INSTITUT FÜR INFORMATIK
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Forschungs- und Lehreinheit I
Angewandte Softwaretechnik

Supervised Machine Learning Assisted Real-Time Flow Classification System

A Real-Time Approach to Flow Classification

Isara Anantavrasilp

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender: Univ.-Prof. Dr. Georg Carle

Prüfer der Dissertation: 1. Univ.-Prof. Bernd Brügge, PhD.

2. Univ.-Prof. Dr. Dr. h. c. Alexander Schill
Technische Universität Dresden

Die Dissertation wurde am 17.05.2010 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 24.09.2010 angenommen.

To Mankind, Science . . . and You.

Abstract

A Flow Classification System (FCS) is a process and mechanism that assigns a class to a network connection (flow). In QoS-aware networks, QoS-aware applications can identify and assign service classes to their flows. The flows are then treated by the networks according to their classes. However, most of the existing network applications are QoS-unaware applications, prompting a need for an enhanced FCS that can automatically identify the service classes of the flows.

This dissertation describes a new FCS, called Supervised Machine learning Assisted Real-Time (SMART) Flow Classification System, designed to classify QoS-unaware flows in real-time. It uses a novel concept of flow prefix, which refers to a certain number of flow packets. We empirically show that the characteristics of a flow can be estimated by observing only up to a specific prefix. Evaluations on benchmark datasets have shown that observing only 11 packets is sufficient to achieve more than 90% classification accuracy.

SMART uses a machine learning algorithm to automatically identify relationships between the characteristics and the classes of the flows from QoS-aware applications. The learned relationships are then used to identify the QoS-unaware flows. We have evaluated our SMART FCS over a variety of real-world data, including flow samples collected from individual users and a large dataset collected from an edge router of an organizational network. The results show that our approach achieves average correctness of 98.82% and 99.66% in individual-users and large-network benchmark datasets, respectively.

Acknowledgement

I owe my deepest gratitude to my Doktorvater, Prof. Bernd Brügge, professor for Applied Software Engineering at Technische Universität München, who has supervised and guided me during my research. Not only providing me valuable scientific and technical advices, he also motivated and encouraged me with perpetual energy and enthusiasm. It is an honor for me to work with him.

I am heartily thankful to Prof. Alexander Schill, professor for Computer Networks and Prof. Steffen Hölldobler, professor for Knowledge Representation and Reasoning at Dresden University of Technology. This thesis would not have been possible without their kindly help, suggestions as well as detailed and constructive comments. I am deeply grateful to my supervisor at BenQ Mobile GmbH & Co. OHG, Dr. Thorsten Schöler, for his important support and throughout this work. I also warmly thank Dr. Kenjiro Cho, Deputy Research Director at Internet Initiative Japan, Inc., and the WIDE Project, Japan, for their valuable data.

I am indebted to my friends and colleagues at TU-Dresden, TU-München and BenQ Mobile for their scientific suggestions, including Ari Saptawijaya, Tobias Pietzsch, Boontawee Suntisrivaraporn, Sebastian Bader, Bertram Fronhöfer, Surapa Thiemjarus, Kiattisak Roonprasang, Tansir Ahmed, Petr Osipov, Arsalan Minhas, Dennis Pagano, Damir Ismailović, Florian Schneider, Helmut Naughton, Nitesh Narayan, and Yang Li. I am also pleased to thank Araya Raiwa, Surapa Thiemjarus, Teerapat Anantavarasilpa, Sujitra Thongjab and Phee for their tireless efforts in data collection. My special thanks go Suvaporn Photjananuwat for the wonderful cover of this thesis.

My most sincere and warmest gratitude go to my beloved families, Baumeister and Anantavrasilp, especially Mama and Papa, mom and dad, and Yai. Without their advices, support and encouragement, this thesis would not be possible — Thank you all for believing in me. In addition, I would also like to thank Bee, my brother, who has proof-read every single word of this thesis.

I would like to show my gratitude to my friends in Thailand, Germany and the United Kingdom for trusting and believing in me.

Overview

Chapter 1
Introduction to the research
Motivation and need of real-time adaptive flow classification,
Problem description and challenges
Overview of Supervised Machine learning Assisted Real-Time (SMART) flow classification system
Research contributions

Chapter 2
Overview of computer networks and quality-of-service (QoS) support
Discussion of flow classification system components
Survey and comparisons of previous systems

Chapter 3
Rigorous and unified mathematical framework for flow classification
System decomposition of SMART using Unified Modeling Language (UML)
Reviews of previous FCSs using the proposed framework

Chapter 4
In-depth review and analysis of current machine learning algorithms
Definitions of their performance measurements

Chapter 5
Description of the SMART and its components
Extensive evaluations of the system on individual-user and large-network datasets

Chapter 6
Exploring the possibilities of real-time flow classification
Extension of SMART to support real-time classification and empirical evaluations of SMART

Chapter 7
Conclusion of research
Directions for future works

Appendix A
Signatures of the communication protocols used in experiments

Appendix B
Additional evaluation results

Typographical Conventions

The following table explains the typographical conventions used in this thesis.

Table 1: Typographical Conventions

Formatting Convention	Type of Information	Example
"Quoted"	Introduction of a term.	"Classifier"
Italics	Used to emphasizing the importance of a point or to indicate a mathematical notation.	A learner can be trained to *learn*. $prefix(f, 3) = (p_1, p_2, p_3)$

`Monospaced`	Elements in UML diagrams, program codes, computer input or output, such as protocol signatures and classification rules.	`If a AND b THEN x`
		
		`FCS Component`

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Mobile devices, such as smartphones or Personal Digital Assistants (PDAs), are getting smaller and faster everyday. They are not meant to just make phone calls, create short notes or facilitate simple organizing; common applications now include sending emails, playing videos, watching television, attending videoconferences, playing online games and processing other tasks that would only be possible on a desktop computer just a few years ago. This is due to the emerging computing and wireless networking technologies as well as the growing demand of the users who want to have more connectivity, usability and entertainment, while at the same time also need to maintain their mobility. Current network services have also evolved from just sending emails or web browsing into real-time offerings such as videoconference or broadcasting, which consequently demand much higher network performance than traditional applications. In response, mobile networks are also getting faster, as we are now moving from typical Global System for Mobile communications (GSM) and General Packet Radio Service (GPRS) to Universal Mobile Telecommunications System (UMTS), High-Speed Packet Access (HSPA), Worldwide Interoperability for Microwave Access (WiMAX) and Long Term Evolution (LTE). These new technologies offer better network performances such as higher data rate or lower delay.

While it could be argued that applications demand can always be met by constantly increasing network capacity, especially in wireless links, simply increasing network performance might not be possible due to physical limitations. This issue also holds in the case of wired networks, especially in home-network where members of a household share the same Internet connection. To this end, resource management schemes, such as quality-of-service (QoS) management, are required. Informally, QoS can be described as a measurement on how the user experiences the services [Räi03].

Recent network standards, such as Internet Protocol version 6 (IPv6), 802.11e Wireless LAN, WiMAX, UMTS and LTE incorporate QoS management schemes into their specifications. The implemented schemes are based on "Differentiated Services (DiffServ)" architecture, under which an application in a network node can specify an appropriate service class to its network connection or "flow". The packets within the flows are then marked with a "service class" indicating the flow's type of service so that the network can treat each of them appropriately. This way, packets requiring, for instance, lower transfer delay or higher throughput will be served first, whereas those

that can sustain higher delay or less speed will be put on hold or even discarded. The downside of the DiffServ approach is that explicit network capacities or resources are not guaranteed to the flows. Another QoS management approach, "Integrated Services (IntServ)", is designed to specifically address this issue by allowing applications to reserve network resources along the communication route. This is done through a resource reservation protocol. However, due to its scalability problem, it is not widely-implemented [Räi03]. Therefore, this research is restricted to the DiffServ approach.

Regardless of which QoS management scheme is employed, effective QoS management within a network requires certain applications to specify the QoS requirement to their flows. We call such applications "QoS-aware applications". In contrast, the commonly used applications at present are designed based on the best-effort scheme. These so-called "QoS-unaware applications" are not aware of the concept of QoS and do not specify any classes to their flows — hence they cannot receive the QoS-support provided by the network. Therefore, a mechanism that can correctly assign the service classes to the flows is essential. The class assignment process is called "flow classification" and the mechanism that carries out the classification is called a "flow classification system (FCS)". The classification process has to be carried in a very short period of time so that the flows can promptly benefit from the QoS support. Because new, unseen applications or services can be dynamically introduced to the network at anytime, the flow classification system must be able to handle them as well. The flow classification task can be formalized as follows.

Flow Classification Task Given a flow, the flow classification system has to assign to the flow a service class that matches the requirements of the flow.

Requirements

- Accuracy: The flow classification system must classify flows with high accuracy.
- Robustness: The classifier must be able to handle new or unknown applications.
- Generality: It also must be able to handle any kind of Internet applications.
- Real-time: Given certain classification accuracy, the classification time is bound by an upper limit.

1.1 Challenges

In the process of developing a practical technique to classify flows in real-time, several challenges and constraints have to be overcome.

1.1.1 Dynamic Nature of Internet Applications

Until recently, the Internet has been restricted to a small number of services such as Email, FTP, Telnet, Gopher, etc. These services are bound to specific transport protocol ports and connections of a service would be established only through designated ports. Moreover, the protocols are open and standardized. Through standardized port

assignments and protocol syntaxes, identifying the flows' services (and, in turn, their classes) is relatively easy. Along with the emergence of the World-Wide-Web (WWW) in the past decade, however, came new services, such as videoconferences, peer-to-peer file sharing (P2P) and online games. Most of these services are no longer using standardized transport protocol ports. Some use dynamic port assignments, while others, especially P2P applications, avoid using specific port numbers because they are usually used to transfer illegal media contents such as music or movie files. Using transport ports is therefore not effective in flow classification. To make the matters worse, the protocol syntax is in many cases proprietary and closed while packets could also be encrypted. Distinguishing flows based on protocol syntax or specific payload contents is also not feasible. Even though the flow classification system could be pre-programmed to handle non-standardized protocols, it might not be able to handle applications using unknown protocols or even updated versions of existing protocols.

1.1.2 Generality

Recent network services, such as videoconference and online games, implement the UDP protocol to minimize delays. Thus, at the network level, the flow classification system has to be general enough to handle not only TCP but also UDP protocol. At the application level, the system must be able to handle as many applications as possible.

1.1.3 Classifying Flows in Real-Time

To provide appropriate and timely service quality to flows, the classification has to be done in a bounded period of time — ideally immediately after the flow is seen. This task is not always straightforward as the flow information at the classification time may be limited.

1.1.4 Lack of General Flow Classification Model

Flow classification has played an important role in many network activities. A lot of research has been carried out for the past several years targeting different aspects including quality-of-service support, security, management and provisioning. As a result, different systems have been developed without a common underlying model that can explain the common components and processes, making analyzing, comparing and understanding the relationships among different approaches rather difficult [SOMS08]. As a matter of fact, without a unambiguous model that precisely describes the flow classification task, understanding the task itself could prove to be a formidable challenge.

1.2 Real-Time Adaptive Flow Classification System

In this section, an overview of the a new flow classification system, called "Supervised Machine learning Assisted Real-Time Flow Classification System (SMART)", is

provided. We will also briefly discuss how our approach fulfills the requirements and overcomes the aforementioned challenges.

A flow classification process, in general, consists of three main processes — flow observation, feature extraction and classification. In the flow observation process, the QoS-unaware flow to be classified is observed. Essentially, a flow is a sequence of packets that belong to the same connection and the observation is carried out by a packet capturing sensor that resides in a network device.

In the feature extraction process, the observed flow is transformed into abstract representations called "features", which, in principle, are the characteristics of the flow that we are interested in. Examples of features include transport protocol, average throughput, packet counts and statistics of packet inter-arrival time (IAT). Each feature is computed by a designated "feature function". Multiple features of a single flow are usually presented together as a vector called "feature vector". The feature calculations and the feature vector construction is managed by the "feature extractor" or, in short, the "extractor". The set of features used by the system plays a very important role in the flow classification. If the features are discriminative, the flows belong to different classes would be easier to be distinguished from each other.

In this thesis, a set of features, which can be effectively used to distinguish flows is identified. The proposed set of features can be used on both TCP and UDP flows as required in our setting. We also investigate the discriminability of those features and single out the features that are the most useful to the classification using "feature selection" methods. Moreover, a study of the relationships between features and flow observation duration is provided. By incrementally scrutinizing the characteristics of the flows over observed "prefixes" (which refer to the number of observed packets), we identify the minimal number of packets required to be observed in order to obtain the preferred accuracy. The results of the analysis lead to an effective and accurate real-time flow classification system.

After a flow is abstracted into a feature vector, it will be classified by the "classifier", which in general is a set of rules that analyzes the values in the feature vector and assigns the class accordingly. In this dissertation, a set of five classes are defined. In contrast to previous works, the proposed classes are intended especially to assist QoS support and capture different service quality requirements. Figure 1.1 illustrates the classification process.

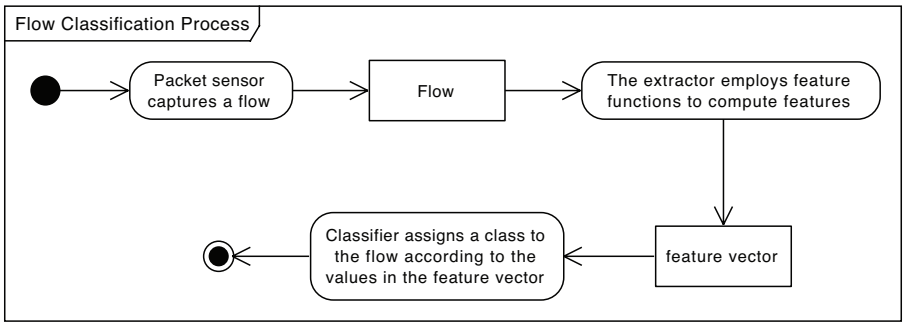


Figure 1.1: Activity diagram of a flow classification process.

Some FCSs in the literature are equipped with static classifiers, which cannot be automatically updated. These might not be able to effectively handle flows of unseen applications as the classifiers do not recognize them. The problem can be circumvented by issuing patches to update the classifier regularly (analogous to updating an anti-virus software) or by asking the user for classification assistance. Such approaches are, however, not practical. In response, our proposed FCS is equipped with a learner that is able to *learn* the relationship between the flow characteristics and their classes, as well as update their classifiers correspondingly. This way, the classifier has the ability to handle unseen flows without human-intervention — consequently making our FCS adaptive, robust and immune to the dynamic nature of the Internet.

In the learning process, the flows of QoS-aware applications are captured and extracted into feature vectors along with their service classes issued by the applications. The set of pairs of feature vectors and classes is then stored in a "dataset". Finally, the learner analyzes the relationships between the features and the classes in the dataset and induces a classifier accordingly. Figure 1.2 shows an activity diagram of the learning process. The quality of the induced classifier depends on the employed set of features and the learner. Therefore, in this thesis, in-depth evaluations of various machine learning techniques are carried out to determine which method is the most suitable for the flow classification task.

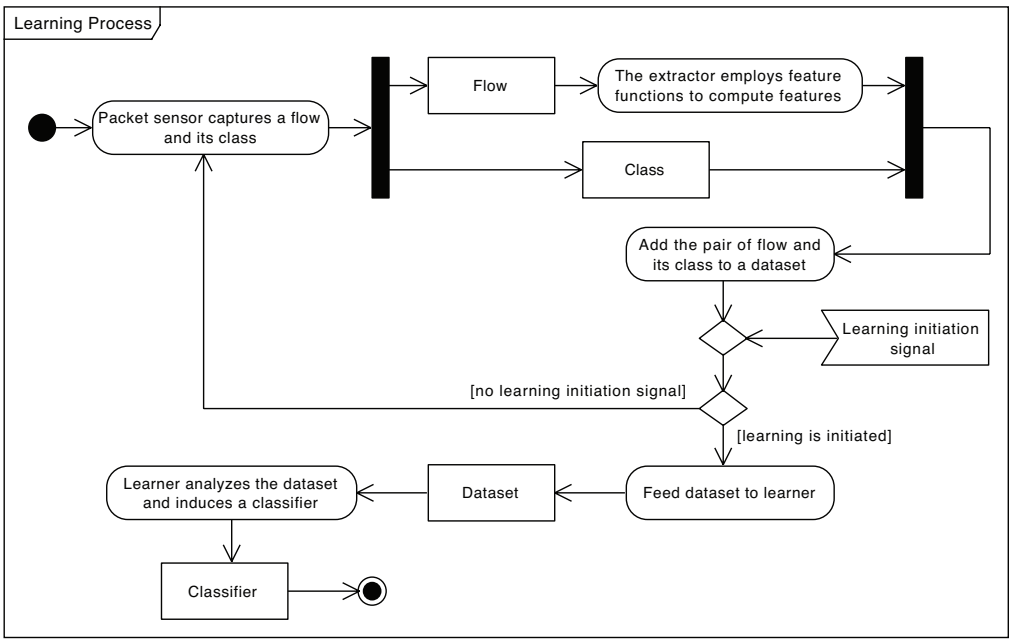


Figure 1.2: Activity diagram of a learning process.

As mentioned earlier, flow classification systems in the past several years have been developed and evaluated separately without a general model that can be used to identify, categorize and compare them. It is therefore difficult for a newcomer to understand the principle of flow classification and to find a suitable method for her domain. This thesis, in turn, provides a uniform model, which describes the

flow classification components and processes on an abstract level as well as allows us to classify existing flow classification approaches and compare them in a unified framework.

1.3 Contributions

This research introduces SMART, which is a novel flow classification system aimed to provide QoS support to QoS-unaware applications. Based on a new concept of flow prefix, our approach is able to identify the class of the given flow in real-time. The proposed method is accurate, adaptive, self-updatable and generic. Our work contributes to the existing network literature in a number of ways:

- We develop a unified mathematical model, which precisely describes the flow classification components and processes on an abstract level. This new model provides a fundamental framework in which different flow classification approaches can — for the first time — be distinctively described, compared and categorized.

- We propose a set of service classes intended for QoS requirements.

- The novel concept of flow prefix, which refers to a certain number of flow packets, is introduced.

- We empirically show in this thesis that the characteristics of a flow can be estimated by observing only up to a specific prefix as well as identify the optimal number of packets that should be observed.

- We present a set of features aimed to capture flow characteristics and also utilize feature selection techniques to identify the features that are most discriminative and useful to the classification. Moreover, the feature functions employed in this thesis are precisely and clearly defined. This will ensure the correct interpretations of the functions, which allows our research and experiments to be replicated. Comprehensive evaluations are conducted on real-world data collected from individual users and a large, diverse network. The results indicate that our set of features is discriminative and generic. Furthermore, we incrementally investigate the relationship between the features and prefixes and show that the characteristics of the flows can be captured using only partial flows. To the best of our knowledge, no such study has been conducted before.

- We review and evaluate flow features as well as several machine learning algorithms in order to identify the most suitable learner for flow classification. Unlike most of the studies in the literature, our research is focused on supervised learners that give human-comprehensible classification models as outputs. Not only can such models be used in the flow classification process but they can also be employed by experts to analyze the flow behaviors as well as the relationships between the flow characteristics and classes.

- An analytical framework is developed in this thesis as an experimental platform. Moreover, to ensure data integrity, a sophisticated signature-based flow classification system is introduced. We also identify a large number of signatures of various application protocols to be incorporated with the implemented FCS.

We believe that the research presented in this thesis will add value to many different areas in computer science. Researchers in the networking area can benefit directly from our novel real-time flow classification technique. Additionally, our solid mathematical classification model as well as extensive analysis on machine learning algorithms in a real-world scenario can also be useful to knowledge representation, machine learning and software engineering communities.

1.4 Thesis Structure

We begin with an introduction to computer networks and quality-of-service management in Chapter 2. Then, we will move on to a discussion on flow classification system and how to incorporate it with QoS-support. Existing flow classification systems implemented in many areas are also reviewed. In Chapter 3, a new unified mathematical model describing flow classification process and components is discussed. This model will be used to describe all processes and components throughout this dissertation. Apart from mathematical model, the implementation, deployment and system decomposition of SMART are also described using Unified Modeling Language (UML). Chapter 4 provides an overview of machine learning and a survey on current learning techniques. We will also discuss how to evaluate a learner as well as to define the performance measurement of a learning algorithm. Chapter 5 describes SMART, our new adaptive flow classification system. SMART employs machine learning algorithm, which allow it to update its classifier without human-intervention. It also uses a novel feature, throughput difference, which is intended to capture the burstiness of flows. How the learning algorithm and the feature be integrated into the FCS will be described in the chapter. In Chapter 6, a new concept of prefix is presented. Also, a study on the relationship between features and prefixes as well as a discussion about a novel real-time classification technique that requires only the smallest number of prefix are provided. Here, SMART employs another novel feature, packet-size difference, which can be used to capture the differences of packet sizes within a flow in real-time. We will also present the results of critical evaluations of the proposed technique and identify the most discriminative features using a feature selection method. Finally, Chapter 7 concludes the thesis with a discussion of our results and contributions, as well as directions for future research.

Chapter 2

Preliminaries

This chapter provides fundamental knowledge on network architecture, quality-of-service, basic principles and surveys on flow classification. The chapter is organized as follows. Introductions to computer networks and quality-of-service are provided in Section 2.1 and 2.3, respectively. Section 2.4 discusses how QoS-support can be provided to QoS-unaware flows. Finally, background and state-of-the-art on flow classification systems are described in Section 2.5.

2.1 Introduction to Computer Networks

In the wake of the 20th century, we have seen revolutions in computers and communication technologies. Computers are getting much smaller and more mobile. Recent communication technologies allow several computers to exchange data and, together, accomplish given tasks even though they are far apart. A system in which a collection of computers are connected together in the way that they can exchange information is called a "computer network" or simply a "network". We usually call a network by the protocols that are employed by that network. For example, we call a network whereby the computers are interconnected by the Internet Protocol (IP) an IP-network.

In this thesis, the term "computer" covers not only ordinary desktop or notebook computers but also mobile devices such as personal digital assistant (PDA) and smartphones. Since there are many kinds of devices that can be connected to a network, we will refer to them simply as "network devices", or more broadly as "host".

The hosts are physically linked together by "physical media". These media could be copper wire, optical fibers, radio signals, etc. In each computer, there exists a set of software that performs the tasks given by the user. These are called "applications". In a sense, an application is software that interacts with a user, receiving the tasks from a user, executing them and providing feedback. Applications can also communicate with other applications residing in other computers in the same network. Furthermore, in a single network device, there could be more than one application dedicated to different tasks.

As an example, consider a user, Alice, who wishes to establish a videoconference with another user Bob. Alice has a smartphone that has a camera as well as a piece of

software that can perform videoconferences. Indeed, Bob has to have a device, say a desktop computer, that possesses the same functionality. Here, Alice and Bob are the users. Alice's smartphone and Bob's computer are the hosts. Their videoconference clients are the applications. We will see now how the devices are connected together.

Classical communication technologies such as the telephone are based on analog signals. In an analog telephone network, for example, actual voice signal is transmitted as electrical voltage. The telephone network is operated on a so-called "circuit switching" scheme. When two network devices in this network want to connect to each other, a physical connection is established for both nodes. Here, the network devices could be telephones or modems. The connection media are the telephone lines. Although this methodology might guarantee that there would be no traffic congestion between the two nodes, it might not be efficient. It would be extremely expensive to have enough connection reservations for all pairs of network nodes. Furthermore, analog signals are also prone to error. Digital technology eliminates these issues [Tan03][LGW04].

