Italy, October 2004.

[42] T. Karagiannis, K. Papagiannaki, and M. Faloutsos. BLINC: Multilevel Traffic Classification in the Dark. In *SIGCOMM'05*, Philadelphia, USA, August 2005.

[43] R. R. Kompella and C. Estan. The Power of Slicing in Internet Flow Measurement. In *IMC'05*, Berkeley, USA, October 2005.

[44] S. Kumar, S. Dharmapurikar, F. Yu, P. Crowley, and J. Turner. Algorithms to Accelerate Multiple Regular Expressions Matching for Deep Packet Inspection. In *SIGCOMM '06*, Pisa, Italy, September 2006.

[45] J. Kurose and K. Ross. *Computer Networking: A Top-down Approach Featuring the Internet.* Addison Wesley, 2003.

[46] P. Langley. Selection of Relevant Features in Machine Learning. In *Proceedings of the AAAI Fall Symposium on Relevance*, New Orleans, USA, 1994.

[47] W. Li and A. Moore. Learning for Accurate Classification of Real-time Traffic. In *CoNEXT'06 Student Workshop*, Lisboa, Portugal, December 2006.

[48] lindump. http://awgn.antifork.org/codes/lindump.c.

[49] J. Ma, K. Levchenko, C. Krebich, S. Savage, and G. Voelker. Unexpected Means of Protocol Inference. In *IMC'06*, Rio de Janeiro, Brasil, October 2006.

[50] A. Madhukar and C. Williamson. A Longitudinal Study of P2P Traffic Classification. In *MASCOTS'06*, Monterey, USA, August 2006.

[51] A. McGregor, M. Hall, P. Lorier, and J. Brunskill. Flow Clustering Using Machine Learning Techniques. In *PAM'04*, Antibes Juan-les-Pins, France, April 2004.

[52] A. Moore and K. Papagiannaki. Toward the Accurate Identification of Network Applications. In *PAM'05*, Boston, USA, March 2005.

[53] A. Moore and D. Zuev. Internet Traffic Classification Using Bayesian Analysis Techniques. In *SIGMETRIC'05*, Banff, Canada, June 2005.

[54] A. Moore, D. Zuez, and M. Crogan. Discriminators for use in flow-based classification. Technical report, Queen Mary, Univeristy of London, August 2005.

[55] T. Mori, M. Uchida, R. Kawahara, J. Pan, and S. Goto. Identifying Elephant Flows through Periodically Sampled Packets. In *IMC '04*, Taormina, Italy, October 2004.

[56] T. Nguyen and G. Armitage. Synthetic Sub-flow Pairs for Timely and Stable IP Traffic Identification. In *Australian Telecommunication Networks & Applications Conference (ATNAC)*, Australia, December 2006.

[57] T. Nguyen and G. Armitage. Training on multiple sub-flows to optimise the use of Machine Learning classifiers in real-world IP networks. In *LCN'06*, Tampa, USA, November 2006.

[58] Packeteer. http://www.packeteer.com/.

[59] V. Paxson. Bro: A System for Detecting Network Intruders in Real-Time. *Computer Networks*, 31(23-24):2435–2463, 1999.

[60] V. Paxson. Empirically-Derived Analytic Models of Wide-Area TCP Connections. *IEEE/ACM Transactions on Networking*, 2(4):316–336, August 1998.

[61] J. Postel. User Datagram Protocol. RFC 768 (Standard), August 1980.

[62] J. Postel. Internet Protocol. RFC 791 (Standard), September 1981. Updated by RFC 1349.

[63] J. Postel. Transmission Control Protocol. RFC 793 (Standard), September 1981. Updated by RFC 3168.

[64] M. Roesch. Snort: Lightweight Intrusion Detection for Networks. In *13th Systems Administration Conference (LISA), USENIX*, pages 229–238, November 1999.

[65] 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 *IMC'04*, Taormina, Italy, October 2004.

[66] W. Saddi, N. B. Azzouna, and F. Guillemin. IP Traffic Classification via Blind Source Separation based on Jacobi Algorithm. In *ECUMN'04*, Porto, Portugal, October 2004.

[67] S. Sen, O. Spatscheck, and D. Wang. Accurate, Scalable In-Network Identification of P2P Traffic Using Application Signatures. In *WWW'04*, New York, USA, May 2004.

[68] S. Sen and J. Wang. Analyzing Peer-to-Peer Traffic across Large Networks. *IEEE/ACM Transactions on Networking*, 12(2):219–232, 2004.

[69] F. Donelson Smith, F. Hernàndez-Campos, K. Jeffay, and D. Ott. What TCP/IP Protocol Headers Can Tell us about the Web. In *SIGMETRICS '01*,

Cambridge, USA, June 2001.

[70] Q. Sun, D. R. Simon, Y. Wang, W. Russell, V. N. Padmanabhan, and L. Qiu. Statistical Identification of Encrypted Web Browsing Traffic. In *SP '02*, Oakland, USA, May 2002.

[71] S. Tong. *Active Learning: Theory and Applications.* PhD thesis, Standford University, August 2001.

[72] N. Williams, S Zander, and G. Armitrage. A Preliminary Performance Comparison of Five Machine Learning Algorithms for Practical IP Traffic Flow Classification. *Computer Communication Review*, 30:5–16, October 2006.

[73] I. H. Witten and E. Frank. *(2005) Data Mining: Pratical Machine Learning Tools and Techniques.* Morgan Kaufmann, San Francisco, 2nd edition, 2005.

[74] K. Xu, Z. Zhang, and S. Bhattacharyya. Profiling Internet Backbone Traffic: Behavior Models and Applications. In *SIGCOMM '05*, Philadelphia, USA, August 2005.

[75] S. Zander, T. Nguyen, and G. Armitage. Automated Traffic Classification and Application Identification using Machine Learning. In *LCN'05*, Sydney, Australia, November 2005.

[76] S. Zander, T. Nguyen, and G. Armitage. Self-Learning IP Traffic Classification Based on Statistical Flow Characteristics. In *PAM'05*, Boston, USA, March 2005.

[77] S. Zander, N. Williams, and G. Armitage. Internet Archeology: Estimating Individual Application Trends in Incomplete Historic Traffic Traces. In *PAM'06*, Adelaide, Australia, March 2006.

[78] Y. Zhang and V. Paxson. Detecting Backdoors. In *USENIX Security Symposium*, Denver, USA, August 2000.

[79] D. Zuev and A. Moore. Traffic Classification using a Statistical Approach. In *PAM'05*, Boston, USA, March 2005.

Appendix A

Payload Signatures

These are the payload-based signatures we used in Bro to create our classification base truth. Note $\backslash x$ signifies a hex character.

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

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

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 smtp_id {
ip-proto = tcp
payload /.*(SMTP|ESMTP)/
event "SMTP"
}

signature smtp2_id {
dst-port = 25
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"
}
```