Encoding analog data into series of 0s and 1s makes the data easier to recognize and thus more accurate. As the data can be encoded into streams of bits, they can be further cut into small chunks called "packets". Data packets enable another networking scheme, which is easier and cheaper to maintain, called "packet switching". Instead of establishing dedicated connection between the two hosts, packet switching network transfers data between the hosts through arbitrary paths. Each packet might travel through different paths. At the destination, all the packets are assembled to reconstruct the original data.

The entire communication process is governed by a set of rules. These rules — the "protocols" — define how two or more network devices communicate with each other. For instance, they define how the data are divided into packets and how the packets are delivered through the network. Generally, protocols are designed to work with each other in a stack-like manner. Each protocol operates only on its own level or "layer" and relies on the functionality of the protocol in the layer below it. This kind of network model is called a "layered" model. As an example, consider the aforementioned scenario where Alice wants to perform a videoconference with Bob. The videoconference applications used by both users must have adhere to the same protocol, e.g., H.323 protocol [H.306], to define how to establish a videoconference session. The H.323 protocol relies on the UDP protocol to divide the video data into packets and to relay those packets between the two hosts.[1] The UDP protocol in turn relies on the IP protocol to route the packets to the designated host. Finally, the IP protocol also relies on WiMAX protocol [80204] to deliver the data through the physical links.

It is important to note that the number of layers, functionality and the protocols in each layer may differ from network to network. In the following section, we will discuss a network architecture called the "Transmission Control Protocol / Internet Protocol architecture" or "TCP/IP architecture". It is the core architecture of the Internet and the architecture on which our research is based.

[1] A videoconference protocol might rely also on other protocols, such as Real-time Transport Protocol (RTP) [SCFJ03] or Real Time Streaming Protocol (RTSP)[SRL98] to control the video packets. However, for the sake of simplicity, we do not consider them here.

2.2 TCP/IP Architecture

The TCP/IP architecture is a network architecture consisting of two major protocols, TCP and IP. Although the name may suggest otherwise, the TCP/IP architecture has more than these two protocols. In this section, we will discuss how different protocols, which are designed for different tasks, work together and how they are incorporated to form a network architecture.

2.2.1 TCP/IP Reference Model

The "TCP/IP Reference Model" or, in short, the "TCP/IP model" is a layered abstract model for network protocol design introduced by the Internet Engineering Task Force (IETF) [Bra89]. It does not describe some particular protocols, but rather the communication functions, which should be performed by the protocols. It consists of four layers, arranged from top to bottom: Application, Transport, Internet and Network Access layer (see Figure 2.1). The upper layers are closer to the user. The lower layers are responsible for transmitting the data from the upper layers through the physical media. The following sections explain the operations of the TCP/IP layers from the top layer to the bottom one.

2.2.2 Application Layer

All application-oriented protocols, including the H.323 protocol, the Hypertext Transfer Protocol (HTTP) [BLFF, FGM+], the File Transfer Protocol (FTP) [PR85], etc., belong to this layer and we call them "application protocols". These protocols do not handle any data transportation or routing. They focus only on how to represent, reconstruct and interpret the data. Note that the protocols are not applications. They are actually sets of rules that applications in different hosts use to communicate with each other. Each protocol is designed to convey a certain task. For instance, H.323 is designed to deliver a videoconference session while HTTP is designed to transmit hypertext pages (e.g., web pages). There can be more than one application using the same protocol and there can be more than one protocol designed for the same task. For example, videoconference clients could be different, yet they can still be used together as long as they use the same video protocol (i.e., H.323).

2.2.3 Transport Layer (TCP Layer)

When the data are sent from the application layer to the transport layer, they are divided into packets with the "protocol header" added to each of them. The header contains protocol information, such as the transport port, header length, or packet checksum, which is required by the protocol to control the packet delivery. The most important thing in the transport protocol header is the "transport ports" or simply the "ports".

When applications on two hosts want to communicate with each other, one of them has to open a communication channel and wait for the incoming data. This channel is called a transportation port. When an application is waiting for incoming data, it

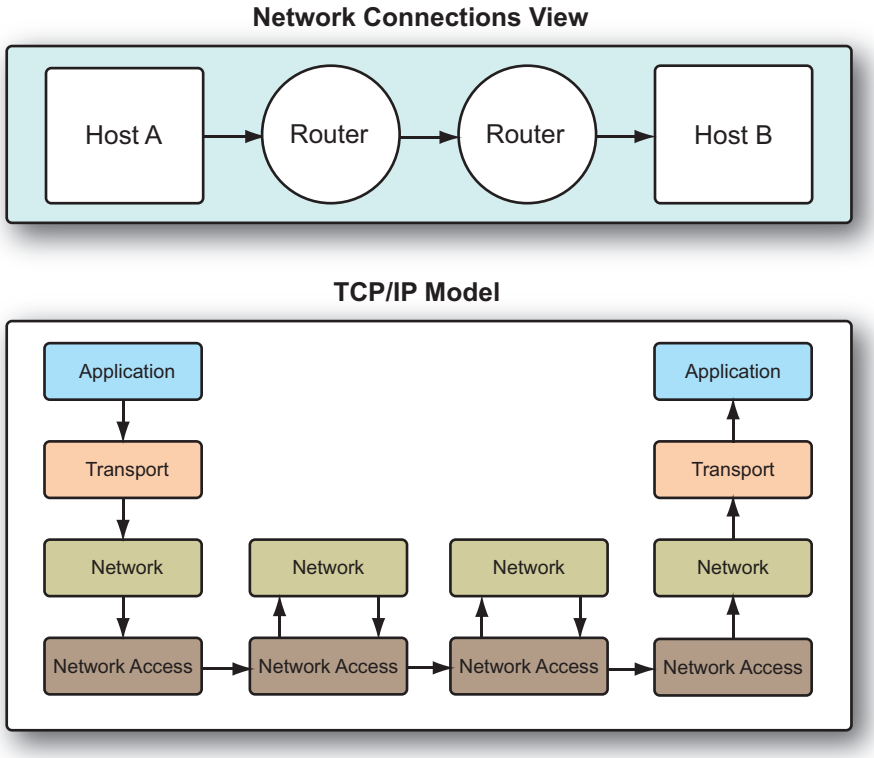


Figure 2.1: TCP/IP reference model. The upper diagram shows the network connection view of a communication path between two hosts. The lower diagram shows the same connection path in TCP/IP model. As shown in the figure, IP routers operate only up to network layer. They check the IP header of the IP packets to decide which would be the next gateway or router to use. Each gateway or router along the communication path is sometimes referred to as "hop".

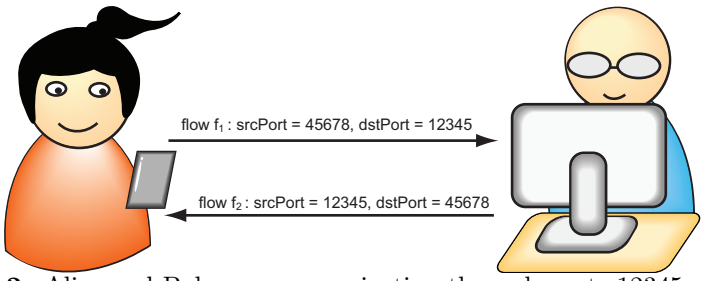


Figure 2.2: Alice and Bob are communicating through ports 12345 and 45678.

is said to "listen" to the port. Strictly speaking, a port is an internal address that identifies which application should receive the data packet that is sent to the host. A port cannot be used by more than one application at the same time. To this end, the packets can be sent to an application without specifying the application name explicitly. They can simply be sent to the port and the data will be directed to the application listening to it. Still, the sender has to be assigned to a port as well, so that to the receiver side can identify which application is the one sending the packet. If the receiver wishes to communicate back to the sender, it can send the packet back to the port that the sender is assigned to. That is, in each packet, there are two port numbers. One is the port of the application that sends the packet and the other is the port specifying the application that should receive the packet. We call the sending port the "source port" and the receiving port the "destination port".

Going back to our example, when Alice's videoconference client, called Video-A, wants to send video data to Bob's client, called Video-B, Video-A simply sends the data to the port that Video-B is waiting, say port 12345. Video-A, in turn, has to send the data through a port as well (say, port 45678). When Video-B wants to send the data back to Video-A, it can send the data back to port 45678. Here, the source and destination ports of the packets that are sent by Video-A to Video-B are 45678 and 12345, respectively. When the packets are sent back in the opposite direction, however, port 45678 would be the destination and port 12345 would be the source port. (See Figure 2.2.)

Furthermore, encapsulating the data with the header also provides abstraction of protocols and services. If an application sends the data to the TCP layer, for example, the TCP protocol attaches its header to the data and sends it further to the IP layer. The IP protocol has to be concerned only that it is a packet of TCP protocol regardless of which application generates the data. Figure 2.3 illustrates how the protocol header is added to the data.

There are two major transport protocols that are typically used: the Transmission Control Protocol (TCP) [Pos81b] and the User Datagram Protocol (UDP) [Pos80]. TCP is designed to guarantee both data correctness and completeness. If the data are not transferred correctly or some packets are missing along the path, they will be retransmitted. TCP detects the incorrect and missing data by the confirmations from the receiver. For every packet that is correctly received, the receiver has to send an acknowledgement back to the sender. TCP also provides congestion control such that the packet sending rate is adjusted with respect to the current network capability and

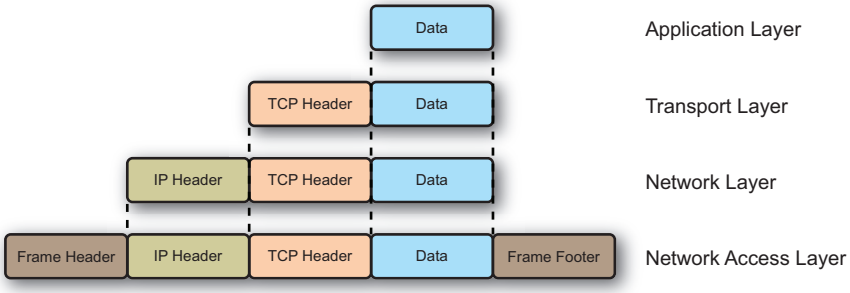


Figure 2.3: Packet encapsulation. TCP/IP architecture encapsulates the data from the upper layer by attaching a "header" of the current-layer protocol into the data.

congestion.

On the other hand, UDP is a simpler transport protocol that guarantees only data correctness but not completeness. It ensures that the data are transferred without error but it does not guarantee that every data packet is transferred. This allows UDP to transmit the data faster than TCP because there would be no overhead in data retransmission in case of packet loss or error. Thus, services that use UDP are normally the ones that are sensible to transmission delay but tolerable to erroneous and missing data [Tan03, page 43].

It is important to note that, in a single host, different applications could use different transport protocols. (Some applications can also use more than one protocol.) Each transport protocol has its own port numbering. That is, port 12345 of the TCP protocol is not the same as port 12345 of the UDP protocol.

2.2.4 Network Layer (IP Layer)

While the data correctness and completeness are taken care of by the transport protocols, the protocols in the network layer are responsible for the data delivery. The dominant packet delivery protocol over the Internet is the Internet Protocol (IP). Currently, the Internet Protocol that is generally in use is IP version 4 or IPv4 [Pos81a].

Once the data from the transport layer is sent to IP layer, the IP header is added to it similar to the transport layer. The most important information in the IP header are the "IP addresses". Like the transport ports, in a packet, two IP addresses have to be specified — namely, the "source IP address" (or the "source address") and the "destination IP address" (or the "destination address"). The former identifies which host has sent the packet, whereas the latter identifies where the packet should be sent to.

Combining with the information in the transport protocol header, one can distinguish which data packets belong to which applications on which host. For each packet, the source and destination IP addresses specify the sending and receiving hosts; the transport protocol, the source and the destination ports specify which application is the sender and which should be the receiver of the packet. In computer networks literature, the transport protocol, the source and destination addresses, as well as the source and destination ports, are usually referred to collectively as the "5-tuple". A sequence

of packets that is sent by a host to another host such that all packets in the sequence has the same 5-tuple value is called a "flow". In a sense, a flow is an application-to-application connection in which the source and destination of the connection are specified by the components in the 5-tuple. From here on, we will call the headers of the transport and network-layer protocols together as the "packet header". and the data from the application layer will be called the "packet payload". The number of packets that belong to a flow will be called the "flow length".

IP delivers the packets by checking the IP address in the packet header. If the destination host is connected directly to the source host (i.e., the local network), the packet will be delivered directly. Otherwise, it will be sent outside of the local network through a "gateway" or "router"[2]. If there is more than one gateway, the most appropriate one is selected through a process called "routing", which is done for each individual packet in the network. Different packets heading to the same destination could be routed through different gateways.

2.2.5 Network Access Layer

Sometimes referred to as the "link layer", the network access layer is the lowest layer in the model and also the closest to the physical network. It contains protocols which a network device needs to transmit the data through a physical medium. The protocols in this layer are designed with respect to the specific physical media to which they are intended to access. Each physical medium could have their own data units. For example, in the Ethernet network [80205], a data unit is called a "frame" whereas in Asynchronous Transfer Mode (ATM) network [I.199, GH99], it is called a "cell".

After the data has been transferred to the destination host, the protocol headers are stripped off level-by-level. Once the network access layer receives the data frame from the physical link, it takes off the frame header and footer and then forwards the data to the IP layer. The IP layer, in turn, obtains the packet, strips off the header and sends the packet to the transport layer and so on. At the end, the user application receives and reconstructs the data from the incoming packets.

2.3 Quality of Service

Traditionally, the routers in the IP-network route the packets based solely on the destination of the packets. Every data packet is intended to be transferred as fast as possible. The routers do not concern which applications send the packets or how *urgent* the packets are — for instance, the videoconference packets would be treated exactly the same as web-browsing packets. If the incoming packets are more than what it can handle, they will be either buffered or discarded. This is called the "best-effort" scheme.

[2]Both gateway and router can decide which path the packets should be routed to. Typically, a gateway has an extra feature such that it is normally used to bridge two networks together. The two networks can be of the same or of different types. That is, a gateway translates two different protocols. However, current routers are also equipped with the translation facility. Thus the terms gateway and router can be used interchangeably.

The TCP/IP protocol suite was introduced in the 1970s. At the time, major services were only email, FTP and Telnet [PR83] (WWW has been developed much later [FGM+]). These applications are not sensitive to network delay. When the network load is high, they can afford to wait until their buffered (or discarded) data are transmitted (or retransmitted). The best-effort scheme is therefore sufficient. However, recent applications, such as video or audio-conferences, multimedia streaming and even online games, are sensitive to transmission delay and variations in delays (or some other parameters). If the packets are not delivered in time, the applications may not be able to perform their tasks or services properly. The performance of the task is measured by its "quality-of-service (QoS)". To deliver the task with proper QoS, a mechanism that can provide an appropriate level of network performance to the applications is required. Such mechanism is called "QoS-support" and a network that is equipped with QoS-support is called a "QoS-support network".

2.3.1 Services

Loosely speaking, a service is a set of functions that work together to deliver a given task, such as sending a message to another user, retrieving an information from a website, or making a call to another user. The corresponding service to those tasks are: email, web-browsing, file transferring or videoconferencing. Due to open and flexible environment of the IP networks, it is difficult to define a "service" precisely [Räi03]. Some even go so far as to believe that standardizing services is impossible [EN01]. Still, the TeleManagement Forum [Tel04] has provided a definition of a service that, which we will use in the following:

Definition 2.1 (Service) A *service* is a set of independent functions that are an integral part of one or more business processes. This functional set consists of the hardware and software components as well as the underlying communication medium.

In [Tel04], the term "business process" refers to actual business operations such as trading, data transactions, or communication among individuals. A service is an operation or a task that a user is performing and could be carried out by different applications on different network architectures. Thus, a service can be thought as a functionality of a system, to which the user is interacting. How a service is carried out is not irrelevant. In turn, in the Unified Modeling Language (UML), a service can be modeled as a use case and the user is the actor interacting to the use case. Take the videoconference session between Alice and Bob again as an example. The videoconferencing is a service. It consists of the videoconference clients on Alice's smartphone and Bob's computer. Their hardwares (i.e. the smartphone and the computer) are connected together through a network architecture. A service is thus not just an application, network device, or the network individually, but a combined functionality. Consequently, the quality of the videoconference (e.g., the video or audio quality, the resolution, or the frame rate), does not rely only on any single component, but the collective performance of every component. These components include, for instance, processing power of Alice's smartphone and Bob's computer, their software clients and their network's capability.

2.3.2 Quality of Service

Extensive research has been done in the area of quality-of-service and there is a wide range of definitions for different contexts and purposes (e.g., [X.995], [E.894], [Moo01], [SW03]). We use the the following definition of the International Telecommunication Union's Telecommunication (ITU-T) [G.101a]:

Definition 2.2 (Quality-of-Service) The *quality of service* is the collective effect of service performances, which determine the degree of satisfaction of a user of service.

According to the definition, quality of service is the end result of every underlying process that affects the end user's experience of service. Nevertheless, service quality does not depend solely on these values, as it is also affected by psychological factors [Räi03]. Human tends to forget and forgive a glitch or quality dip in a communication over time if it does not occur too much and too often [EST02, Cla01] and does not coincide with an important piece of information [BSD00]. This means smooth and constant service quality are not always necessary although one should aim to achieve the acceptable quality level.

2.3.3 Quality of Service Management Models

As mentioned earlier, TCP/IP (especially IPv4) does not provide native QoS support. It needs to be deployed additionally. There are currently two major quality of service models proposed for TCP/IP: "Integrated Services (IntServ)" model and "Differentiated Services (DiffServ)" model. The two models, which are designed based on different approaches, are both standardized (in [BCS94] and [BBC+98]). The difference between them is that the network devices on IntServ model can negotiate with other devices in the network to reserve the network resources along the path from the connection sources to the destinations, while differentiated services treat each packet differently according to its "service class" without any explicit negotiation.

Integrated Services (IntServ)

In the Integrated Services (IntServ) architecture [BCS94], a service can negotiate and explicitly request network resources for each flow. A flow QoS requirement is specified by the flow descriptor, which consists of "filter specification (filterspec)" and "flow specification (flowspec)". The filterspec is used to identify the packets that belong to the flow, while the flowspec provides the flow QoS requirements.

The protocol designed to negotiate network resources in IntServ is the Resource reSerVation Protocol (RSVP)[ZBHJ97]. In an IntServ network, when an application tries to specify service quality to a flow, the RSVP messages are sent throughout the network to negotiate required resources. The routers that have appropriate resources would, in turn, accept the request, reserve the resources and inform the source about the reservation. This process is rather complex and requires high overhead [Räi03]. Moreover, reserving resources and keeping track of each reservation in large-scale networks would require high storage spaces and computational power from the routers. Thus, IntServ model is not scalable in large networks [Kil99, Räi03, LGW04].

Differentiated Services (DiffServ)

Due to the scalability problem of IntServ, Differentiated Services (DiffServ) has been proposed by the Internet Engineering Task Force (IETF) [BBC+98]. DiffServ is designed right from the beginning to be simple and scalable. The idea is to mark each packet with a "service class label" so that the routers can give them an appropriate treatment accordingly. Packets of classes with higher priorities will be treated with more importance. This would resolve the scalability problem in two ways. First, the routers do not have to maintain per-flow service quality. Second, the complex resource reservation process is not necessary, as the packets are classified into a service class only once at the network edge.

Since the routers treat each packet solely on the basis of its class label, explicit network resources are not required and no resource reservation is needed at run-time. Therefore, service quality has to be negotiated before any connection is established. This is done through the "service level agreement (SLA)", a service contract between a user and a network operator specifying service performance and forwarding treatment that the customer would receive. It also includes a "traffic conditioning agreement (TCA)", which specifies traffic profiling, shaping and classification that are to apply to the flows. In essence, SLA defines the QoS characteristics of each class. (Details on the SLA and TCA can be found in [BBC+98, Gro02].) An SLA can be either static or dynamic. A static SLA is an agreement that is made on a long-term basis (e.g., monthly or yearly). A dynamic SLA is made more frequently, even each time a connection is established.

A service class in the DiffServ model is labelled in the IP packet at the "Type-of-Service (TOS)" field in the IPv4 header or "Traffic-Class" field in IPv6 header. This label is called "differentiated service code point (DSCP)", or just "code point" [NBBB98]. The TOS field and the traffic class field together are called "DS field" in DiffServ terminology. After a packet is classified and labelled with a code point, it will be sent out to the network as an ordinary IP packet (i.e., through a non-deterministic path and no resource reservation). However, unlike typical IP packets, the routers along the paths will forward each packet differently based on its code point (hence the name differentiated services). This forwarding treatment is called "per-hop behavior (PHB)".

Per-hop behaviors lift the processing burden of the routers within the DiffServ network. They forward the packets according to the service class label specified by the DSCP. The traffic controlling is done at the edge node. Although this scheme can overcome the scalability problem of the IntServ architecture, it cannot really guarantee the uniform level of service quality along the route because each hop would handle packets based on its capability and current traffic condition. In any case, thanks to its scalability and simplicity, recent network standards, such as IPv6, 802.11e Wireless LAN, WiMAX or UMTS, implement DiffServ-based flow-prioritization schemes in their specifications.

2.3.4 QoS-Aware and QoS-Unaware Applications

So far we have discussed the TCP/IP architecture and its QoS management models (e.g., IntServ and DiffServ). These models assume that applications accessing the networks are aware of QoS-support and that they are supposed to specify the class of the service to their flows by themselves. These applications are called "QoS-aware applications". The flows that are specified by a service class is called "QoS-aware flows". There is also a large body of research that focuses on QoS specification languages that allow users and applications to express their needs (e.g., [Cam96, Flo96, LBS+98, AV98, WNGX01, NWG02]). An extensive review on QoS specification languages can be found in [JN02] and [JN04]. Nevertheless, most of the existing applications are not QoS-aware. Instead, the flows are treated in the best-effort fashion and cannot fully benefit from the QoS-support provided by the networks. These applications are called "QoS-unaware applications". The flows from those applications are called "QoS-unaware flows".

2.4 Providing QoS-Support to QoS-Unaware Applications

To provide QoS support to QoS-unaware flows, a number of mechanisms are proposed. They are aimed to incorporate some means of QoS-specification in the QoS-unaware flows so that they can be treated as QoS-aware flows. The added QoS-specification might not have to be a service class because some approaches are not aimed for DiffServ network.

A proxy-based system introduced by Tsetsekas et al. [TMV01] identifies the flows' QoS requirements at proxy servers according to their application-layer protocols. For a flow with an unknown protocol, a preliminary resource reservation is given, after which its resource usage will be periodically measured to provide a more accurate service quality for the flow. The knowledge of service quality requirements, however, is specific to that particular flow. Dharmalingam and Collier [DC02] have introduced a so-called manager node that resides in the network. It can interact and receive the QoS-support request for each application as well as an appropriate "application type" directly from the user. In [MHS02], a QoS request can be set up for QoS-unaware applications without modification to the source code by employing the QoS Library Redirection. This library redirection is carried out by rewriting the communication library in the operating system. When QoS-unaware applications open connections through the library, appropriate service qualities are assigned to the flow allowing QoS support to be added to the library instead of modifying the applications. A similar approach has also been implemented for the Microsoft Windows system. Roscoe and Bowen [RB00] facilitate transparent QoS support for Windows NT by modifying Winsock protocol stack. The modified stack captures the flow-opening calls from the applications. A policy daemon will consequently assign traffic policies to the flows according to the applications or the hosts that applications are connecting to. Both approaches, however, are merely designed as a mechanism to provide predefined QoS to QoS-unaware applications; they do not have the facility to automatically recognize applications' service types.

Shih et al. [SLSH04] pointed out that modifying operating system itself is rather complicated and proposed a transparent QoS mechanism that can transparently provide QoS to QoS-unaware applications without modifying either applications or the operating system. QoS support for QoS-unaware flows is provided by a QoS manager, which acts as a middleware between the kernel and the applications. Through the QoS manager, the user can specify which flows require quality-of-service support. Service quality level can then be assigned to the flows based on observed traffic characteristics.

While the aforementioned works have been developed from different approaches, none can automatically identify the certain QoS requirement or service class of each flow without human assistance or predefined QoS assignment rules. They are merely mechanisms aimed to assign QoS requirements to the flows and do not have the ability to decide what is the appropriate requirement for each flow. This is problematic when these systems have to deal with unknown applications or when the users lack sufficient technical knowledge. Therefore, a system that can identify the types of flows and assign appropriate QoS requirements to them without human-intervention would certainly be welcome. Such a system is called "flow classification system (FCS)". The next section presents an overview of the several FCSs.

2.5 Flow Classification System Components and Survey

Flow classification is a method to categorize a given flow to an appropriate group or class. A flow classification system is an implementation of such method. It originated in the area of network security where attacks or unauthorized flows must be detected [PN97][ZP00][ML05][RG07]. The network administrators also use the FCSs to distinguish different types of flows for network management and provision [SSW04] [MZ05b][BTS06][RG07][L7-08]. Recently, along with the need of QoS management, the flow classification has been employed to identify service classes of the QoS-unaware flows [RSSD04][ZNA05b][WZA06][AS07b][AS07a][TAO07].

2.5.1 Overview

Flow classification consists of three main processes: Flow observation, feature extraction and classification. The flow classification begins by observing the flows. Then the feature extraction process transforms flows, which are sequences of packets, into abstract representations called "features". In essence, features are characteristics of a flow that can be used to distinguish different types of flows. The employed features vary among different approaches, ranging from transport port numbers, payload content, to other characteristics such as throughput and average packet sizes. The feature extraction is done by a component called a "feature extractor" or simply "extractor". Multiple features of a single flow are usually presented together as a vector called "feature vector". In the classification process, a "classifier" analyzes these features and assigns the classes to the flows accordingly. Figure 1.1 illustrates the classification process.

Nevertheless, in the domains where the characteristics or the behaviors of the traffic are not well understood, designing an effective classifier might be difficult, hence

requiring adaptive flow classification systems that are able to induce and/or update their classifiers. We call such systems "adaptive flow classification systems". These are equipped with "learners", which allow them to induce new or update classifiers by analyzing relationships between the seen flows and their classes. The learning process is shown in Figure 1.2. In some domains, such as QoS-support, the flow classification has to be carried out in a limited period of time so that the flow can benefit from the assigned class. Such a system is called a "real-time flow classification system". Thorough discussion and precise definitions of such systems are provided in Chapter 3.

2.5.2 Features

The simplest approach to distinguish different kinds of the flows is to look at their transport ports. Traditionally, well-known Internet application protocols have specific transport port numbers registered to themselves by the Internet Assigned Numbers Authority (IANA) [Int07]. These registered ports are standardized and the registered applications are supposed to use only these ports to transfer their data. Some of these ports are shown in Table 2.1. Consequently, the applications that the flows belong to can be identified simply by associating the ports to applications, that is, the only concerned features are transport ports. We call an FCS that classifies flows using only transport port a "port-based FCS". Simple firewall applications or classical classification techniques such as [MKK+01, LC03, PN97] are examples of such systems. Advantage of port-based approach is that only one packet per flow is required to obtain the transportation ports, making the classification faster and simpler compared to other methods. However, applications might not communicate via registered ports or some application protocols are not registered to IANA. Madhukar and Williamson [MW06] have shown that unregistered traffic has increased from 10%-30% in 2003 to 30%-70% in 2005, causing the use of only the port numbers as features to be unreliable [RSSD04, KBFC04, KPF05, MP05]. Another approach used to identify flow classes is called "signature-based" approach. Signature-based FCSs work under the assumption that each class (usually defined over application-layer protocols) has specific payload contents, such as protocol commands and syntaxes. A specific content that can be used to identify a class is called a "signature" of the class. The payloads of the packets in the flows are extracted and searched for the signatures. In effect, the feature used by signature-based approach is the packet payload itself. This approach is analogous to how anti-virus software detects virus codes that reside in the files and is sometimes referred to as "deep packet inspection". One of the earliest works in signature-based classification was carried out by Zhang and Paxson in 2000 [ZP00]. They introduced a set of small algorithms designed to detect a number of application protocols, where each algorithm attempts to find specific signatures of a designated application protocol. In 2004, Sen et al. of AT&T Labs [SSW04] have proposed a method similar to Zhang and Paxson's FCS that focuses mainly on identification of peer-to-peer (P2P) flows. Moore and Papagiannaki [MP05] combined port-based and signature-based approaches by using both transport ports and signatures as their features. Their FCS consists of multiple classifiers, each of which classifies flows using only a specific number of

Application	Port Number
FTP Data Channel	20
FTP Control Channel	21
SSH	22
Telnet	23
SMTP	25
HTTP	80
POP3	110
IRC	194
HTTPS	443
Doom	666

Table 2.1: Example of port numbers and their corresponding protocols or applications assigned IANA in [Int07]. The assigned ports which are commonly known for specifics services are called "well-known ports". IANA maintains the port assignments only for ports 0-1023. Nevertheless, IANA also suggests in [Int07], ports above 1023 that are also well-known (although not formally assigned) such as port 5900 for Virtual Network Computing (VNC) [RSFWH98] or port 5004 for real-time transport protocol (RTP) [SC03].

packets. In the classification process, the classifier requiring the smallest number of packets attempts to classify the flow first. If it fails, another classifier that requires more packets would take over the classification. In case all classifiers fail to pinpoint the class in any flow, a human expert will be informed to manually classify the flow. The signature-based approach is also implemented in many network firewalls and intrusion detection systems (IDSs) including Snort [RG07], CheckPoint [Che08], Cisco IPS 4200 [Cis08] and Linux L7-filter [L7-08].

Because each application protocol typically has its own unique signatures, classifying flows using the signatures usually yields very high classification accuracy. However, accuracy does not come without a cost as signature-based classifiers always require predefined signatures. The signature set must be identified beforehand and, to be able to handle new applications, must be updated constantly. In addition, searching for signatures in packet payloads might raise privacy issues and might not work if the data in the payload are encrypted. Haffner et al. [HSSW05] and Ma et al. [MLK06] attempt to automatically identify signatures of a number application protocols using information retrieval techniques. Although the experimental results of both methods are promising, they are still plagued by the privacy and payload-encryption problems. Furthermore, the FCSs using this method tend to suffer from long signature-searching time [SSW04]. Many researches have contributed to this issue (see, for example, [DKSL03], [NSS05], [APA+05], [RJM06], [RG07]).

The signature-based approach, while reliable, is not flexible as the set of signatures must be updated constantly, leading to the development of more general techniques that also consider other flow features. These techniques, collectively called "flow-behavior-based" approaches, try to capture common "characteristics" of the flows in the same classes. The idea was originated in 2000 by Zhang and Paxson [ZP00], who tried to distinguish the flows that contain keystrokes and the ones that do not. It was observed that the flows which deliver keystrokes such as SSH or FTP commands

have specific packet sizes and inter-arrival times (IATs) characteristics [PF95]. As a consequence, the classifier is designed to capture the ratios of specific packet sizes and IATs. Using predefined ratios of small packets and short IATs, two kind of flows, keystroke and non-keystroke, can be distinguished. This technique was extended in [ML05]. Relying also on the predefined ratios, her approach can identify up to 11 application protocols. A similar approach has also been proposed in [TAO07]. There, if the packet IAT of two consecutive packets is larger than a predefined threshold, a "headpoint" is said to occur. The flows can be classified as "real-time" or "bulk" based on specific patterns of the headpoint occurrences. After the features are extracted, they will be analyzed by a classifier so that the flow can be classified accordingly. All of aforementioned FCSs are equipped with a classifier that has been pre-programmed by an expert, which means that they are still not be able to classify flows with unseen characteristics.

To attack this problem, machine learning techniques have been introduced to provide adaptability to the classification systems. Using machine learning, the patterns hidden in a set of data can be *learned* automatically. The earliest work to employ machine learning in flow classification is introduced by Early et al. in 2003 [EBR03]. The authors observed that different services generate TCP packets differently. For instance, HTTP traffic contains fewer packets with PSH flag than Telnet traffic. To capture this behavior, for each TCP flag[3], ratio of packets with that flag is used as a feature. A machine learning algorithm, C5.0 [Rul07], is employed to find relationships between ratio of each TCP flag and different service classes. Nevertheless, using TCP flags as features limits the use of FCS only to TCP flows. This might not be practical for conversational services such as videoconferences, VoIP, or real-time gaming as they mainly use UDP as transport protocol.

This idea has been extended by Roughan et al. [RSSD04] who employ a "clustering algorithm" as the learner to analyze various statistics of the flow, such as standard deviation of packet sizes, average data volume, etc. and then classify the flows accordingly. The same set of features is used by Williams et al. [WZA06]. However, instead of clustering algorithms, several "supervised learners" are utilized and evaluated. A similar approach has also been implemented by Zander et al. in 2005 [ZNA05b, ZNA05a]. In 2005, Moore and Zuev [MZ05b] employed Naive Bayes learner to analyze over 200 flow features, most of which are statistics of TCP flags and control packets in the flows. Nevertheless, after their investigation, only 3 - 50 features are found to be more discriminative than others. These FCSs, although adaptive, are not suitable for QoS-support because the entire flows have to be observed before they can be classified and, thus, cannot be classified in real-time.

Bernaille et al. [BTA+06][BTS06] point out that one can classify flows without observing all packets in the flows and propose a new set of features, which require only few packets of a flow to compute. The ratio of packet sizes of the flows that are transferred in both directions are different for different application protocols. This phenomenon is more pronounced in the first few packets of the flows. To capture

[3]The TCP state flags are: URG - Urgent, ACK - Acknowledgment, PSH - Push, RST - Reset, SYN - Synchronize and FIN - Finish. Details on these flags can be found in networking literatures such as [Tan03].

this behavior, packet sizes of both directions of each communication[4] are compared with respect to their orders, that is, the packet size of the first incoming packet is compared with that of the first outgoing packet and the sizes of second packets in both directions are then compared and so on. Because the features do not require every packet in a flow, the classification can be done while the flow is still running. We call such FCS a "real-time FCS". Nevertheless, to classify a flow, its coflow is always required to compute the packet size ratios. This means that the FCS can be used only on TCP flows where coflows always exist. Moreover it might not be useful in the networks where outgoing and incoming packets are transferred through different paths. Another real-time FCS is proposed by Erman et al. [EMA+07]. The features employed in their FCS include, for example, total number of packets, data volume, total caller to callee bytes and packets and total callee to caller bytes and packets. The caller in this case is the one who initiates TCP handshake and the callee is the one who responds the initiation. Thus, it does not work with UDP flows. In their method, several classifiers are employed simultaneously which each of them using a specific number of packets. For instance, once the observed number of packets reaches 5, the classifier K_5 will be used. When the number of packets reaches 10, K_{10} will be used to classify the flow again.[5] After the flow is reclassified, the previous class will be replaced by the new class. The FCS proposed in [EMA+07] suffers the same disadvantage as [BTA+06] as a coflow is always required to compute features and only TCP flows can be classified. Nevertheless, although an approach to classify flows within specific length is proposed, the author did not specify how many packets should really be observed. Moreover, as we will see in Section 6.2.1, empirical experiment results reveal that using multi-classifiers is not necessary because flow characteristics will not fluctuate after a specific flow length. Adding more classifiers after that point would be redundant.

All flow classification systems mentioned earlier analyze and classify each flow individually. Other approaches such as [KPF05] and [XZB05] take a different route and focus on classifying the hosts instead. After the type of the host (e.g., a server, a client, or an attacker) is determined, all flows from that host will be classified as the same class. While this approach is rather useful in a network provision where an overview of the networks is required, it is ineffective for flow-level classification.

Several researches on flow classification systems have been explored. Port-based and signature-based classification systems are non-intelligent, that is, the classification system does not have the learner and the classifier cannot be induced by the system itself. Although they can classify flows in real-time, the classifier must be constantly updated. Most of the flow-behavior-based classification systems, on the other hand, are intelligent as they are generally equipped with learners. Nevertheless, most of these systems are intended for network administration or intrusion detection and thus do not require or support real-time classification (except for [BTA+06, BTS06] and [EMA+07]). They can classify only TCP flows and are not suitable to handle UDP

[4] In this thesis, a flow is defined unidirectionally. In the situation where communication is bidirectional, there exist two flows that are transferred in opposite direction. They are said to be "coflows" of each other. Detailed discussion regarding the coflows can be found in Chapter 3.

[5] In this thesis, we call an approach that employs several classifiers to classify flows at different contexts a "multi-classifiers approach".

Table 2.2: Categorization of Flow Classification Systems

	Non-Adaptive	Adaptive
Non-real-time	Flow-behavior-based [ZP00],[ML05], [DWF03],[TAO07] Host-behavior-based [KPF05],[XZB05]	Flow-behavior-based [EBR03],[MZ05b], [WZA06],[RSSD04], [ZNA05b, ZNA05a]
Real-time	Port-based [MKK+01],[LC03],[PN97] Signature-based [ZP00],[SSW04],[MP05], [RG07],[Che08],[Cis08]	Flow-behavior-based [BTA+06, BTS06] [NA06][EMA+07]

applications, such as videoconferences and online games. It is clear then that an intelligent flow classification system that is self-adaptive and able to classify flows regardless of their transport protocol would represent an improvement to the existing systems. In this thesis, such an approach is proposed and is explored in detail in the following chapters. Moreover, existing FCSs can be grouped based on their adaptability and their abilities to classify flows in real-time. Table 2.2 shows a comparison of the flow classification systems.

2.5.3 Service Classes

The differentiated service standard [BBC+98] focuses only on service differentiation with respect to the given code points. How the code points should be interpreted is not a concern; it does not define how many classes should be employed, which applications or services belong to the classes or what the QoS requirements of the classes are. As the class definition is open, different sets of service classes are used in different FCSs depending on their respective purposes. For example, Zhang and Paxson [ZP00] introduced an FCS to detect unauthorized flows that carry Secure Shell (SSH) [Ylo06] and File Transfer Protocol (FTP) [PR85] commands. The service classes employed by this system are "keystroke" and "non-keystroke". Another FCS, also proposed by Zhang and Paxson in [ZP00], is aimed for more general network management purposes and employs a set of application protocols, including Telnet [PR83], SSH, Rlogin [Kan91], FTP, Napster [Tys07] and Gnutella [KAD+04, Man07], as service classes. Application protocols are also employed as service classes in some other studies, such as [EBR03], [ML05], [ZNA05b], [WZA06] or [BTA+06]. An FCS proposed by Sen et al. [SSW04] is intended mainly to identify various peer-to-peer (P2P) flows. Thus, the applied service classes are solely the P2P protocols, namely, Gnutella, eDonkey [eDo07], DirectConnect [Dir07], Kazaa [Ber03] and Bittorrent [Coh03]. Table 2.3 lists the application-layer protocols considered by various researches.

Identifying the application protocols individually, while they are being used in network provision and management, does not suit our purpose of assisting QoS-support. This is because some protocols could be used to serve different kind of services with

Employed Protocol	[ZP00]	[ML05]	[TAO07]	[EBR03]	[RSSD04]	[MZ05b]	[ZNA05b]	[WZA06]	[SSW04]	[MP05]	[BTA$^+$06]	[EMA$^+$07]	[NA06]
SSH*	×	×				×				×		×	×
Telnet*	×	×		×	×	×	×	×		×			×
Rlogin	×	×				×				×			
DNS					×	×	×	×		×			×
POP3*		×	×	×		×				×	×	×	
IMAP*		×	×	×		×				×	×	×	
SMTP*		×	×	×		×	×	×		×	×	×	×
NNTP*			×								×	×	×
HTTP*		×	×	×		×	×			×	×	×	×
HTTPS*		×			×						×	×	×
FTP*	×	×	×	×	×		×	×			×	×	
SMB*			×									×	
VNC*												×	
IRC*													
AIM Chat*							×						
Yahoo Messenger*													
ICQ												×	
MSN Chat*		×					×					×	
MSN Video*												×	
Half-life*						×		×		×			
Wolfenstein													×
HTTP Video*													
HTTP Audio*													
RTSP*			×									×	
QuickTime*			×										
RealAudio*			×		×	×				×			
Windows Media Player*			×			×				×			
Gnutella	×					×			×	×		×	
Napster	×						×						
eDonkey									×		×	×	
Kazaa					×	×			×	×	×	×	
DirectConnect									×			×	
Bittorrent						×			×	×		×	

Table 2.3: Application-layer protocols considered by various researches on flow classification – the ones included in our research are marked with *. Most studies do not classify flows into explicit protocols. The protocols are categorized based on the purposes of the FCSs. For example, [MP05] and [MZ05b] groups the protocols based on the practical purposes of the protocols and classify flows into one of the groups, [NA06] distinguishes only "Wolfenstein" and "the rest", and [TAO07] categorizes the protocols only into real-time and non-real-time traffics. In our research, the protocols are categorized with respect to their QoS requirements. In addition, there are more protocols that are considered in [MP05], [MZ05b], and [EMA$^+$07]. They are not included here because their proportions are extremely small. Also, [MP05] and [MZ05b] were conducted by the same group of researchers. They share the same sets of data and service classes.

Service Categories	[MZ05b]	[MZ05b]	[KPF05]	[EMA+07]
Remote Access	×	×		
Database	×	×		×
Mail	×	×	×	×
WWW	×	×	×	×
P2P	×	×	×	×
Games	×	×	×	
Chat			×	
Streaming	×	×	×	
Services	×	×	×	
Bulk	×	×	×	
Attack	×	×		

Table 2.4: List of researches that deine service classes based on practical purposes of the considered applications.

different QoS requirements. For example, HTTP can be used to deliver web pages, transfer files, or even perform media streaming. Some researchers, define the classes based on the practical purposes of the services. For instance, Moore and Zuev [MZ05b] and Moore and Papagiannaki [MP05] categorized services into "Bulk data", "Database", "Mail", "WWW", etc. Karagiannis et al. [KPF05] categorized services into "WWW", "Peer-to-Peer", "FTP", "Mail", etc. (See Table 2.4 for more details.) Again, defining service classes this way cannot differentiate the QoS requirements of the services in different classes.

DiffServ standard is nevertheless intended primarily to provide QoS support. To this end, IETF carried out two standard-track PHB specifications for DiffServ, namely, "expedited forwarding PHB (EF PHB)" [JNP99, DCB+02] and "assured forwarding PHB (AF PHB)" [HBWW99]. EF PHB provides low-delay, low-jitter and low-lost *end-to-end* connections through the network. EF PHB traffic will be provided with very low delay and assured bandwidth. At each hop, the minimum outgoing rate is well-defined (i.e. fixed and will not fluctuate) and the arrival rate is conditioned so that it will not exceed the minimum outgoing rate. RFC 2598 [JNP99] suggests that the edge node supposed to control its traffic limit to be lower than the outgoing rate and so the packets will not be queued. To avoid queuing, EF PHB traffic would be given absolute priority over other traffic (i.e. it has to be served first) or it is served in designated queue — effectively providing guaranteed QoS. (This is possible only when there are not so many EF PHB traffics on the network).

In the AF PHB scheme, users are offered choices of service classes to their aggregate traffic. Each class defines QoS parameter-values that the users would receive. (The agreement of the service classes and the corresponding QoS values is called "profile" and normally done through the SLA). As long as the traffic from a user does not exceed the agreed value, the traffic is called "in-profile" and the user will receive the agreed QoS with high assurance. The user, however, can access the network beyond the agreed value (or "out-profile") but the exceeding traffic will be served with no guarantee (i.e., in best effort manner). Four service classes defining four traffic profiles have

been proposed. Flows can be classified into one of the four classes according to their requirements. Furthermore, within one class, traffic can also be further divided into three more "drop precedences" (low, medium and high). When a congestion occurs, the router would drop the packets with higher drop precedence first, allowing finer traffic classification. Both class and drop precedence are marked at the DSCP. The PHB schemes, however, are merely packet queueing and dropping protocols. They do not specify any concrete mapping between PHB categories and services or applications.

In 2001, the International Telecommunication Union (ITU) defines eight different classes in ITU-T Recommendation G.1010 [G.101b]. The classes, shown in Table 2.5, are defined with respect to end-to-end user perception, i.e., how the user should experience the services. The classes are defined as a universal specification and do not depend on any specific network architecture. It can be seen from Table 2.5 that the proposed classes are divided mainly on the basis of delay sensitivity, which, according to [G.101b], has direct impact on the user perception. Among the same delay requirements, the services are further categorized with respect to error tolerances. These service classes are also adapted by 3GPP as part of their UMTS standard in 2004 [3GP04]. In both [G.101b] and [3GP04], explicit values of network performance expectations of each service class are also suggested. These are, however, beyond the scope of our research and will not be discussed here. In 2006, a service class guideline was proposed as an informational[6] Request for Comments (RFC) [BCB06]. It is designed especially for DiffServ networks. In this guideline, 12 different classes are proposed: Two for network operation and administration services and 10 for user services. The classes are divided based on their tolerance to delay, delay variation, packet loss and also data rate requirement or "elasticity". A service is said to be "elastic" if it can always wait for the data to arrive [BCS94]. This kind of service is tolerant to delay and jitter and its data rate can be adjusted over the course of the communication. Otherwise, it is said to be "inelastic". In our context, elastic services are comparable to real-time services and inelastic are comparable to non-real-time services. A summary of the characteristics of the user service classes is shown in Table 2.6. Further details, such as example applications and recommended PHBs of each class, can be found in [BCB06]. Classes proposed in G.1010 and RFC 4594 are aimed at capturing the QoS requirements of the services, whereby the services are grouped based on the delay sensitivity. However, such fine-grained classes are not suitable for flow classification task because some applications that provide the same services might belong to different classes. For instance, Skype and MSN are applications that provide videoconference service, yet the characteristics of their connections are highly different.

In this thesis, the service classes are defined with respect to the delay sensitivity and interactivity of the applications in the classes. Five classes are identified and these include "Strict Conversational (StrConv)", "Relaxed Conversational (RlxConv)", "Streaming", "Interactive" and "Bulk". They are also intended to be in compliance with the classes proposed in RFC 4594, an industrial standard guideline. Thus, they are comparable to the classes shown in Table 2.6. Their characteristics as well as examples

[6]Informational RFC is not a standard. It provides rather information or recommendation only and it might be originated by either IETF itself or other individual.

Error tolerant	Conversational voice and video	Voice / video messaging	Streaming audio and video	Fax
Error intolerant	Command / control	Transactions	Messaging, downloads	Background
	Interactive (delay << 1 s)	Responsive (delay ~2 s)	Timely (delay ~10 s)	Non-critical (delay >> 10 s)

Table 2.5: Model for user-centric QoS categories proposed by ITU [G.101b]. It is also adapted by 3GPP as part of UMTS standardization [3GP04].

Service Class	Traffic Characteristics	Elasticity	Sensitive to			G.1010 Rating
			Delay	Jitter	Loss	
Signaling	Variable size packets, some what bursty short-lived flows	Inelastic	H	L	H	Responsive
Telephony	Fixed-size small packets, constant emission rate, inelastic and low-rate flows	Inelastic	VH	VH	VH	Interactive
Real-time Interactive	RTP/UDP streams, inelastic, mostly variable rate	Inelastic	VH	H	H	Interactive
Multimedia Conference	Variable size packets, constant transmit interval, rate adaptive, react to loss	Rate adaptive	VH	H	M-H	Interactive
Broadcast Video	Constant and variable rate, inelastic, non-bursty flows	Inelastic	M	H	VH	Responsive
Multimedia Streaming	Variable size packets, elastic with variable rate	Elastic	M	L	M-H	Timely
High-Throughput Data	Variable rate, bursty long-lived elastic flows	Elastic	L-M	L	H	Timely
Low-Latency Data	Variable rate, bursty short lived elastic flows	Elastic	M-H	L	H	Responsive
Low-Priority Data	Non-real-time and elastic	Elastic	L	L	L	Non-critical
Standard	Unspecified	Elastic	N/A	N/A	N/A	Non-critical

Table 2.6: Summary of service classes proposed in RFC 4594 [BCB06] and their characteristics. L, M, H, VH in the tolerance columns stand for low, medium, high and very high respectively. Rate adaptive elasticity means that the service can change its data rate, but only in fixed-step manner, resembling step-wise data rate.

of applications that belong to each class are described in Section 5.1. Apart from our research, some flow classification studies classify flows based on delay-sensitivities and interactivity as well. Roughan et al. [RSSD04] classifies flows into "Remote Access", "Streaming","Bulk" and "Transactional" (which includes HTTP and DNS) and [TAO07] classifies flows into real-time and non-real-time. Their classes, however, do not capture all types of services. For instance, [RSSD04] entirely ignore videoconference and online gaming while, Tai et al. categorizes both HTTP, FTP and Mail protocols together as non-real-time and considers only streaming protocols as real-time [TAO07].

2.6 Summary

In this chapter, we have discussed about the TCP/IP network as well as QoS management models that are designed to be used with it. We have also pointed out that DiffServ model is more scalable and supported by many network standards, including the upcoming IPv6, WiMAX and 802.11e. Thus, it is chosen as our target QoS model. Since the QoS-unaware applications may not benefit from the QoS support provided by the network, flow classification systems have been proposed to provide the QoS support to QoS-unaware applications. The flow classification task, however, requires more than assigning a proper class to a given flow. It has to be carried out in real-time. To handle unknown applications, the FCS has to be adaptive as well.

We have reviewed a large number of FCSs component by component and found that none of them can fulfill all requirements of the flow classification task listed in Chapter 1. Moreover, as it turns out, all existing FCSs are developed separately without any underlying model. Therefore, understanding and comparing them are rather cumbersome. In conclusion, not only a real-time adaptive FCS itself, but also a rigid model that describes the flow classification domain and process are still missing.

Chapter 3

A Unified Framework for Flow Classification

In the previous chapter, a review of existing flow classification systems was given. As discussed, different systems are developed independently without any general underlying model that can explain the common elements and processes, making analyzing, comparing, and understanding the relationships among different approaches rather difficult. This issue has been raised in [SOMS08] but the authors have only introduced a reliable measurement method, not a model that can identify the classification systems. In response, we propose a mathematical model that can describe the flow classification components and processes as well as existing classification approaches. The model provides a unified framework for the analysis and comparison of different flow classification approaches.

In the following sections, we model the components in a flow classification system. These include, e.g., the packets, flows, flow characteristics or features, and service classes.

3.1 Packets and Flows

When an application at a host sends its data to another host, the data are divided into chunks called packets. These packets are then traversed across the network to the destination host. Each packet consists of three main parts — the transportation and the network-layer protocol headers, the packet payload with higher-layer protocol headers and the application data. We will now discuss a mathematical model that describes the packets. Because we would like to have a model that can describe any IP packet regardless of which transport protocol is in use, the protocol-specific information such as TCP flags or window sizes is ignored.[1] The model contains only the general packet information including the transport protocol, source IP, source port, destination IP, destination port, packet timestamp, packet size, and payload. This information

[1]For details on this information, see, e.g., [Pos81a], [DH98], [Pos81b] and [Pos80] for IP version 4 (IPv4), IP version 6 (IPv6), TCP and UDP headers respectively

is transport-protocol independent and can always be obtained. We call such model a "packet model".

Here, we consider only TCP and UDP as they are the protocols commonly used to transfer users' data. The timestamp is the time when the packet is observed by the packet capturing component[2]. The packet size is the size of the packet specified in the IP header. Because the packets are intended to transfer data (i.e., the content in the payload), we start modeling our framework from the lowest element in the domain: The packet payload.

Computer data are sequences of bits or binary numbers (i.e., sequences of 0s and 1s). We thus represent the data as sequences of strings.

Definition 3.1 (String) Let Σ be a finite non-empty set of symbols. A *string* or *data string* is a finite sequence

$$a_1 \dots a_n$$

where $a_i \in \Sigma$ and $n \in \mathbb{N}$. We call n the length of the string. The string of length 0 is called *empty string* and denoted by ϵ. The set of *strings of length* n is denoted by Σ^n where $\Sigma^0 = \{\epsilon\}$. The *set of non-empty strings* is denoted by $\Sigma^+ = \bigcup_{m \geq 1} \Sigma^m$ and $\Sigma^* = \Sigma^+ \cup \Sigma^0$ is called a *set of strings*.

Although the computer data can be represented as bit-strings (i.e., $\Sigma = \{0, 1\}$), referring to the data directly as bit-strings might not be practical. Thus, in this thesis, the data are treated byte-wise instead of bit-wise and represented in sequences of bytes. We use 2-digit hexadecimal number to represent the value of each byte. For example, the byte 1010 0010 can be represented as A2.

The following is an example of a string that is taken from a payload in a packet. Spaces are added here to maintain the readability and to distinguish each byte, although, strictly speaking, they are not part of the strings.

```
CD AD 52 54 53 50 2F 31 2E 30 20 32 30 30 20 4F
4B 0D 0A 53 65 72 76 65 72 3A 20 51 54 53 53 2D
```

When QoS-aware applications establish flows, they also specify service classes to their flows. As discussed in Section 2.3.3, in the DiffServ model, the service class is marked in each packet. The routers along the route will then treat the packets accordingly. We define the service classes as the following set:

$$\mathcal{C} = \{c_1, \dots, c_k\}.$$

The elements of $\mathcal{C}$ are not required to be explicitly specified. This allows the system designer to apply any sets of classes according to the applications of the system. The set of classes employed in an FCS defines the purpose of the FCS because it specifies what would the flows be classified into. For example, an FCS presented in [ZP00]

[2]The packet capturing component could reside anywhere in the network, including the source and destination hosts and the intermediate routers. Its location depends on the purpose of the capturing.

is aimed for network security. It classifies flow into *attack* and *normal*. Thus the following service class is employed:

$$\mathcal{C}' = \{attack, normal\}$$

An FCS intended for network management as presented in [WZA06] classifies flows according to application protocols. The employed set of classes is then:

$$\mathcal{C}'' = \{http, ftp, smtp, telnet, dns, half\text{-}life\}$$

In our case, the FCS facilitates QoS support. The set of classes is therefore defined with respect to QoS requirements of the flows. As discussed in Section 2.5.3, the following service classes are employed: Strict conversational class (StrConv), relaxed conversational class (RlxConv), interactive class, streaming class and background class. That is:

$$\mathcal{C} = \{StrConv,\ RlxConv,\ Interactive,\ Streaming,\ Background\}. \tag{3.1}$$

Characteristics and examples of applications that belong to each class are described in Section 5.1. Unlike QoS-aware applications, QoS-unaware applications do not assign any class to their packets. In this case, the service class is said to be "unspecified" or "empty" and it is denoted by ε.

Definition 3.2 (Packet Model) Consider the following sets:

$\mathbb{N}$	natural numbers
$\mathbb{I}$	set of IP-addresses
$\mathbb{P}$	set of transport protocols $= \{TCP,\ UDP\}$
$\mathcal{C}^{\varepsilon}$	$\mathcal{C} \cup \{\varepsilon\}$
Σ^*	set of strings over $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, \mathrm{A}, \mathrm{B}, \mathrm{C}, \mathrm{D}, \mathrm{E}, \mathrm{F}\}$

Elements of $\mathbb{P} \times \mathbb{I} \times \mathbb{N} \times \mathbb{I} \times \mathbb{N} \times \mathbb{N} \times \mathbb{N} \times \mathcal{C}^{\varepsilon} \times \Sigma^*$ are called *packet models*. A *set of packet models* is denoted by $P \subseteq \mathbb{P} \times \mathbb{I} \times \mathbb{N} \times \mathbb{I} \times \mathbb{N} \times \mathbb{N} \times \mathbb{N} \times \mathcal{C}^{\varepsilon} \times \Sigma^*$. Let $p = (w_1, \ldots, w_9) \in P$, we define the following functions:

$$\begin{array}{lll}
protocol: & P \to \mathbb{P}: & p \mapsto w_1 \\
srcIP: & P \to \mathbb{I}: & p \mapsto w_2 \\
srcPort: & P \to \mathbb{N}: & p \mapsto w_3 \\
dstIP: & P \to \mathbb{I}: & p \mapsto w_4 \\
dstPort: & P \to \mathbb{N}: & p \mapsto w_5 \\
timestamp: & P \to \mathbb{N}: & p \mapsto w_6 \\
size: & P \to \mathbb{N}: & p \mapsto w_7 \\
class: & P \to \mathcal{C}^{\varepsilon}: & p \mapsto w_8 \\
payload: & P \to \Sigma^*: & p \mapsto w_9
\end{array}$$

A connection between two services in two different hosts is called a flow. Different connections can be distinguished from each other via the 5-tuple values, i.e., source IP, destination IP, source port, destination port, and the transport protocol. Hence, a

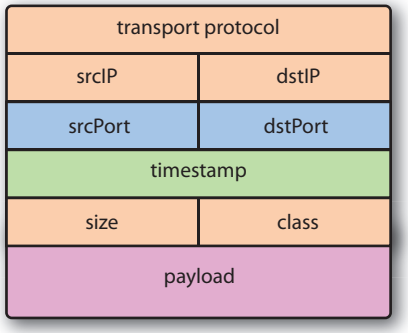


Figure 3.1: The packet model. The orange part is the information abstracted from the IP header [Pos81a]. The blue part is abstracted from the transport protocol header (see [Pos81b] and [Pos80] for TCP and UDP headers respectively), the pink part is the packet payload, and the green part is obtained from the packet capturing component. The actual protocol headers contain more information. However, they are not relevant to our discussion and thus discarded.

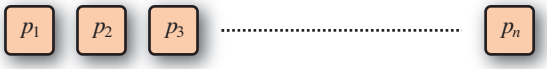


Figure 3.2: A flow is a sequence of packets $p_1, \ldots, p_n$ such that they have the same transport protocol, IP addresses and transport ports.

flow can be modeled as a sequence of packets such that all packets in the flow contain the same 5-tuple values (see Figure 3.2). At any rate, because the service class is given to each packet in the flow individually, it is possible that different packets in the same flow could have different classes. In this discussion, we assume that all packets in a flow are marked as the same class.

Definition 3.3 (Flow) Let P be a set of packet models, $\mathbb{N}^+ = \{1, 2, 3, \ldots\}$, and $n \in \mathbb{N}^+$. A *flow* f is a non-empty finite sequence

$$(p_i \mid 1 \leq i \leq n)$$

where

$$\begin{array}{rlcl} (\forall 1 \leq i \leq n) & p_i \in P, & & \\ (\exists r \in \mathbb{P})(\forall 1 \leq i \leq n) & \mathit{protocol}(p_i) & = & r, \\ (\exists s \in \mathbb{I})(\forall 1 \leq i \leq n) & \mathit{srcIP}(p_i) & = & s, \\ (\exists u \in \mathbb{N})(\forall 1 \leq i \leq n) & \mathit{srcPort}(p_i) & = & u, \\ (\exists t \in \mathbb{I})(\forall 1 \leq i \leq n) & \mathit{dstIP}(p_i) & = & t, \\ (\exists v \in \mathbb{N})(\forall 1 \leq i \leq n) & \mathit{dstPort}(p_i) & = & v. \end{array}$$

n is said to be the *length of* f. A *set of flows* is denoted by $\mathcal{F}$. In addition, let

Table 3.1: The table shows the information of each packet model within a flow. The columns indicate the components of each packet. For the sake of simplicity, the timestamp is shown in the milliseconds that the packets are captured, starting from zero. In reality, the timestamp is stored in UNIX time format [RR95].

Packets	Protocol	srcIP	srcPort	dstIP	dstPort	timestamp	size	class
p_1	UDP	9.119.124.200	27960	24.195.14.189	27960	000000	68	RlxConv
p_2	UDP	9.119.124.200	27960	24.195.14.189	27960	023658	66	RlxConv
p_3	UDP	9.119.124.200	27960	24.195.14.189	27960	047861	65	RlxConv
p_4	UDP	9.119.124.200	27960	24.195.14.189	27960	071661	65	RlxConv
p_5	UDP	9.119.124.200	27960	24.195.14.189	27960	093118	66	RlxConv
p_6	UDP	9.119.124.200	27960	24.195.14.189	27960	116573	65	RlxConv
p_7	UDP	9.119.124.200	27960	24.195.14.189	27960	140576	68	RlxConv
p_8	UDP	9.119.124.200	27960	24.195.14.189	27960	161649	68	RlxConv
p_9	UDP	9.119.124.200	27960	24.195.14.189	27960	185548	68	RlxConv
p_{10}	UDP	9.119.124.200	27960	24.195.14.189	27960	230565	68	RlxConv
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
p_{707}	UDP	9.119.124.200	27960	24.195.14.189	27960	170924	68	RlxConv

$(p_i \mid 1 \leq i \leq n) \in \mathcal{F}$. We extend the functions introduced in Definition 3.2 as follows:

$$
\begin{array}{lll}
\mathit{protocol}: & \mathcal{F} \to \mathbb{P}: & (p_i \mid 1 \leq i \leq n) \mapsto \mathit{protocol}(p_1) \\
\mathit{srcIP}: & \mathcal{F} \to \mathbb{I}: & (p_i \mid 1 \leq i \leq n) \mapsto \mathit{srcIP}(p_1) \\
\mathit{srcPort}: & \mathcal{F} \to \mathbb{N}: & (p_i \mid 1 \leq i \leq n) \mapsto \mathit{srcPort}(p_1) \\
\mathit{dstIP}: & \mathcal{F} \to \mathbb{I}: & (p_i \mid 1 \leq i \leq n) \mapsto \mathit{dstIP}(p_1) \\
\mathit{dstPort}: & \mathcal{F} \to \mathbb{N}: & (p_i \mid 1 \leq i \leq n) \mapsto \mathit{dstPort}(p_1) \\
\mathit{class}: & \mathcal{F} \to \mathcal{C}^{\varepsilon}: & (p_i \mid 1 \leq i \leq n) \mapsto \mathit{class}(p_1)
\end{array}
$$

We call $f \in \mathit{flow}$ such that the class is not empty (i.e., $\mathit{class}(f) = \varepsilon$) a *QoS-unaware flow* or *QoS-aware flow* otherwise.

According to the definition, a flow of zero packet (i.e., an empty sequence) is not allowed because, in reality, zero packet means there is no packet sent. Thus, the flow does not exist. Table 3.1 shows a sequence of packets that constitute a flow. The sequence containing the first l packets of a flow is called the l-prefix of the flow, which is characterized as follows.

Definition 3.4 (Prefix of a Flow) Let f be a flow of length n, i.e., $f = (p_i \mid 1 \leq i \leq n)$ and $l \in \mathbb{N}^+$. The *prefix* of f of length l is a flow defined as follows:

$$
\begin{array}{lll}
\mathit{prefix}: & \mathcal{F} \times \mathbb{N}^+ \to \mathcal{F} & \\
& (f, l) \mapsto \begin{cases} f & \text{if } l \geq n, \\ (p_i \mid 1 \leq i \leq l) & \text{otherwise.} \end{cases}
\end{array}
$$

We call $\mathit{prefix}(f, l)$ the *l-prefix of* f.

To exemplify the prefix of a flow, the 3-prefix of the flow shown in Table 3.1 refers to (p_1, p_2, p_3). Figure 3.3 provides further illustration of the flow prefix. Because the DiffServ architecture defines the flow as unidirectional, in our framework, flows are

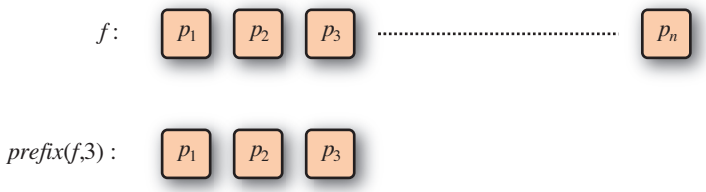


Figure 3.3: Prefix of a flow of length three.

modeled as unidirectional as well. A two-way communication between two hosts (e.g., a TCP socket) thus consists of two flows - one for each direction. Figure 2.2 illustrates such scenario. In the figure, the flows f_1 and f_2 have the corresponding pairs of ports. Source port of f_1 is the destination port of f_2 and vice versa. They are said to be transferred in "opposite direction" of each other. We call f_1 and f_2 "coflows" of each other.

Definition 3.5 (Coflow) Consider a set $\mathcal{F}$ of flows. *Coflow* is a binary relation defined as follows:

$$Coflow \subseteq \mathcal{F} \times \mathcal{F}$$

such that $(f, f') \in Coflow$ iff

$$\begin{aligned} protocol(f') &= protocol(f), \\ srcIP(f') &= dstIP(f), \\ srcPort(f') &= dstPort(f), \\ dstIP(f') &= srcIP(f), \\ dstPort(f') &= srcPort(f). \end{aligned}$$

If $(f, f') \in Coflow$, then $f' \in \mathcal{F}$ is said to be a *coflow of* $f \in \mathcal{F}$.

Definition 3.5 does not restrict the number of coflows. Still, a flow generally has only one coflow, i.e., the flow that communicates in response to it. When a flow f is established from a host A to another host B, host B can establish only one coflow f' back to A, in response to f. In this case, f and f' are said to belong to the same "conversation".

Definition 3.6 (Managed Set of Flows) Let $\mathcal{F}$ be a set of flows. $\mathcal{F}$ is said to be *managed* iff for each $f \in \mathcal{F}$ there exists at most one $f' \in \mathcal{F}$ such that f' is a coflow of f. Given a managed set of flows $\mathcal{F}$, if there exists $f, f' \in \mathcal{F}$ such that f' is a coflow of f, then f' is said to be the coflow of f.

One should observe that given a managed set of flows $\mathcal{F}$, $f \in \mathcal{F}$ is a coflow of $f' \in \mathcal{F}$ iff $f' \in \mathcal{F}$ is a coflow of f. In the following, only the managed set of flows is considered and, hence, the set of flows are presumed to be managed unless stated otherwise.

3.2 Features

So far, the packet, flow, and service classes, which are the fundamental elements in data transmission, have been defined. In general, however, a flow can be characterized as a feature vector, where each component of the vector indicates a feature of the flow. As briefly discussed in Section 2.5.1, features are the abstract representations that describe particular characteristics of a flow. Examples of features include transport protocol, port number, payload content, throughput, number of packets, or connection time. They are usually presented together as a vector called a "feature vector". Each feature is extracted from a flow by a "feature function", which is defined as follows.

Definition 3.7 (Features) Let $\mathcal{F}$ be a set of flows, and D be a set. A *feature function* is a function

$$\begin{aligned} V: \quad & \mathcal{F} \to D \\ & f \mapsto d. \end{aligned}$$

We call d a *feature of* f. A *set of feature functions* is denoted by $\mathcal{V}$.

It is worth pointing out that, in flow classification literature, the classification methods are usually evaluated without mentioning the feature extraction process as the data used in the experiments are already extracted as feature vectors. However, the extraction process in fact plays a crucial role in the overall classification performance especially in real-time systems where the classification has to be done in a limited time period. This is because different features require different amount of time and number of packets to compute. Thus, to be able to specify the maximum number of packets used in the extraction process, we extend the feature function as follows.

Definition 3.8 (Length-Restricted Feature Function) Let $V : \mathcal{F} \to D$ be a feature function and $f \in \mathcal{F}$ a flow. A *length-restricted feature function* V' *corresponding to* V of length $l \in \mathbb{N}^+$ is defined as follows:

$$\begin{aligned} V': \quad & \mathcal{F} \times \mathbb{N}^+ \to D \\ & (f, l) \mapsto V(\mathit{prefix}(f, l)) \end{aligned}$$

A *set of length-restricted feature functions* is denoted by $\mathcal{V}'$.

In a sense, a length-restricted feature function is an encapsulated function or an interface to a feature function that restricts the feature function to considering only up to prefix of length l. The actual feature calculation is still done by the feature function.

Definition 3.9 (Feature Vector) Let $V_i : \mathcal{F} \to D_i, 1 \leq i \leq d, d \in \mathbb{N}^+$, be feature functions. Elements of $D_1 \times \ldots \times D_d$ are called *feature vectors*. A set $\mathcal{X}_d = D_1 \times \ldots \times D_d$ is called a *set of feature vectors*.

Intuitively, a feature vector is a vector of features where each component in the vector is computed by a feature function. Collectively, they provide an abstraction of

a flow. The set $\mathcal{X}_d$ is the set of all feature vectors with the same number of components d. The subscript d will be dropped if the number of the components is irrelevant to the context. A feature vector, as mentioned earlier, is an abstracted representation of a flow. Each component in the vector is computed by a designated feature function. Still, each feature function can calculate only one component. The transformation of a flow to a feature vector that consists of multiple components (i.e., multiple features) is carried out by the following function.

Definition 3.10 (Feature Extraction Function) Let $\mathcal{V} = \{V_1, \ldots, V_d\}$ be a set of feature functions such that $V_i : \mathcal{F} \to D_i, 1 \leq i \leq d$, $d \in \mathbb{N}^+$, and $\mathcal{X}_d = D_1 \times \ldots \times D_d$. A *feature extraction function corresponding to* $\mathcal{V}$ is

$$\begin{aligned} E_{\mathcal{V}} : \quad & \mathcal{F} \to \mathcal{X}_d \\ & f \mapsto \langle V_1(f), \ldots, V_d(f) \rangle. \end{aligned}$$

We call $E_{\mathcal{V}}(f)$ an *abstraction* of f or the *feature vector of* f.

This means that a feature extraction function $E_{\mathcal{V}}$ corresponding to $\mathcal{V}$ is the function that maps a flow to a vector using all features in $\mathcal{V}$. Naturally, the dimension of the resulting vector is equal to the cardinality of $\mathcal{V}$.

At this point, a feature vector is only an element of $D_1 \times \ldots \times D_d$. The features in a feature vector can be calculated using an arbitrary flow length, which could also be the entire flow. Although calculating features using the entire flow might reflect the actual characteristics of the flow, it is very time consuming because all the packets in the flow must first be observed. This might not be suitable for real-time classification and, therefore, the feature extraction function is extended so that it considers only up to a given flow length l.

Definition 3.11 (Length-Restricted Feature Extraction Function) Let $\mathcal{V}' = \{V'_1, \ldots, V'_d\}$ be a set of length-restricted feature functions such that $V'_i : \mathcal{F} \times \mathbb{N}^+ \to D_i, 1 \leq i \leq d, d \in \mathbb{N}^+$, and $\mathcal{X}_d = D_1 \times \ldots \times D_d$. An *l-restricted feature extraction function corresponding to* $\mathcal{V}'$ is defined as follows:

$$\begin{aligned} E'_{\mathcal{V}'} : \quad & \mathcal{F} \times \mathbb{N}^+ \to \mathcal{X}_d \\ & (f, l) \mapsto \langle V'_1(f, l), \ldots, V'_d(f, l) \rangle. \end{aligned}$$

We call $E'_{\mathcal{V}'}(f, l)$ the *l-feature vector of* f.

In other words, the feature extraction function maps the flows into a feature vector using designated length-restricted features. Note that all the features are computed using up to the same flow length (i.e., at most l). For the sake of simplicity, if it is clear from the context which set of features is used, we can also refer to the components in a feature vector and a length-restricted feature vector not just by their values, but also by the feature functions corresponding to them. For instance, consider a feature-vector

$$\langle UDP, 27960, 27960, 46828, 17.09, 2740.08 \rangle, \tag{3.2}$$

where the components are computed by the following functions: *protocol*, *srcPort*, *dstPort*, *dataVolume*, *connTime* and *dataTPUTAvg*, respectively. The vector shown in (3.2) could be written as:

$$\langle protocol = UDP, srcPort = 27960, dstPort = 27960, dataVolume = 46828, \\ connTime = 17.09, dataTPUTAvg = 2740.08\rangle.$$

In addition, if the explicit value of each component is not required, each component can be denoted by the name of the corresponding feature function, e.g.,

$$\langle protocol, srcPort, dstPort, dataVolume, connTime, dataTPUTAvg\rangle.$$

After a QoS-unaware flow is transformed to a feature vector, it can be analyzed and then an appropriate service class can be assigned. This class assignment process is called "flow classification". A QoS-unaware flow or a feature vector assigned to a class is said to be "classified". In the classification process, the classifier does not analyze the flow directly, but rather the feature-vector of the flow. Therefore, a classifier can be modeled as a function that maps a feature vector to a class.

Definition 3.12 (Classifier) Let $\mathcal{V} = \{V_1, \ldots, V_d\}$ be a set of feature functions such that $V_i : \mathcal{F} \to D_i, 1 \leq i \leq d, d \in \mathbb{N}^+$, $\mathcal{X}_d = D_1 \times \ldots \times D_d$, and $\mathcal{C}$ a set of service classes. A *classifier corresponding to* $\mathcal{X}_d$ is a function

$$K_{\mathcal{X}_d} : \mathcal{X}_d \to \mathcal{C}.$$

A set of all classifiers corresponding to $\mathcal{X}_d$ is denoted by $\mathcal{K}_{\mathcal{X}_d}$.

In other words, the classifier classifies flows with respect to their feature values. This then implies that the classifier has to be compatible with the set of feature vectors $\mathcal{X}_d$. Nevertheless, if it is clear from the context, the subscript d will be dropped. As discussed in Section 2.5, there are numerous flow classification systems in the literature, which can be distinguished primarily by their classifiers or, to be precise, the domains and codomains of the employed classifiers. The domain of the classifier designates the features that are concerned by the FCS. The codomain of the classifier designates the service class of the FCS, which, in turn, defines the purpose and the use of the FCS.

The fundamental components of the flow classification system introduced so far are summarized in Figure 3.4. (The last component, the learner, will be discussed in Section 3.3.) Finally, we define a general flow classification system as follows.

Definition 3.13 A *generic flow classification system* is a tuple

$$\langle P, \mathcal{F}, \mathcal{V}, \mathcal{X}_d, E_{\mathcal{V}}, \mathcal{C}, K_{\mathcal{X}_d}\rangle$$

where

- P is a set of packet models,
- $\mathcal{F}$ is a managed set of flows,

- P - a set of packet models
- $\mathcal{F}$ - a set of flows
- $\mathit{prefix} : \mathcal{F} \times \mathbb{N}^+ \to \mathcal{F}$ - the flow prefix function
- $\mathcal{V} = \{V_1, \ldots, V_d\}$ - a set of feature functions such that $V_i : \mathcal{F} \to D_i \in \mathcal{V}$, $1 \leq i \leq d, d \in \mathbb{N}^+$
- $\mathcal{V}' = \{V'_1, \ldots, V'_d\}$ - a set of length-restricted feature functions such that $V'_i : \mathcal{F} \times \mathbb{N}^+ \to D_i : (f, l) \mapsto V_i(\mathit{prefix}(f, l))$, $1 \leq i \leq d$
- $\mathcal{X}_d = D_1 \times \ldots \times D_d$ - a set of feature vectors
- $E_{\mathcal{V}} : \mathcal{F} \to \mathcal{X}_d$ - a feature extraction function
- $E'_{\mathcal{V}'} : \mathcal{F} \times \mathbb{N}^+ \to \mathcal{X}_d$ - a length-restricted feature extraction function
- $\mathcal{C}$ - a set of service classes
- $\mathcal{K}_{\mathcal{X}_d}$ - a set of classifiers
- $K_{\mathcal{X}_d} : \mathcal{X}_d \to \mathcal{C} \in \mathcal{K}_{\mathcal{X}_d}$ - a classifier used by the system
- $L : \mathfrak{D} \to \mathcal{K}_{\mathcal{X}_d}$ - a learner

Figure 3.4: Fundamental components of flow classification.

- $\mathcal{V} = \{V_1, \ldots, V_d\}$ is a set of feature functions such that $V_i : \mathcal{F} \to D_i \in \mathcal{V}$, $1 \leq i \leq d$, $d \in \mathbb{N}^+$,
- $\mathcal{X}_d = D_1 \times \ldots \times D_d$ is a set of feature vectors,
- $E_{\mathcal{V}} : \mathcal{F} \to \mathcal{X}_d$ is a feature extraction function,
- $\mathcal{C}$ is a set of service classes and,
- $K_{\mathcal{X}_d} : \mathcal{X}_d \to \mathcal{C}$ is a classifier.

Generic flow classification systems employ ordinary feature extraction function that uses all packets in the flow. The features are thus calculated after the flow is finished. In real-time flow classification, the number of packets used to calculate the feature values is restricted. Therefore, real-time flow classification system is then defined as:

Definition 3.14 A *real-time flow classification system* is a tuple

$$\langle P, \mathcal{F}, \mathit{prefix}, \mathcal{V}, \mathcal{V}', \mathcal{X}_d, E'_{\mathcal{V}'}, \mathcal{C}, K_{\mathcal{X}_d} \rangle$$

where

- $P, \mathcal{F}, \mathcal{V}, \mathcal{X}_d, \mathcal{C}, K_{\mathcal{X}_d}$ are the components defined in Definition 3.13,
- $\mathit{prefix} : \mathcal{F} \times \mathbb{N}^+ \to \mathcal{F}$ is the flow prefix function,

- $\mathcal{V}' = \{V'_1, \ldots, V'_d\}$ is a set of length-restricted feature functions such that $(\forall i \in \{1, \ldots, d\}) V'_i : \mathcal{F} \times \mathbb{N}^+ \to D_i : (f, l) \mapsto V_i(\textit{prefix}(f, l))$ and
- $E'_{\mathcal{V}'} : \mathcal{F} \times \mathbb{N}^+ \to \mathcal{X}_d$ is a length-restricted feature extraction function,

That is, the real-time flow classification system extends the generic one by employing two extra components *prefix* and $\mathcal{V}'$. It also uses length-restricted feature extractor, $E'_{\mathcal{V}'}$, instead of ordinary one ($E_{\mathcal{V}}$). As a result, the number of packets used to calculate the feature values can be specified. Note that, in both systems, the classifier $K_{\mathcal{X}_d}$ is predefined.

Modeling the flow classification system this way restricts the way elements interact among each other to an extent that each function has specific domain and codomain. The other elements such as packets and flows are also precisely defined. As a result, the classification process, i.e. mapping a flow to a class, is also precise and well-defined. The whole processes of general and real-time classification can be expressed simply as $K_{\mathcal{X}_d}(E_{\mathcal{V}}(f))$ and $K_{\mathcal{X}_d}(E'_{\mathcal{V}'}(f, l))$ respectively.

3.3 Adaptive Flow Classification System

In the machine learning literature, there exists a class of methods called "supervised learning". Given a set of pairs, these methods attempt to estimate structural patterns in the set [SD90][WF05]. In our setting, the pairs are those of feature vectors and their service classes, and, for a given set of pairs, a supervised learner, or simply a "learner", is to estimate a function that maps a feature vector to an appropriate service class with respect to the observed pairs. In other words, the learner induces a classifier from the given set of pairs of feature vectors and service classes.

In flow classification scenario, the feature vectors utilized by the learner have to be computed from the flows where each of these flows is marked with a service class. Therefore, the pairs of feature vectors and classes have to be computed from the pairs of flows of their classes. We call a set of flows-classes pairs that will be used in the learning process a "raw dataset". A set of pairs of feature vectors and service classes is called a "dataset".

Definition 3.15 (Raw Dataset) Let $\mathcal{F}$ be a set of flows and $\mathcal{C}$ be a set of service classes. Elements of $\mathcal{F} \times \mathcal{C}$ are called *flow instances* or *flow examples*. We call a finite set $\mathcal{R} \subseteq \mathcal{F} \times \mathcal{C}$ a *raw dataset* or a *set of raw data*.

In other words, a raw dataset is a set of (unique) flows that have a service class assigned. Keep in mind that, for the QoS-aware flows, the assigned classes are labeled in each packet in the flows (see Definition 3.2 and 3.3). The class assigned for each flow in the raw dataset might not be the same as the class specified in each packet of the flows. This allows us to associate any classes to the flow instances, regardless of what their original classes are.

In the following, we will formally define the dataset, or a set of pairs of feature vectors and classes. We will begin with an introduction to multiset, which is the collection of objects that forms the basis for a dataset. Then, the formalization of the dataset will be given, followed by the functions that map a raw dataset to a dataset.

Conceptually, a multiset is a collection of elements whereby certain elements can occur more than once. The number of times an element occurs in a multiset is called the *multiplicity* (i.e., number of occurrences) of that element. Formally, it is defined as a function that maps an element in a set to a natural number indicating the multiplicity of the element.

Definition 3.16 (Multiset) Let X be a set of elements, a *multiset* M *over* X is a function

$$\begin{aligned} M: \quad & X \to \mathbb{N} \\ & x \mapsto k. \end{aligned}$$

We call k the *multiplicity* x *in* M. For all $x \in X$, if $M(x) > 0$, x is said to be an *element* of M. If there exist no $x \in X$ such that $M(x) > 0$, then M is said to be *empty*, denoted by $M = \emptyset$. The *set of underlying elements* of M is the set $U_M = \{x \in X \mid M(x) \neq 0\}$. M is *finite* if U_M is finite.

To explicitly enumerate all elements of M, we use the *dot* notation

$$M = \dot{\{}x_{1_1}, \ldots, x_{1_{k_1}}, x_{2_1}, \ldots, x_{2_{k_2}}, \ldots, x_{n_1}, \ldots, x_{n_{k_n}} \dot{\}}$$

where $x_i \in X$ and k_i is the multiplicity of x_i, $i \in \{1, \ldots, n\}$, $n \in \mathbb{N}^+$. Elements of a multiset is not ordered. Therefore, any permutation of the notation can be used. For example, given a multiset containing four elements, a, a, b, and c, $\dot{\{}a, a, b, c\dot{\}} = \dot{\{}a, b, a, c\dot{\}}$.

Definition 3.17 (Cardinality of a Multiset) Let X be a set of elements, the *cardinality* of a multiset M over X, denoted by $\mathit{Card}(M)$, is given by

$$\mathit{Card}(M) = \sum_{x \in X} M(x).$$

We use the notation $|M|$ to denote $\mathit{Card}(M)$. Further operators and functions of multiset are given in, e.g., [Bli89],[GJ09],[JGT01], and [SIYS07].

Definition 3.18 (Dataset) Let $\mathcal{X}$ be a set of feature vectors and $\mathcal{C}$ be a set of service classes. A *dataset* is a finite multiset over $\mathcal{X} \times \mathcal{C}$. Elements of a dataset are called *data instances* or *examples*. The set of all datasets is denoted by $\mathfrak{D}$.

In other words, a dataset is a finite multiset $\dot{\{}\langle \boldsymbol{x}_1, c_1 \rangle, \ldots, \langle \boldsymbol{x}_m, c_m \rangle\dot{\}}$ such that $\langle \boldsymbol{x}_i, c_i \rangle \in \mathcal{X} \times \mathcal{C}$, $1 \leq i \leq m$. Although a collection of pairs of feature vectors and classes is generally called a "dataset", the term is somewhat a misnomer. A dataset, unlike a raw dataset, is a multiset, not a set. This is because elements in a dataset could be identical [Koh95]. Depending on the feature functions employed, feature extraction function might not be one-to-one. In other words, given a feature extraction $E_{\mathcal{V}}$ corresponding to a set of features $\mathcal{V}$, there exist flows $f, f' \in \mathcal{F}$ such that $E_{\mathcal{V}}(f) = E_{\mathcal{V}}(f')$. This is true for length-restricted feature extraction function as well. The reason behind this is that each flow in raw dataset can be distinguished by the 5-tuple.

However, if the 5-tuple is not considered when the flows are extracted to feature vectors, then the flows might lose their uniqueness. Therefore, to ensure that abstractions of all flows in the raw dataset exist after the feature extraction, a multiset is required. The multiplicities of feature vectors are necessary for learners that induce classifiers based on statistics of flow features such as C4.5 decision tree algorithm [Qui93], Naive Bayes, Partial Rule (PART) [FW98], and many others [WF05].

Now, we have two types of data collections, namely, raw dataset and dataset. To transform a raw dataset into a dataset, all flows in the raw dataset have to be extracted to feature vectors and stored into a dataset. The transformation is done by a function called "dataset generator" defined below.

Definition 3.19 (Dataset Generator) Let $\mathcal{F}$ be a set of flows, $\mathcal{C}$ a set of service classes, and $\mathfrak{D}$ the set of all datasets. Given a set of feature vector $\mathcal{V}$ and a feature extraction function $E_{\mathcal{V}} : \mathcal{F} \to \mathcal{X}_d$ with $d = |\mathcal{V}|$, a *dataset generator corresponding to* $\mathcal{V}$ is a function defined as follows:

$$\begin{aligned} G: \quad & \mathcal{P}(\mathcal{F} \times \mathcal{C}) \to \mathfrak{D} \\ & \{\langle f_1, c_1\rangle, \ldots, \langle f_m, c_m\rangle\} \mapsto \dot{\{}\langle E_{\mathcal{V}}(f_1), c_1\rangle, \ldots, \langle E_{\mathcal{V}}(f_m), c_m\rangle\dot{\}}. \end{aligned}$$

Definition 3.20 (Length-Restricted Dataset Generator) Let $\mathcal{F}$ be a set of flows, $\mathcal{C}$ a set of service classes, and $\mathfrak{D}$ the set of all datasets. Given a set of length-restricted feature vector $\mathcal{V}'$, an l-restricted feature extraction function $E'_{\mathcal{V}'} : \mathcal{F} \times \mathbb{N}^+ \to \mathcal{X}_d$ with $d = |\mathcal{V}'|$ and $l \in \mathbb{N}^+$, an *l-restricted dataset generator corresponding to* $\mathcal{V}'$ is a function defined as follows:

$$\begin{aligned} G': \quad & \mathcal{P}(\mathcal{F} \times \mathcal{C}) \times \mathbb{N}^+ \to \mathfrak{D} \\ & (\{\langle f_1, c_1\rangle, \ldots, \langle f_m, c_m\rangle\}, l) \mapsto \dot{\{}\langle E'_{\mathcal{V}'}(f_1, l), c_1\rangle, \ldots, \langle E'_{\mathcal{V}'}(f_m, l), c_m\rangle\dot{\}}. \end{aligned}$$

We call $\dot{\{}\langle E'_{\mathcal{V}'}(f_1, l), c_1\rangle, \ldots, \langle E'_{\mathcal{V}'}(f_m, l), c_m\rangle\dot{\}}$ a *dataset of length l* or *l-dataset*.

The dataset generator abstracts all flows from the given set of raw data to a dataset. This is done without concerning the flow length. The l-restricted feature-vector generator, in contrast, abstracts the feature vectors with respect to l. The resulting dataset, $\dot{\{}\langle E'_{\mathcal{V}'}(f_1, l), c_1\rangle, \ldots, \langle E'_{\mathcal{V}'}(f_m, l), c_m\rangle\dot{\}}$, thus contains only the l-feature vectors.

Because a learner induces a classifier from a dataset, it can be modeled as a function that maps a dataset to its classifier.

Definition 3.21 (Learner) Let $\mathfrak{D}$ be the set of all datasets and $\mathcal{K}$ be a set of classifiers, a *learner* is a function defined as follows:

$$\begin{aligned} L: \quad & \mathfrak{D} \to \mathcal{K} \\ & \dot{\{}\langle \boldsymbol{x}_1, c_1\rangle, \ldots, \langle \boldsymbol{x}_m, c_m\rangle\dot{\}} \mapsto K. \end{aligned}$$

The definition basically states that the learner induces a classifier from a given set of data. At this point we are not concerned with any learners in particular; the learner and the classifier are thought simply as functions. With a learner, we can extend our generic flow classification system as follows:

Definition 3.22 A *adaptive flow classification system* is a tuple

$$\langle P, \mathcal{F}, \mathcal{V}, \mathcal{X}_d, E, \mathcal{C}, \mathcal{K}_{\mathcal{X}_d}, L \rangle$$

where

- $P, \mathcal{F}, \mathcal{V}, \mathcal{X}_d, E, \mathcal{C}$ are the components defined in Definition 3.13,
- $\mathcal{K}_{\mathcal{X}_d}$ is a set of classifiers and
- $L : \mathfrak{D} \to \mathcal{K}_{\mathcal{X}_d}$ is a learner.

The adaptive flow classification system uses a learner to induce a classifier effectively replacing a predefined classifier making the classification system adaptive. Likewise, a learner can be employed in real-time flow classification system as well. By replacing the predefined classifier with a learner, a real-time adaptive flow classification system can be defined as:

Definition 3.23 A *real-time adaptive flow classification system* is a tuple

$$\langle P, \mathcal{F}, \mathit{prefix}, \mathcal{V}, \mathcal{V}', \mathcal{X}_d, E'_{\mathcal{V}'}, \mathcal{C}, \mathcal{K}_{\mathcal{X}_d}, L \rangle$$

where

- $P, \mathcal{F}, \mathit{prefix}, \mathcal{V}, \mathcal{V}', \mathcal{X}_d, E'_{\mathcal{V}'}, \mathcal{C}$ are the components defined in Definition 3.14,
- $\mathcal{K}_{\mathcal{X}_d}$ is a set of classifiers and
- $L : \mathfrak{D} \to \mathcal{K}_{\mathcal{X}_d}$ a learner.

Note that the dataset and the classifier are not included in the model. This is because what distinguishes different classification systems is the learner, not the dataset used to induce the classifier or the induced classifier. Also, an adaptive flow classifier employs the feature extraction function that is not length-restricted, and thus the entire flow must also be observed in the classification process. A real-time adaptive FCS, in contrast, can restrict the length of the flow considered as it is equipped with a length-restricted feature extraction function. Consequently, the induced classifier can classify the flow at any preferred flow length. This is true in both dataset generation and classification processes.

3.4 Unsupported Methodologies

The proposed model is designed to be as general as possible. It is intended to cover all flow classification components and processes, regardless of which transport protocols are used. However, it still has some limitations and there are some flow classification approaches that cannot be described by our model.

3.4.1 TCP-Specific Features

As discussed in Section 2.2, the TCP protocol provides mechanisms to guarantee data correctness and completeness. In doing so, the TCP protocol adds some information in the TCP header of each packet. Such information is used in some flow classification researches that are discussed below. Nevertheless, our model of the packet does not contain protocol specific information such as TCP flags or window size. Thus, classification approaches that utilize TCP-specific information are not supported by the current framework.

In 2003, Early et al. [EBR03] introduced a classification system that classifies flows using TCP state flags. A flag is a binary variable describing a property of a packet. Early et al. observed that different services generate TCP packets differently. For instance, HTTP traffic contains fewer packets with PSH flag than Telnet traffic. To capture this behavior, for each TCP flag, Early et al. propose a feature that calculate the ratio of the number of packets with that flag set against the total number of packets. In effect, there are in total six features, one for each TCP flag.

Moore and Zuev [MZ05b, MZ05a] proposed 248 features to be used in the flow classification scheme. Most of them are related to TCP-specific properties, such as number of control packets, window sizes, round-trip time (RTT) between packets sent and their acknowledgements. Other features are calculated from packet inter-arrival time, Fourier transform of inter-arrival time, transport port, etc. The full list of the features is available in [MZ05a].

Nonetheless, although our current model cannot describe TCP-specific features, it can be extended without much effort by adding TCP header fields into the packet model.

3.4.2 Host-Behavior-Based Features

Another breed of flow classification technique, which takes a totally different approach than the works discussed earlier, is called host-behavior-based classification. The host-behavior-based methods do not attempt to identify flow types with respect to individual flow's characteristics. Instead, they try to identify the services or applications running on the hosts based on the hosts' interactivity rather than directly classify the flow types. It works under the assumption that different types of hosts (e.g., a web server or an end-user client) exhibit different communication characteristics. For example, a host is likely to be a service provider (e.g., web server or files server) if it communicates with other hosts using only few source ports. Thus, the flows from that host are considered as interactive flows. Host-based behavior classification systems are not intended to classify individual flows. They are generally designated for network management.

In SIGCOMM 2005, Karagiannis et al. introduced a host-classification scheme called BLINd Classification (BLINC) [KPF05]. BLINC collects the transport protocol, source IP and port and destination IP and port (i.e., the 5-tuple) of each flow coming in and out of a host. That is, BLINC considers five features. To identify the type of a host, Karagiannis et al. introduced a host-classification rules set that is expressed in a visualized form called "graphlets". These rules are used to identify the host type

based on, e.g., the transport ports used by the host, the number of flows from or to the host, and the number of other hosts that are in contact with the host. BLINC is not equipped with a mechanism that can generate the classification rule set. The classification rules proposed by Karagiannis et al. are created by hand.

Another approach which is similar to BLINC is a traffic profiler introduced by Xu et al. The approach has also been introduced in SIGCOMM 2005 [XZB05]. The traffic profiler is a classification system intended to identify interesting (or rather strange) host behaviors compared to other hosts. It considers four features, namely, source IP, source port, destination IP, and destination port. In their approach, the flows are clustered based on the four features using a clustering algorithm. After the hosts are clustered, the common behavior within the cluster can be identified.

The aforementioned systems are not considered flow classification systems. This is because they classify hosts, not individual flows in the hosts. They are therefore not covered by our framework.

3.5 An Example of Flow Classification Scenario

This section presents a simple flow classification scenario. We will see how to transform a flow into a feature vector and how a feature vector be classified. An example of a small dataset will also be given.

Consider a set of feature functions

$$\mathcal{V} = \{protocol, srcPort, dstPort, dataVolume, connTime, dataTPUTAvg\} \tag{3.3}$$

where

$$\begin{aligned}
protocol: \quad & \mathcal{F} \to \mathbb{P} \\
& (p_i \mid 1 \le i \le n) \mapsto protocol(p_1), \\
srcPort: \quad & \mathcal{F} \to \mathbb{N} \\
& (p_i \mid 1 \le i \le n) \mapsto srcPort(p_1), \\
dstPort: \quad & \mathcal{F} \to \mathbb{N} \\
& (p_i \mid 1 \le i \le n) \mapsto dstPort(p_1), \\
dataVolume: \quad & \mathcal{F} \to \mathbb{N} \\
& (p_i \mid 1 \le i \le n) \mapsto \sum_{i=1}^{n} size(p_i), \\
connTime: \quad & \mathcal{F} \to \mathbb{R} \\
& (p_i \mid 1 \le i \le n) \mapsto \frac{(timestamp(p_n) - timestamp(p_1))}{1000}, \\
dataTPUTAvg: \quad & \mathcal{F} \to \mathbb{R} \\
& (p_i \mid 1 \le i \le n) \mapsto \frac{dataVolume((p_i \mid 1 \le i \le n))}{connTime((p_i \mid 1 \le i \le n))}.
\end{aligned}$$

The feature function *dataVolume* measures the total amount of data that are transferred. *connTime* determines the flow running time. The feature function *dataTPUTAvg* calculates the average data throughput of the flow (in bytes). The following

is the set of length-restricted feature functions corresponding to feature functions presented above:

$$\mathcal{V}' = \{protocol', srcPort', dstPort', dataVolume', connTime', dataTPUTAvg'\} \tag{3.4}$$

where

$$\begin{aligned}
protocol' : \quad & \mathcal{F} \times \mathbb{N}^+ \to \mathbb{P} && (3.5)\\
& (f, l) \mapsto protocol(prefix(f, l)), \\
srcPort' : \quad & \mathcal{F} \times \mathbb{N}^+ \to \mathbb{N} && (3.6)\\
& (f, l) \mapsto srcPort(prefix(f, l)), \\
dstPort' : \quad & \mathcal{F} \times \mathbb{N}^+ \to \mathbb{N} && (3.7)\\
& (f, l) \mapsto dstPort(prefix(f, l)), \\
dataVolume' : \quad & \mathcal{F} \times \mathbb{N}^+ \to \mathbb{N} && (3.8)\\
& (f, l) \mapsto dataVolume(prefix(f, l)), \\
connTime' : \quad & \mathcal{F} \times \mathbb{N}^+ \to \mathbb{R} && (3.9)\\
& (f, l) \mapsto connTime(prefix(f, l)), \\
dataTPUTAvg' : \quad & \mathcal{F} \times \mathbb{N}^+ \to \mathbb{R} && (3.10)\\
& (f, l) \mapsto dataTPUTAvg(prefix(f, l)).
\end{aligned}$$

A feature vector corresponding to the length-restricted feature functions defined in (3.5) to (3.10) is an element of the following cartesian product:

$$\mathcal{X}_6 = \mathbb{P} \times \mathbb{N} \times \mathbb{N} \times \mathbb{N} \times \mathbb{R} \times \mathbb{R} \tag{3.11}$$

Given the set feature functions $\mathcal{V}$ from (3.3) and a corresponding set of feature vectors $\mathcal{X}_6$ from (3.11), a feature extraction function $E_\mathcal{V}$ corresponding to $\mathcal{V}$ is characterized as follows:

$$\begin{aligned}
E_\mathcal{V} : \quad & \mathcal{F} \to \mathcal{X}_6 \\
& f \mapsto \langle protocol(f), srcPort(f), dstPort(f), dataVolume(f), \\
& \qquad connTime(f), dataTPUTAvg(f) \rangle.
\end{aligned} \tag{3.12}$$

Consider the flow shown in Table 3.1. It is a UDP flow whose source and destination ports are 27960. It has the length of 707 and 46828 bytes of its data have been transfered in 17.09 seconds. It has an average throughput of 2740.08 bytes per second (Bps). Applying the feature extraction function to the flow yields the following feature vector:

$$E_\mathcal{V}(f) = \langle UDP, 27960, 27960, 46828, 17.09, 2740.08 \rangle. \tag{3.13}$$

The first component in the vector, the protocol, indicates the transport protocol of the flow. The second and third components indicate the source and destination ports of the flows respectively. The fourth component, the total data volume, is the sum of

the sizes of packets one to five. The fifth component is the connection time, which is the timestamp difference between the packet p_1 and p_{707}. Finally, the last component is the average data throughput calculated by dividing the total data volume by the connection time.

Consider the set of length-restricted feature functions $\mathcal{V}'$ in (3.4). A feature extraction function $E'_{\mathcal{V}'}$ corresponding to $\mathcal{V}'$ is the function

$$\begin{aligned} E'_{\mathcal{V}'}: \quad & \mathcal{F} \times \mathbb{N}^+ \to \mathcal{X}_6 \\ & (f,l) \mapsto \langle protocol'(f,l), srcPort'(f,l), dstPort'(f,l), dataVolume'(f,l), \\ & \qquad connTime'(f,l), dataTPUTAvg'(f,l)\rangle. \end{aligned}$$

Applying $E'_{\mathcal{V}'}$ to the flow shown in Table 3.1 by specifying the prefix to five, we have:

$$E'_{\mathcal{V}'}(f,5) = \langle UDP, 27960, 27960, 330, 0.07, 3548.39\rangle. \tag{3.14}$$

Notice that the values of the third to the last element of the vectors in (3.13) and (3.14) are different. This is because, although they are abstractions of the same flow, they are calculated using different flow lengths.

Given the set of feature functions $\mathcal{V}$ from (3.3), set of feature vectors $\mathcal{X}_6$ from (3.11) and $\mathcal{C} = \{StrConv,\ RlxConv,\ Interactive,\ Streaming,\ Background\}$ a set of service classes, the following is an example of a simple port-based classifier:

$$\begin{aligned} K_{\mathcal{X}_6}: & \mathcal{X}_6 \to \mathcal{C} \\ & (\langle protocol, srcPort, dstPort, dataVolume, connTime, dataTPUTAvg\rangle) \\ & \mapsto \begin{cases} StrConv & \text{if } srcPort = 27960 \text{ or } dstPort = 27960, \\ RlxConv & \text{if } srcPort = 1680 \text{ or } dstPort = 1680, \\ Interactive & \text{if } srcPort = 80 \text{ or } dstPort = 80, \\ Streaming & \text{if } srcPort = 32052 \text{ or } dstPort = 32052, \\ Background & \text{otherwise.} \end{cases} \end{aligned} \tag{3.15}$$

The classifier above maps a feature vector to a class with respect to the transport protocol ports. Such classifiers are employed in some FCSs such as [MKK+01], [LC03], and [PN97]. Indeed, the actual port-based classifiers that are used in the real implementations would be more complicated and the classification conditions would be more elaborated. Also, the feature vector would consist of only two components, one for source port and another one for destination port.

Applying $K_{\mathcal{X}_6}$ to the feature vector shown in (3.14), we have

$$\begin{aligned} K_{\mathcal{X}_6}(E'_{\mathcal{V}'}(f,5)) &= K_{\mathcal{X}_6}(\langle UDP, 27960, 27960, 330, 0.07, 3548.39\rangle) \\ &= StrConv. \end{aligned}$$

If we pair up a feature vector and a service class together, a data instance can be constructed. Following is an example of a data instance corresponding to the feature vector in (3.14):

$$\langle\langle UDP, 27960, 27960, 330, 0.07, 3548.39\rangle, StrConv\rangle.$$

Table 3.2: A Small Dataset. Each column describes each characteristic or feature of a flow. The last column indicates the service class of the flow.

protocol	*srcPort*	*dstPort*	*connTime*	*dataVolume*	*avgDataTPUT*	Class
UDP	51662	28314	52.70	222	4.21256	*StrConv*
UDP	57938	63789	18.86	576	30.5401	*StrConv*
UDP	15187	46992	2.73	160	58.6478	*StrConv*
UDP	7777	7777	24.88	2700	108.522	*StrConv*
UDP	29423	46992	15.15	544	35.9253	*StrConv*
TCP	3644	5050	2.16	4369	2022.69	*RlxConv*
TCP	62923	5050	0.78	1253	1606.41	*RlxConv*
TCP	5190	33773	0.26	192	735.63	*RlxConv*
TCP	5050	1213	0.42	1706	4091.13	*RlxConv*
UDP	527	4224	4.224	44	10.42	*RlxConv*
TCP	4314	554	402.53	64356	159.88	*Streaming*
TCP	2049	554	2553.28	2	962.94	*Streaming*
TCP	3458	554	563.93	80844	143.36	*Streaming*
TCP	25626	554	0.26	216	826.13	*Streaming*
TCP	4462	554	264.43	46464	175.72	*Streaming*
TCP	50790	3124	6.70	216	32.22	*Interactive*
TCP	80	51177	1.12	456	408.38	*Interactive*
TCP	3124	2428	10.51	2604	247.76	*Interactive*
TCP	55312	80	1.30	3462	2663.77	*Interactive*
TCP	3124	1454	114.94	36351	316.25	*Interactive*
TCP	48244	25	0.37	372	1016.32	*Bulk*
TCP	25	39186	1.11	751	674.84	*Bulk*
TCP	25	56881	1.05	955	911.99	*Bulk*
TCP	25	4946	10.73	4272	397.99	*Bulk*
TCP	25	15792	4.31	2134	495.25	*Bulk*

The example here is the flow abstraction shown in (3.14), paired with a service class, *StrConv*. For the sake of simplicity, we shall write such a pair of feature vector and a service class as

$$\langle \mathit{UDP}, 27960, 27960, 330, 0.07, 3548.39, \mathit{StrConv} \rangle,$$

or, if the feature functions need to be explicitly specified,

$$\langle \mathit{protocol} = \mathit{UDP}, \mathit{srcPort} = 27960, \mathit{dstPort} = 27960, \mathit{dataVolume} = 46828, \\ \mathit{connTime} = 17.09, \mathit{dataTPUTAvg} = 2740.08, \mathit{class} = \mathit{StrConv} \rangle.$$

Table 3.2 shows an example of a dataset.

3.6 Decomposing SMART

In addition to mathematical model presented earlier, in this section, Unified Modeling Language (UML) diagrams describing SMART will be presented. We will begin with overview of SMART and subsystem decomposition. We will also see how SMART could

be deployed in network devices. Then, we will move on to package and class diagrams describing each component of the system. Finally, we will discuss how SMART switches between classification and learning operations.

3.6.1 Deploying SMART

SMART can be deployed in both user's end-device and the router. Figures 3.5 and 3.6 present deployment diagrams for deploying SMART in a user device and a router, respectively. Deploying SMART in a router allows SMART to learn and classify flows from different devices residing in the network. This is because SMART treats all flows similarly regardless from (or to) which devices. In both figures, the `Application` component consists of all applications running on an end-device. The `NetworkSubsystem` includes all network components such as protocol stacks and physical network interfaces. SMART acts as a support system, which assists the router in identifying the classes of incoming flows. To this end, SMART has two ports, one takes packets as inputs and another outputs service class. (See Figure 3.7.)

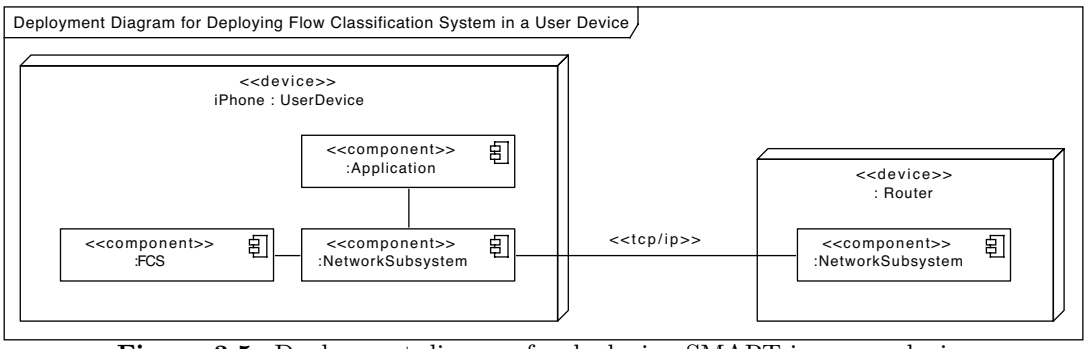


Figure 3.5: Deployment diagram for deploying SMART in a user device

Let us consider the flow classification component in detail. The packets are fed into the component through the input port, which connects the `NetworkSubsystem` component. The incoming packets are first sorted and assembled into flows via the `FlowAssembler`. The `FeatureExtractor` then transforms the flow into a feature vector. The `WorkingModeStrategy` component follows Strategy pattern deciding whether SMART should operate in classification mode or learning mode. The operation mode depends on the type of the flow. If the flow is QoS-unaware, SMART should switch to classification mode and uses the `Classifier` component to identify the service class of the flow. Otherwise, if the flow is QoS-aware, SMART would operate in learning mode and stores the feature vector along with the service class in the `Dataset` component. `Learner` can then use the data to induce the classification model and updates the classifier accordingly.

3.6.2 SMART Class Diagrams

SMART is composed of three major elements, which can be grouped into three packages, including the `FlowModel` package, the `FeatureExtraction` package and the `Classification` package. (See Figure 3.8.) The `FlowModel` package contains classes

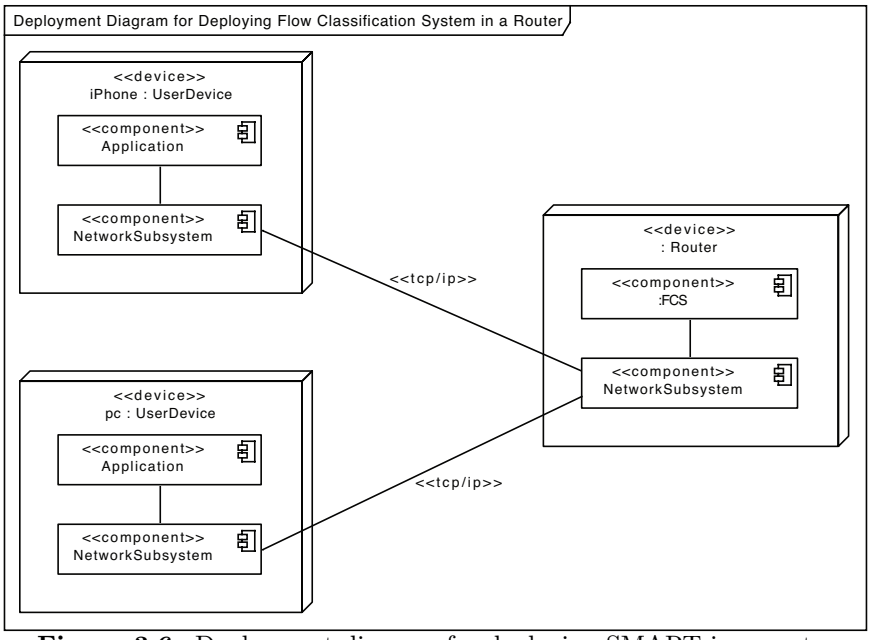


Figure 3.6: Deployment diagram for deploying SMART in a router

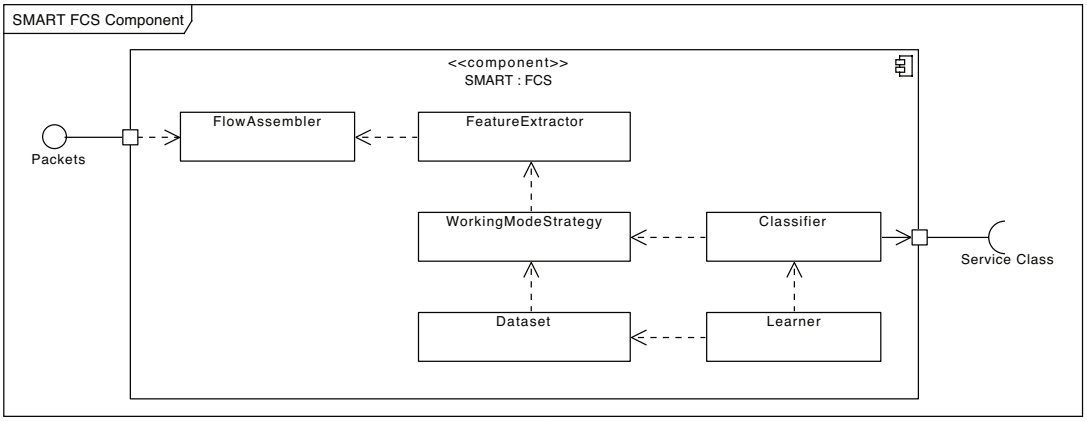


Figure 3.7: SMART Component

of packet, flow, as well as service class. The `FeatureExtraction` package consists of classes related to features and feature extraction, which includes feature value, feature vector and feature function. The `Classification` package contains classifiers and learners classes. As shown in Figure 3.8, `FeatureExtraction` and `Classification` packages depend on `FlowModel`. This is because classes in both packages deal with flows — The `FeatureExtration` extracts flow features and the `Classification` classifies flows. `Classification` package also depends on `FeatureExtraction` as the classifier takes a feature vector as input. In the following, contents of each package will be discussed in detail.

First, let us consider the `FlowModel` package, which contains all classes related to flows. As defined in Definition 3.3, a flow is a non-empty sequence of packets.

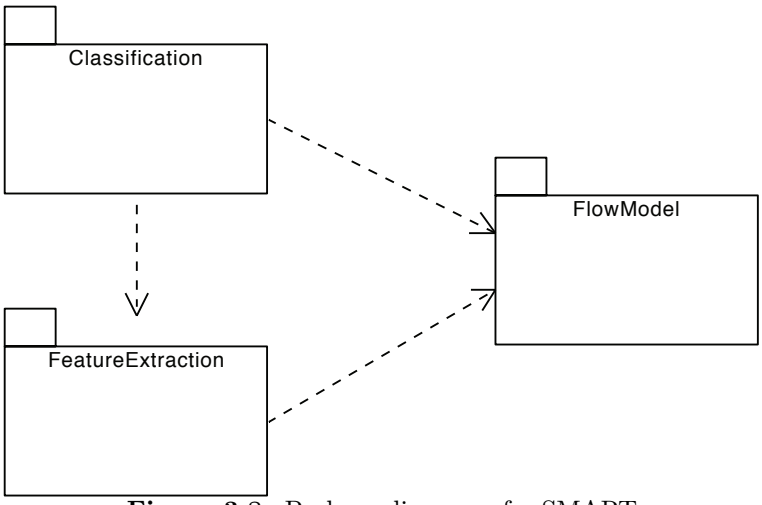


Figure 3.8: Package diagram of a SMART

Therefore, as shown in Figure 3.9, the flow is modeled as a class consists of one or more packets. Also, a flow could be either QoS-aware or QoS-unaware, where QoS-aware flow is assigned to a service class while QoS-unaware is not. As defined in Definition 3.3, if the class of the flow belongs to a set of classes $\mathcal{C}$, the flow is called QoS-aware, whereas if the class is empty, the flow is called QoS-unaware. The class diagram of the flow follows the definition closely. As shown in Figure 3.9, a flow consists of a service class, such that the service class could be either `ServiceClassAbstract` or `UnknownServiceClass`. The `ServiceClassAbtract` is a super class of all QoS service classes. A flow class also has an operation `getPrefix`, which takes an integer as input and returns its prefix of the specified length as output.

The `FeatureExtraction` package, depicted in Figure 3.10, contains all classes corresponding to feature extraction process. As defined in Definition 3.10, a feature extraction function takes a flow as input and returns a feature vector as output. It employs a feature function to compute each feature value in the feature vector. Following the definition, in our model, `FeatureExtractor` class consists of one or more `FeatureFunction` classes. Its operation, `extract`, takes flow as input and produces an instance of `FeatureVector` class, which consists of one or more feature values, as output. For feature function, we use Abstract Factory pattern to model `FeatureFunction` class. This is because all feature functions share the same functionality (compute a feature value from a flow), but their implementations are different.

The `Classification` package, shown in Figure 3.11, comprises of classes corresponding to classification and learning. These classes include `Classifier`, `Learner` and `Dataset` classes. `Classifier` class considers the values in a feature vector and assigns an appropriate service class accordingly. The learner uses data in the dataset to induce new classification model and updates the classifier.

As mentioned earlier, SMART consists of two working modes: Classification and learning modes. These modes are activated based on the class of the incoming flow. If the flow is QoS-unaware, SMART will operate in classification mode, whereas if the flow is QoS-aware, it will operate in learning mode. (See Figure 3.12.) This mode-

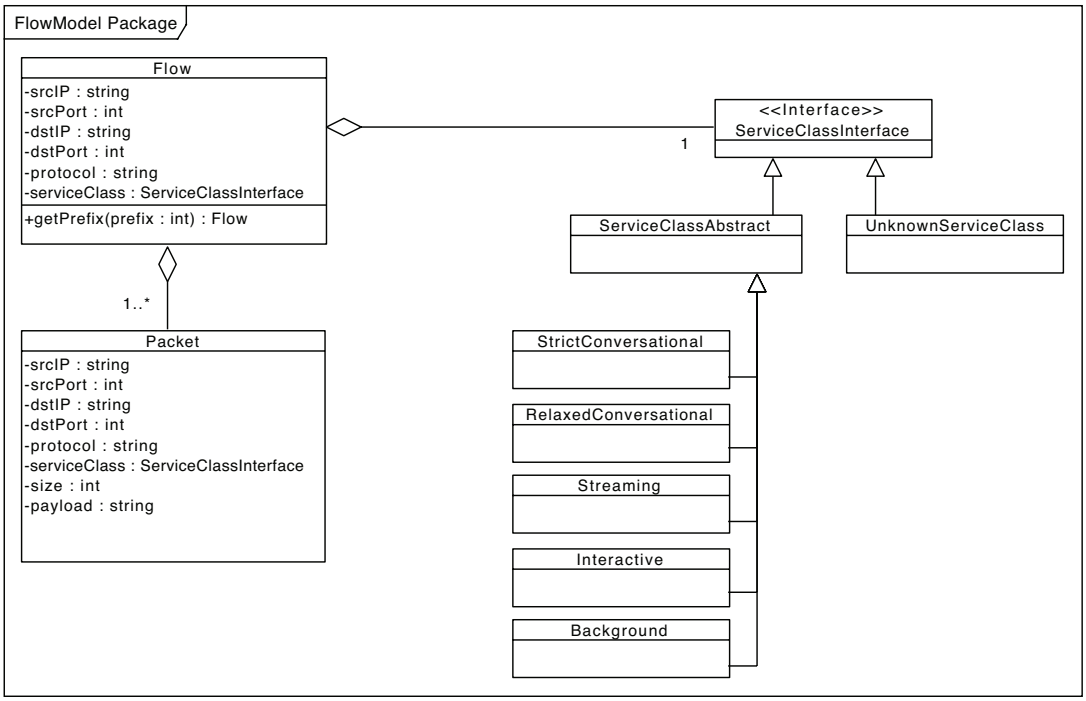


Figure 3.9: The contents of `FlowModel` package

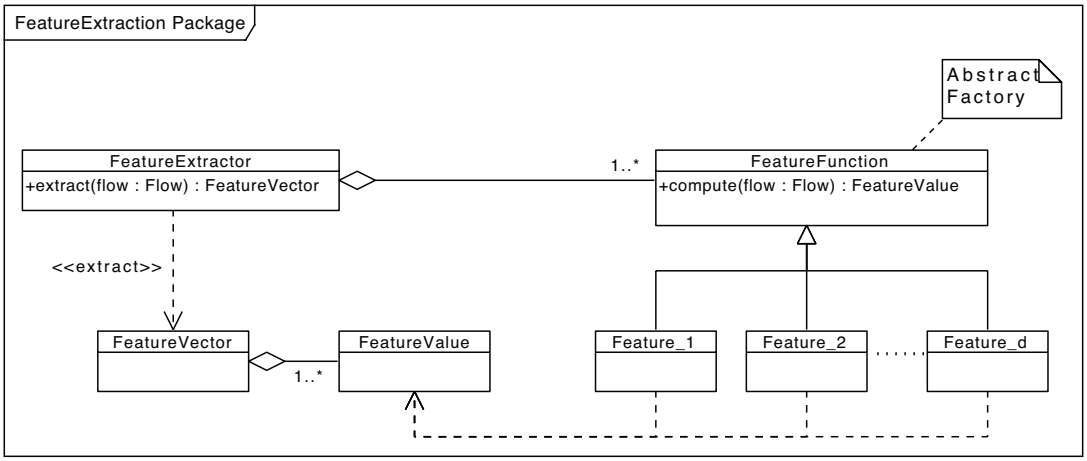


Figure 3.10: The contents of `FeatureExtraction` package

switching policy is described in `WorkingModePolicy` object, which determines the working mode based on the service class (i.e., the `ServiceClassInterface` class).

3.7 Conclusion

In this chapter, we have described a general framework of flow classification systems. The fundamental elements and processes of flow classification systems are recognized and formally defined. The components are modeled using basic mathematical con-

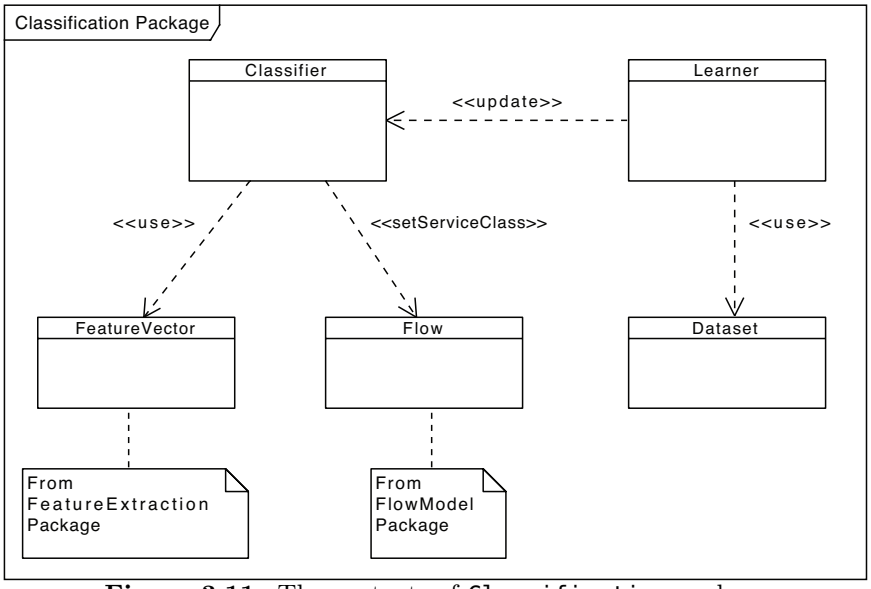


Figure 3.11: The contents of `Classification` package

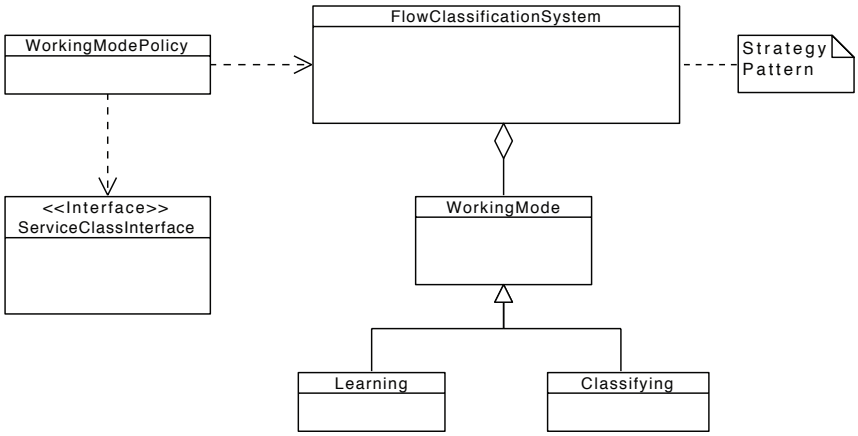


Figure 3.12: SMART working-mode switching strategy

structs, which makes the model precise, versatile, and easy to verify. We have made a distinction between flows and feature vectors, feature functions and feature values, as well as learners and classifiers. Also, we have derived the basic properties of flow classification systems based on their adaptability and their abilities to classify flows in real-time. Using our model, flow classification systems can be clearly and easily categorized and compared. Apart from mathematical model, the implementation, deployment and system decomposition of SMART are also described using UML.

However, the model still has some restrictions. The model of the packet does not contain protocol specific information such as TCP flags or window size. Thus, some approaches that utilize TCP-specific information are not supported by the current framework. Moreover, our model does not recognize host-behavior based approaches as flow classification systems as they actually classify the hosts, not the flows.

Chapter 4

Machine Learning

Machine learning is a set of methods designed to recognize (or "learn") and describe structural patterns in a set of data. They are generally categorized into "supervised" and "unsupervised" learning methods. Supervised learning is a class of learning methods that estimate patterns in a set of data — represented as a set of pairs. Each pair consists of the characteristic and the class of an object. A characteristic of an object is a fixed, predetermined collection of its descriptions, which in our context pertain to the feature vectors. The class of an object refers to its service class. The goal of supervised learning is to identify and describe the relationships between the pair, which can then be used to classify unseen instances. In contrast, unsupervised learning methods attempt to estimate the pattern in the data only from the features vectors (i.e., the classes of the instances are not given). The goal of such methods is to identify how the data are organized or structured without any explicit set of classes.

In this chapter, the discussion on the potential learners to be used in our classification system will be given. Since the service classes are always given to the QoS-aware flows (which are used as the training data) in the flow classification scenario, supervised learning perfectly suits the domain. The discussion will therefore focus mainly on supervised methods. Moreover, considering that machine learning algorithms are best described by how the learned knowledge is presented, the technique used to describe the knowledge will dictate the most suitable learning methods for obtaining the knowledge.

4.1 Decision Tree

A decision tree is an analytical tool that explicitly maps observations of an object to its value in a tree-like structure. Each node in the tree is associated with a feature and its branches are labeled with the values corresponding to the associated feature. The leaves of the tree are labeled according to the service classes into which the instances are to be classified. The classification of an instance requires testing the value of each of its features at the corresponding node. The test is done by comparing the value of the instance with the value labeled at each branch starting from the root. When the value of the instance matches with the value of a branch, the classification will

continue through that branch until a leaf is reached. The instance is then classified as the class labeled on the leaf. An example of a decision tree is shown in Figure 4.1. An algorithm that induces a decision tree from a dataset is called a "decision tree learner".

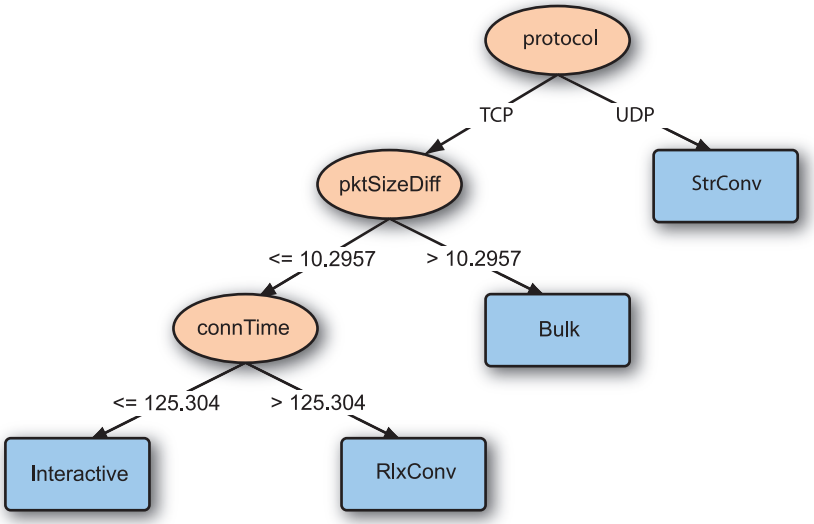


Figure 4.1: An example of a decision tree. Each branch of a node is associated to a feature. The leaves of the tree are labeled with the classes. Here, StrConv and RlxConv refer to strict and relaxed conversational classes respectively.

A decision tree consists of nodes at which the corresponding feature values are tested. Thus, the central idea of constructing a tree is to select which feature to test at each node. In the beginning, a decision tree consists of only one node (i.e., the root). The decision tree learner then selects the feature that best divides the dataset with respect to some criterion. (The selection criteria and how the selection is done will be discussed below.) A branch is created for each possible value of the selected feature (i.e., each element of the codomain of the feature) and the instances in the dataset are divided according to these values.[1] Then, at the end of each branch, a node is created and the next feature to be tested will be selected using only the instances associated with the branch. The process is repeated until the data cannot be divided further.

In general, all decision tree learners construct the tree in the same manner, with the main exception being the criterion by which the effectiveness of the features is evaluated. Examples of such measures are information gain, which is used in many learners such as Hunt et al. [HMS66] and Quinlan [Qui86], biased-corrected variant of information gain called gain ratio, introduced also by Quinlan to be used in his C4.5 decision tree learner [Qui93], and the Gini index employed by Breiman et al. [BFSO84][2]. In our research, we will restrict ourselves only to C4.5, which is generally

[1] If the codomain is continuous, an extra processing must be done. A detailed discussion regarding this topic will be given later in this section.

[2] Other metrics such as chi-square [Min89] and Bhattacharya [Bha43] and Kolmogorov-Smirnoff distances (see, e.g., [CLR67]) have been used as well. Extensive reviews on decision tree construction methods as well as feature evaluation functions can be found in [BN92], [Mur98], [LLS00], and [Kot07].

accepted as the most influential and widely-used tree learner [WF05][Kot07]. Apart from its popularity, Lim et al. [LLS00] showed that C4.5 provides the best trade-off between accuracy and learning time. Furthermore, its feature-evaluation function, which is based on information gain, has been shown to be very effective in selecting the features [BN92]. In the following, we will see in detail how C4.5 induces a tree from a given dataset.

Ross Quinlan [Qui93] proposed a method based on "information gain" to select the feature to be tested at each step while the tree is being grown. This measure results in a decision tree learning algorithm called C4.5. Information gain is derived from the concept of "entropy", which is a measure of impurity or uncertainty of a given set of data introduced by Shannon in 1948 [Sha48].

The main idea of C4.5 is focused on the entropy of the classes in the given dataset. More precisely, we want to know which features can divide a set of data into smaller subsets such that each of them contains as little entropy as possible (i.e., as many elements in them belong to the same class as possible). In the following, we will define some auxiliary functions required to describe the entropy followed by the definition of entropy.

Let X be a set. We define functions that compare two elements in the class as follows.

$$\begin{aligned} \mathit{matched}: \quad & X \times X \to \mathbb{N} \\ & (x, x') \mapsto \begin{cases} 1 & \text{if } x = x', \\ 0 & \text{otherwise.} \end{cases} \end{aligned} \tag{4.1}$$

$$\begin{aligned} \mathit{unmatched}: \quad & X \times X \to \mathbb{N} \\ & (x, x') \mapsto \begin{cases} 1 & \text{if } x \neq x', \\ 0 & \text{otherwise.} \end{cases} \end{aligned} \tag{4.2}$$

Definition 4.1 (Frequency) Let X be a set, $M = \dot{\{}m_1, \ldots, m_n\dot{\}}$ is a finite multiset over X, and $x \in X$. The *frequency of x in M* is a function

$$\begin{aligned} \mathit{Freq}: \quad & X \times \mathcal{P}(M) \to \mathbb{N} \\ & (x, \dot{\{}m_1, \ldots, m_n\dot{\}}) \mapsto \sum_{i=1}^{n} \mathit{matched}(x, m_i) \end{aligned}$$

Definition 4.2 (Frequency of a Class) Let $\mathcal{X}$ be a set of feature vectors, $\mathcal{C}$ a set of service classes, $\mathfrak{D}$ a set of all datasets over $\mathcal{X} \times \mathcal{C}$, $\mathcal{D} = \dot{\{}\langle \boldsymbol{x}_1, c_1\rangle, \ldots, \langle \boldsymbol{x}_m, c_m\rangle\dot{\}} \in \mathfrak{D}$ and $c \in \mathcal{C}$. The *frequency of c in $\mathcal{D}$* is

$$\begin{aligned} \mathit{ClassFreq}: \quad & \mathcal{C} \times \mathfrak{D} \to \mathbb{N} \\ & (c, \dot{\{}\langle \boldsymbol{x}_1, c_1\rangle, \ldots, \langle \boldsymbol{x}_m, c_m\rangle\dot{\}}) \mapsto \mathit{Freq}(c, \dot{\{}c_1, \ldots, c_m\dot{\}}) \end{aligned}$$

Given a dataset $\mathcal{D}$, randomly select an instance that belongs to a class c_j from $\mathcal{D}$ has probability

$$\frac{\mathit{ClassFreq}(c_j, \mathcal{D})}{|\mathcal{D}|}.$$

Therefore, the information that c_j conveys is

$$-\log_2\left(\frac{\mathit{ClassFreq}(c_j,\mathcal{D})}{|\mathcal{D}|}\right) \tag{4.3}$$

bits. Combining both measures, we define the entropy as follows.

Definition 4.3 Let $\mathcal{X}$ be a set of feature vectors, $\mathcal{C}=\{c_1,\ldots,c_k\}$ a set of service classes, $\mathfrak{D}$ a set of all datasets over $\mathcal{X}\times\mathcal{C}$, and $\mathcal{D}\in\mathfrak{D}$. The *entropy of* $\mathcal{D}$ with respect to classes is a function

$$\begin{aligned} H: \quad & \mathfrak{D}\to\mathbb{R} \\ & \mathcal{D}\mapsto -\sum_{j=1}^{k}\left(\frac{\mathit{ClassFreq}(c_j,\mathcal{D})}{|\mathcal{D}|}\right)\times\log_2\left(\frac{\mathit{ClassFreq}(c_j,\mathcal{D})}{|\mathcal{D}|}\right) \end{aligned}$$

The entropy can be interpreted as the weighted average of the information conveyed by a class in a dataset where the weight is the probability of each class. With entropy, we measure the discriminability of a feature as follows.

Definition 4.4 (Information Gain) Let $\mathcal{V}=\{V_1,\ldots,V_d\}$ be a set of feature functions such that $V_i:\mathcal{F}\to D_i, 1\le i\le d$, $d\in\mathbb{N}^+$, $\mathcal{X}_d=D_1\times\ldots\times D_d$, $\mathcal{C}$ a set of service classes, $\mathfrak{D}$ a set of all datasets over $\mathcal{X}_d\times\mathcal{C}$. Given a feature function $V_j:\mathcal{F}\to D_j\in\mathcal{V}$, $1\le j\le d$, the *information gain of* V_j *relative to* $\mathcal{D}$ is

$$\begin{aligned} \mathit{Gain}: \quad & \mathfrak{D}\times\mathcal{V}\to\mathbb{R} \\ & (\mathcal{D},V_j)\mapsto H(\mathcal{D})-\sum_{x\in D_j}\frac{|\mathcal{D}^x|}{|\mathcal{D}|}H(\mathcal{D}^x) \end{aligned} \tag{4.4}$$

where $\mathcal{D}^x=\dot{\{}\langle x_1,\ldots,x_j,\ldots,x_d,c\rangle\in\mathcal{D}\mid x_j=x\dot{\}}$.

Intuitively, the information gain of a feature function V_j is the difference in entropy of the original dataset compared to the ones that are divided with respect to the outcomes of V_j, weighted by the probability of each outcome. That is, the information gain of V_j is the expected reduction of entropy after knowing the values of V_j.

The information gain is used in Iterative Dichotomiser 3 (ID3) tree-learning algorithm [Qui86], the predecessor of C4.5. However, it biases toward features with many outcomes. This is because higher outcomes mean more partitions, which, in turn, tend to have less class impurity. In an extreme case, imagine a dataset of 100 flows with each flow having a unique source IP address. Using information gain alone, the learner would construct a tree with a single node, *srcIP*, which has 100 branches, and each branch, marked with the source IP of each flow, points to the class of the flow. The resulting tree would be too precise and overfit the training data. In response, another measure called "split information", which aims to suppress the bias, is introduced [Qui93].

Definition 4.5 Let $\mathcal{V}=\{V_1,\ldots,V_d\}$ be a set of feature functions such that $V_i:\mathcal{F}\to D_i, 1\le i\le d$, $d\in\mathbb{N}^+$, $\mathcal{X}_d=D_1\times\ldots\times D_d$, $\mathcal{C}$ a set of service classes, $\mathfrak{D}$ a set

of all datasets over $\mathcal{X} \times \mathcal{C}$. Given a feature function $V_j : \mathcal{F} \to D_j \in \mathcal{V}$, $1 \leq j \leq d$, the *split information of V_j relative to $\mathcal{D}$* is

$$\begin{aligned} \mathit{SplitInfo}: \quad & \mathfrak{D} \times \mathcal{V} \to \mathbb{R} \\ & (\mathcal{D}, V_j) \mapsto -\sum_{x \in D_j} \frac{|\mathcal{D}^x|}{|\mathcal{D}|} \times \log_2 \left(\frac{|\mathcal{D}^x|}{|\mathcal{D}|} \right) \end{aligned} \tag{4.5}$$

where $\mathcal{D}^x = \dot{\{}\langle x_1, \ldots, x_j, \ldots, x_d, c\rangle \in \mathcal{D} \mid x_j = x\dot{\}}$.

Split information is essentially the entropy of $\mathcal{D}$ with respect to V_j, whereas entropy introduced in Definition 4.3, is the entropy of $\mathcal{D}$ with respect to the classes. It represents the information obtained by dividing $\mathcal{D}$ into smaller sets regarding to outcomes of V_j. Then, we can use the following criteria to select a feature as follows.

Definition 4.6 Let $\mathcal{V} = \{V_1, \ldots, V_d\}$ be a set of feature functions such that $V_i : \mathcal{F} \to D_i, 1 \leq i \leq d$, $d \in \mathbb{N}^+$, $\mathcal{X}_d = D_1 \times \ldots \times D_d$, $\mathcal{C}$ a set of service classes, $\mathfrak{D}$ a set of all datasets over $\mathcal{X} \times \mathcal{C}$. Given a feature function $V \in \mathcal{V}$, the *gain ratio of V relative to $\mathcal{D}$* is

$$\begin{aligned} \mathit{GainRatio}: \quad & \mathfrak{D} \times \mathcal{V} \to \mathbb{R} \\ & (\mathcal{D}, V) \mapsto \frac{\mathit{Gain}(\mathcal{D}, V)}{\mathit{SplitInfo}(\mathcal{D}, V)}. \end{aligned} \tag{4.6}$$

Gain ratio normalizes the gain of a feature by the uniqueness of the feature itself. If the feature contains many uniformly distributed values, the split information will be high and, thus, the gain ratio will be low. For example, consider the aforementioned scenario, the split information of the feature srcIP (which is unique for every flow) is $\log_2 m$ where $m = |\mathcal{D}|$. Suppose we have another feature, transport protocol, that splits the dataset into two equal subsets, the split information of transport protocol will be 1. If the two features yield the same information gain, the feature transport protocol would be selected according to the gain ratio criterion.

At first glance, it may seem that the gain ratio can only be applied to finite discrete domains as the entropies $H(\mathcal{D}^x)$ of all elements $x \in \mathcal{D}_j$ have to be computed. However, according to Quinlan [Qui93], this is not necessary. Since the dataset is finite, there could only be finite values of any given feature. Take the feature function *avgDataTPUT* in the dataset shown in Table 3.2 (on page 49) as an example. Although its codomain is $\mathbb{R}$, there exist only 25 outcomes of the feature function in the dataset. Thus, instead of computing the gain ratio of the entire set of real numbers (which is impossible as it is infinite and uncountable), one can compute only that from the feature values present in the dataset. In addition, in case of discrete values, after a feature is selected to be tested at a node, branches are created for all possible values of the feature. In case of continuous feature values, however, C4.5 creates only two branches that splits the dataset into two sets (e.g., set of instances that has feature values $> x$ and $\leq x$) in order to restrict the number branches at each node. If needed, the same feature can be tested again at lower nodes.

To illustrate how the splitting is achieved, consider a sequence $(s_1, \ldots, s_n)$, which contains all distinct values of a feature in a dataset. The components of the sequence

are sorted. There are, in turn, $n-1$ splits to be examined (i.e., for each s_i, $1 \leq i \leq n-1$, the dataset can be split into $> s_i$ and $\leq s_i$). At each split point s_i, the overall gain ratio is computed (at this point, only two feature values are considered: $> s_i$ and $\leq s_{i+1}$) and the splitting point that yield the highest gain ratio is selected.

Taken together, the tree generation process is carried as follows:

- Compute gain ratio of all features using the entire dataset and select the one that has the highest gain to be tested at the root node. If the codomain of the selected feature is discrete, compute the gain ratio using all elements of the codomain. Otherwise, if the codomain is continuous or infinite, determine the splitting point using the method explained above.

- Then, after the root node is created, add branches to the node whereby each of them is labeled with a possible outcome of the selected feature function.

- Next, if the selected feature is discrete, remove the feature from the set of features (as all values of the feature are already considered). On the other hand, if the feature is continuous or infinite, the feature might be tested again and it will be kept in the set of features.

- At each branch, a node is created the same way as the root. However, instead of using the entire set of data, only the instances whose feature values correspond to the branch are considered. That is, at each node, the dataset is divided into smaller partitions, each of which corresponds to the feature value labeled at each branch.

The process is repeated until all instances corresponding to a branch belong only to a single class, at which point a leaf node labeled with that class is created below the branch. Also, if there are no more features to be determined (i.e., all features have been tested along the path from the root), a leaf node labeled with the dominant class of the partition of the dataset associated to the branch is created. The detailed tree construction process is described in Algorithm A.1 in Appendix A.

In the context of flow classification, decision tree learners have been employed by a number of flow classification systems. Early et al. [EBR03] implements the C5.0 algorithm [Rul07], which is a commercial, unpublished descendant of C4.5, to build a decision tree classifier. Later in 2006, C4.5 is employed and evaluated by Williams et al. [WZA06] along with other learners including Naive Bayes algorithm and Bayesian Network. The results show that C4.5 can classify the flows faster than other algorithms while maintaining high accuracy. Detailed discussion on this issue will be provided in Section 4.3.

4.2 Covering Algorithms

C4.5 generates decision trees by analyzing the effectiveness of features and dividing the set of instances into smaller sets according to the values of the selected feature. Then, within these smaller sets, the steps are repeated until the data cannot be divided

any further. This strategy is referred to as a "divide-and-conquer" method. In this section, another kind of learning approach called a "sequential covering" method will be explored. In a sequential covering approach, a set of classification rules is induced from the common characteristics of the instances of each class. After a rule is learned, the instances that are "covered" by the rule are separated (i.e., removed) from the dataset and the process iterated until all examples are covered (i.e., no more example left in the dataset). A rule consists of two parts: precondition and conclusion. The precondition part has one or more logical criteria or tests, which the instance to be classified has to meet; the conclusion specifies the class to be assigned to the given instance if it conforms to the logical expressions in the precondition part. An instance is said to be covered by a rule if its feature values conform to the tests in the precondition part and its class matches the class specified in the conclusion part of the rule.

In this thesis, we will focus on a sequential covering learner called Repeated Incremental Pruning to Produce Error Reduction (RIPPER). It was introduced by William Cohen in 1995 [Coh95] as an improved version of another learner called Incremental Reduced Error Pruning or IREP, developed by Fürnkranz and Widmer in 1994 [FW94]. RIPPER constructs a rule set exactly in sequential fashion, i.e., rules are created one-by-one. Once a rule is found, the instances covered by the rule are removed from the dataset. In each iteration, the learner always tries to construct a rule that covers the largest number of instances. The process is repeated until there is no instance of the target class left or the newly found rule yields an unacceptable error rate.

Conceptually, a rule starts with an empty precondition while it is being generated and a logical test is then repeatedly added to it. In this process, the rule is said to be "grown". The rule is grown until it covers no instance that is not the target class (i.e., the rule is specific enough to cover only the target class). Growing a rule this way typically results in a rule that is too specific and overfits the considered dataset. Therefore, it has to be generalized or "pruned" by eliminating some conditions afterwards. To this end, it is essential to find an effective growing and pruning criterion in order to construct a plausible rule set. In the following, we will see how RIPPER creates a rule set from a dataset as well as the heuristics that are employed to grow and prune the rule.

Given a target class c, RIPPER starts the rule learning by splitting the dataset into two sets $\mathcal{D}^c$ and $\mathcal{D}^{\bar{c}}$ where $\mathcal{D}^c$ is the set of instances that belongs to c and $\mathcal{D}^{\bar{c}}$ contains the instances of the other classes. $\mathcal{D}^c$ is then divided further into $Grow^c$ and $Prune^c$ where the former set will be used to grow the rule whereas the latter will be used to prune it. Likewise, $\mathcal{D}^{\bar{c}}$ is also divided into $Grow^{\bar{c}}$ and $Prune^{\bar{c}}$. The rule is first started with empty precondition and, throughout the growing process, conditions are recurrently added. A condition, like a node in decision tree, is a test of a feature value, which is selected by RIPPER based on the information gain criterion[3].

Unlike a decision tree, however, when a feature is selected, only the value yielding

[3]This selection criterion has been initially employed by another sequential rule-learning algorithm called FOIL [Qui90] which was introduced by Ross Quinlan who also introduced C4.5. FOIL, in fact, is more expressive than RIPPER as it learns first-order logic rules as opposed to propositional logic rules in RIPPER. However, since propositional rule is sufficient to our task, we will restrict ourselves only to propositional rules learning.

the highest gain is kept in the rule. (In C4.5, when a feature is selected to be tested at a node, a branch associated to each possible feature value is created under that node. In this sense, adding a condition is analogous to adding a node with a single branch.) Also, because a sequential algorithm finds rules for only one class at a time, the entropy used to calculate information gain does not measure the overall purity of all classes as in C4.5; rather, it only measures the purity of the target class. To be precise, let r be a rule, and l be a condition to be added to r:

$$GrowEval(l, r) = t^c \left(\log_2 \frac{t'^c}{t'^c + t'^{\bar{c}}} - \log_2 \frac{t^c}{t^c + t^{\bar{c}}} \right) \tag{4.7}$$

where t^c and $t^{\bar{c}}$ are numbers of instances of the target and non-target classes covered by r, and t'^c and $t'^{\bar{c}}$ are numbers of instances of target and non-target classes after l is added to r respectively. That is, Equation (4.7) measures the difference of the information before and after a condition is added to a rule. A rule is grown by repeatedly adding the conditions that best describe instances in the growing set (i.e., $Grow^c$) until the rule does not cover any element in $Grow^{\bar{c}}$ (i.e., covers only the target class). Then it is pruned based on its pruning sets whereby the evaluation metric is:

$$PruneEval(r) = \frac{t^p - t^{\bar{p}}}{t^p + t^{\bar{p}}} \tag{4.8}$$

where t^p and $t^{\bar{p}}$ are numbers of instances in $Prune^c$ and $Prune^{\bar{c}}$ covered by r respectively. In other words, the rule is evaluated on its ability to distinguish the target class from other classes.

At any rate, the growing and pruning processes are used to construct only one rule at a time. The next concern is when to stop adding rules to the rule set and avoid overfitting the data. In RIPPER, the minimum description length (MDL) is employed as a stopping criterion. The concept of MDL is related to Shannon's message information length that has been discussed earlier. In the MDL approach, the induced rule set is thought as a theory that explains a dataset. A larger, more complete theory would be able to explain more instances in the dataset than the simpler one. Here, instances that fail to be explained are called "exceptions". Explanation completeness comes, indeed, with a price — a large and more complete theory would require more bits than a smaller one. According to the MDL principle, similar to Occam's razor, the best theory is one that requires the smallest number of bits to encode both the theory itself and the exceptions not covered by the theory. In RIPPER, the length of a rule is encoded as:

$$\|k\| + k \log_2 \left(\frac{1}{(k/n)} \right) + (n - k) \log_2 \left(\frac{1}{1 - (k/n)} \right)$$

where k is the number of conditions in the rule, n is the number of possible conditions that could appear in a rule and $\|k\|$ is the number of bits required to encode the integer k. The length of the theory (i.e., entire rule set) is the sum of the length of all rules.

The length of the exceptions is computed as follows:

$$\begin{aligned}
&\log_2(|Prune^c \cup Prune^{\bar{c}}| + 1) \\
&+ fp \times \left(-\log_2 \left(\frac{fp + fn}{2t} \right) \right) \\
&+ (t - fp) \times \left(-\log_2 \left(1 - \frac{fp + fn}{2t} \right) \right) \\
&+ fn \times \left(-\log_2 \left(\frac{fn}{u} \right) \right) \\
&+ (u - fn) \times \left(-\log_2 \left(1 - \frac{fn}{u} \right) \right)
\end{aligned}$$

where $t = t^p + t^{\bar{p}}$, $u = |Prune^c \cup Prune^{\bar{c}}| - t$, fp and fn are the number of instances in $Prune^c$ and $Prune^{\bar{c}}$ that are misclassified by the rule set respectively. Details on how these calculations are obtained can be found in Cohen's original paper on RIPPER [Coh95] and Quinlan's studies on the description sizes of rule sets [Qui94][Qui95]. RIPPER stops adding new rules when the total of the description length of the current rule set is δ bits larger than that of the smallest rule set found so far. Here, we use $\delta = 64$ as suggested by Cohen [Coh95].

After a rule set is induced from the given dataset, the rule set is then revised again to optimize the overall performance of the set. In this phase, for each rule r in the rule set, a replacement rule r' is formed by growing and pruning r again. This time, instead of information gain, the growing criterion is the MDL of the entire ruleset. In addition to the replacement rule r', r is also revised again by greedily adding more conditions to it. One of the three variants of r will then be selected according to MDL of the entire rule set[4]. In Algorithm A.2, the pseudo code of RIPPER is illustrated.

Effective rule induction depends upon three fundamental aspects, namely, growing criterion, pruning criterion and stopping criterion. Growing and pruning criteria can be thought together as rule evaluation criteria while the stopping criterion determines the value of the entire rule set. In RIPPER, the rule is evaluated by the ratio of the number of covered target instances to that of all covered instances (see Equation (4.8)) and the rule is grown using the information gains of candidate conditions.

Apart from the aforementioned criteria, many other metrics are used to evaluate rules. For instance, CN2 [CN89], employs entropy to evaluate the induced rules. IREP [FW94] (the rule learner on which RIPPER is based) uses error rate that the rule set made on the pruning set as a stopping criterion. In 1990, Cestnik [Ces90] proposed another evaluation criterion called the m-estimate, which assumes that each rule covers m instances *a priori*.

According to Fürnkranz and Flach [FF03], however, almost all major evaluation metrics employed in current rule learners are closely related and information gain is particularly suitable for covering algorithms. They also suggested that RIPPER's rule

[4]It is worth noting that, to obtain a rule set with lowest MDL, each rule in R has to be revised. Although the most efficient rule set is assured, this makes RIPPER a rather computationally expensive algorithm.

set generalization method is superior to others, which leads us to select RIPPER as a candidate to be used in our flow classification system.

Another successful rule learning method that combines C4.5 and RIPPER is proposed by Frank and Witten [FW98]. To avoid rule set revision after all rules are found (as in RIPPER), C4.5 is employed to generate a rule instead of the ordinary grow-and-prune method. In their approach, the decision tree is created using C4.5 algorithm. However, instead of expanding every node to cover the entire dataset like the usual tree-generation method, only the node with lowest entropy is expanded, resulting in a tree that is not fully explored or a "partial tree". The leaf that covers most instances is selected and a rule is then extracted from its path from the root. Finally, the instances that are covered by the newly obtained rule are removed and the process is repeated as in other sequential covering methods. This method is called "PART" as it uses a partial tree to generate rules. Because it employs C4.5 to generate rules and repeats the process like ordinary sequential-covering learners, we do not find it necessary to describe the algorithm in detail. However, as it bridges two learning schemes that are presented and used in our experiments, it is included in our experiments as well. Detailed discussions on the algorithm can be found in [FW98] and [WF05].

In flow classification, to the best of our knowledge, sequential-covering approaches have not been used in any existing FCSs except in our system, which was published in [AS07b], [AS07a] and [Ana10].

4.3 Statistical Learning Methods

Statistical learning methods predict the class of a new, unseen instance through statistical approaches. One of the simplest statistical learning methods is called "Naive Bayes", which is derived from the Bayes theorem of conditional probability. It was originally described by Nilsson in 1965 [Nil65] and was first used to solve classification problems by Cestnik et al. in 1987 [CKB87]. In a nutshell, Naive Bayes predicts the *most probable* class of an instance with respect to a given dataset. Before we begin our discussion on the method, we will first introduce a few notations and give the formulation of the Bayes theorem. We shall write $\mathrm{P}(c)$ to denote the prior probability of c or, in other words, the probability that c holds. In our flow classification domain, we assume that all classes could occur with the same probability, i.e., $\mathrm{P}(c_i) = \mathrm{P}(c_j)$ for all $c_i, c_j \in \mathcal{C}$. Likewise, given a feature vector $\boldsymbol{x}$, we use $\mathrm{P}(\boldsymbol{x})$ to denote the probability that $\boldsymbol{x}$ is observed and $\mathrm{P}(\boldsymbol{x}|c)$ to denote the probability of observing $\boldsymbol{x}$ in a domain where c holds, i.e., the probability of $\boldsymbol{x}$ when we fix the class to c. Finally, we use $\mathrm{P}(c|\boldsymbol{x})$ to denote the probability of c given $\boldsymbol{x}$.

The Bayes theorem is given by:

$$\mathrm{P}(c|\boldsymbol{x}) = \frac{\mathrm{P}(c)\mathrm{P}(\boldsymbol{x}|c)}{\mathrm{P}(\boldsymbol{x})}. \tag{4.9}$$

Given a vector $\boldsymbol{x}$, the goal of Naive Bayes method is the find the class $c \in \mathcal{C}$ that has the highest $\mathrm{P}(c|\boldsymbol{x})$. Such class is called the "maximum a posteriori (MAP)" class,

denoted by c_{MAP}, which is precisely characterized as follows:

$$\begin{aligned} c_{\text{MAP}} &= \underset{c \in \mathcal{C}}{\text{argmax}}\ \text{P}(c|\boldsymbol{x}) \\ &= \underset{c \in \mathcal{C}}{\text{argmax}}\ \frac{\text{P}(c)\text{P}(\boldsymbol{x}|c)}{\text{P}(\boldsymbol{x})} \end{aligned}$$

The term $\text{P}(\boldsymbol{x})$ can be dropped because it is the same for all classes and will not affect the calculation. Hence,

$$c_{\text{MAP}} = \underset{c \in \mathcal{C}}{\text{argmax}}\ \text{P}(c)\text{P}(\boldsymbol{x}|c).$$

An instance $\boldsymbol{x}$ is a vector composed of feature values (i.e., $\boldsymbol{x} = \langle x_1, \ldots, x_d \rangle$), thus,

$$\begin{aligned} c_{\text{MAP}} &= \underset{c \in \mathcal{C}}{\text{argmax}}\ \text{P}(c)\text{P}(\langle x_1, \ldots, x_d \rangle|c) \\ &= \underset{c \in \mathcal{C}}{\text{argmax}}\ \text{P}(c)\text{P}(x_1, \ldots, x_d|c). \end{aligned}$$

If we assume that the feature values are independent given a class, we get

$$\begin{aligned} c_{\text{MAP}} &= \underset{c \in \mathcal{C}}{\text{argmax}}\ \text{P}(c)\text{P}(x_1|c)\text{P}(x_2|c), \ldots, \text{P}(x_d|c) \\ &= \underset{c \in \mathcal{C}}{\text{argmax}}\ \text{P}(c) \prod_{i=1}^{d} \text{P}(x_i|c). \end{aligned} \tag{4.10}$$

This means that c_{MAP} is a class c that maximizes the sum of the product of the probability of all feature values given c.

The question then is how we can know the probability of $\text{P}(x_1|c)$ to $\text{P}(x_d|c)$ as well as the probability of the class $\text{P}(c)$. In practice, they are obtained from the dataset by analyzing the statistics of feature values and classes. For example, consider the dataset in Table 3.2. The size of the dataset is 25, five of which are *RlxConv* and the probability of *RlxConv* is thus

$$\text{P}(\textit{RlxConv}) = \frac{5}{25} = 0.2.$$

From all *RlxConv* instances, two of them have a destination port of 5050. Thus, the probability of $\textit{dstPort} = 5050$ given the class *RlxConv* is

$$\text{P}(\textit{dstPort} = 5050|\textit{RlxConv}) = \frac{2}{5} = 0.4.$$

In Naive Bayes, to classify an instance, one only has to repeatedly compute the probability of each feature value given all classes in $\mathcal{C}$ and select the class that yields highest probability. That is, Naive Bayes simply looks for the class with the highest probability with respect to the given dataset, without constructing any concrete classification model. Nevertheless, although the probability of the class is obtained using the statistics of the feature values of all instances in the dataset, the entire dataset is not required in the classification phase — only the frequencies of each feature value

and the classes are. In any case, it is important to note that Naive Bayes works under the assumption that the features are independent from each other, which is usually not the case. The method has been shown to work comparatively well in practice, however, and in some cases performs better than more complicated methods such as a decision tree or rule learners [DP97]. Furthermore, because c_{MAP} is the product of the probabilities of all feature values, if there exist any x_i such that $\text{P}(x_i|c) = 0$, the resulting product would be zero. This problem can be fixed using Laplace estimator or m-estimate, which adds one to all numerators and adds the number of added ones to the denominator [Ces90][Kot07].

While the method described above can handle only discrete values, Naive Bayes is also designed to handle numerical values. Typically, Naive Bayes assumes that each numerical feature has a Gaussian probability distribution, which is characterized by the mean and standard deviation. We can compute the probability of observing a value of a feature as follows. Given a class $c \in \mathcal{C}$, a feature $V : \mathcal{F} \rightarrow D$, and $x \in D$, the probability of an instance of class c that $V = x$ is:

$$\text{Pr}_{\mathsf{Gauss}}(V = x|c) = \frac{1}{\sigma_c\sqrt{2\pi}} e^{-(x-\mu_c)^2/2\sigma_c^2} \tag{4.11}$$

where μ_c is the mean and σ_c is the standard deviation (SD) of the values of V with respect to the class c.

Practically, the mean μ and SD σ of a feature are calculated from the values of the feature in the dataset. To determine the probability of a feature value, in the discrete case, one has to count the frequency of the value and compare it with all other values in the dataset as discussed earlier. In case of a numerical value, x in (4.11) has to be instantiated with the value of interest along with the μ and σ calculated from the dataset. Consider again the instances in the dataset in Table 3.2 as an example. The mean and SD of the values of *connTime* of instances that belong to class *RlxConv* are 1.568 and 1.661 respectively. If we are considering the probability of an instance with the feature value, say, $connTime = 2.00$, which belongs to *RlxConv*, we will get:

$$\begin{aligned} \text{Pr}_{\mathsf{Gauss}}(connTime = 2.00|RlxConv) &= \frac{1}{1.661\sqrt{2\pi}} e^{-(2.00-1.568)^2/2(1.661)^2} \\ &= 0.232 \end{aligned}$$

After the probability of the value of each feature is obtained (either continuous or discrete), the likelihood of the class can then be calculated using Equation (4.10) as discussed earlier. Nevertheless, in practice, some features do not follow a Gaussian distribution. For instance, Figure 6.2 illustrates the distribution of the average throughput of instances in a dataset[5], which does not have a Gaussian distribution. Assuming all numeric features to have a Gaussian distribution is, therefore, impractical and would lead to low classification performance. To this end, instead of relying on the Gaussian assumption, John and Langley [JL95] proposed a kernel density method to estimate the probability distribution.

[5]This dataset is called "WIDE Dataset" and it will be used later in our evaluations. A discussion over this dataset as well as detailed explanation of the distribution shown in the Figure will be given in Chapter 6.

Consider a dataset $\mathcal{D}$ and a dataset $\mathcal{D}^c \subseteq \mathcal{D}$ such that all elements in $\mathcal{D}^c$ belong to class c, i.e.,

$$\mathcal{D}^c = \dot{\{}\langle x_{11}, \ldots, x_{i1}, \ldots, x_{d1}, c_1\rangle, \ldots, \langle x_{1m}, \ldots, x_{im}, \ldots, x_{dm}, c_m\rangle\dot{\}}$$

such that $c_k = c, 1 \leq k \leq m$. The kernel estimation of $V_i = x$ is given by:

$$\Pr_{\mathsf{Kernel}}(V_i = x|c) = \frac{1}{m}\sum_{j=1}^{m}\left(\frac{1}{\sigma_c\sqrt{2\pi}}e^{-(x-\mu_{cj})^2/2\sigma_c^2}\right) \tag{4.12}$$

where $x_{i,j}$ is the value of feature V_i of the j-th element in $\mathcal{D}^c$ and $\mu_{cj} = x_{ij}$.

In words, the kernel estimation is the sum of multiple Gaussian probability estimations whose mean of each estimation is an occurrence of the interested feature value (which is V_i, in the equation above), which is seen in the dataset. In a usual Gaussian estimation, the mean μ_c is the average of all values of the interested feature that are found in the dataset and the probability is evaluated only once using (4.11), whereas in case of a kernel estimation, m evaluations are performed.

By summing up multiple Gaussian distributions, the kernel method can capture the distributions with multiple peaks. This method has been shown by John and Langley [JL95] to be very useful and outperform Gaussian-based Naive Bayes in many domains[6]. In the context of flow classification, Naive Bayes method was employed by Moore and Zuev in 2005 [MZ05b] and it has been shown that, by simply changing the estimation method from Gaussian to kernel, the overall prediction correctness is improved from 65.26% to 93.50%. In 2006, Williams et al. [WZA06] evaluated the method along with other learning algorithms including C4.5. The results show that given the same feature and data sets, the learners perform equally well in term of classification accuracy. The main difference, however, is their computational performance. Williams et al. found that C4.5 can classify the flows faster than other algorithms while maintaining high accuracy. Naive Bayes using a kernel estimator, on the other hand, performs the worst in terms of correctness but it is much faster considering the classification model building time. This finding is also confirmed by our evaluations. The missing element in the evaluations presented in [WZA06], however, is the flow observation time. Because learning can be carried out offline, the time the learner takes to learn is not relevant; the time that the flow classification system takes to classify the flows — including features extraction — is much more important.

Another method based on Bayes theorem is called "Bayesian Network", which is essentially a graph describing probability relationships among features. Each node in the graph is associated with a feature whereby the arcs linking two nodes represent causal relationships between features. If there is no arc between any pair of nodes, it means that the two features associated with both nodes are independent. For example, take a Bayesian Network describing a probability relationship between a symptom and a disease contains two nodes, one of which is associated with the symptom and the other with the disease given the symptom. The probability of observing the symptom is given in the symptom node and the probability of the disease given the symptom is

[6] At any rate, both methods perform just as good if the feature values are Gaussian distributed.

indicated in the arc. With this information, one can compute the probability of having the disease if the patient is observed with the symptom. If the disease is associated with more than one symptom, then there would be arcs linked from other nodes, which correspond to the probabilities of other symptoms.

Since we have no background knowledge in the dependencies of the features, it is rather difficult to come up with an appropriate Bayesian network. In our experiments, we will focus only on kernel-based Naive Bayes as it is more accurate than its Gaussian counterpart. Still, there are several approaches, such as [HMC99], [Chi02], and [Ad03], which aim to automatically determine the network structure from a given set of data.

4.4 Instance-Based

If knowledge is hidden in the data, why not use the data themselves to represent the knowledge instead of employing an additional model, such as a decision tree or a rule set? Using the data instances to represent the knowledge is the key concept of instance-based learning. Instance-based learning does not build any classification model or any abstraction from the data: it uses the dataset itself as part of the classifier to classify unseen instances [AKA91]. This kind of classifiers evolve around a classic learning algorithm called "k-nearest-neighbor (k-NN)", which was introduced by Cover and Hart in 1967 [CH67]. In k-NN, an instance is thought of as a point in a d-dimensional space, in which each dimension is related to one of the d features. It is conjectured that the instances of the same class should have the same properties and, therefore, would be close to each other. To classify a new instance, the algorithm finds k nearest instances (or "neighbors") and assigns the majority class of those instances to the new one.

The performance of k-NN is affected by two following aspects: the distance metric and the value of k. The employed distance metric should minimize the distances among instances from a similar class and maximize the distances among those from different classes. In general, Euclidean distance is used as the distance measure. However, other geometric distances such as Manhattan distance [Kra87] can also be used. Examples of the distance metrics used in various k-NN approaches are shown in Figure 4.3. A more detailed review on instance-based learning and distance metrics can be found in [Aha97] and [dA98]. In any case, if different features have different scales of measurement, the distances have to be normalized first. Otherwise, the features with a larger scale of measurement would have more influence than those with smaller scales. In a noisy domain, the noisy instances that reside near the new instance could easily win the majority vote. This problem could be solved by increasing k to suppress the effect of the noise. However, the larger k will also lead to rougher areas that define the classes. This means that even though the instances of the same class, say c, are well-clustered together, a new instance of this class, which resides very close to the existing ones, might be classified as another class if c is not the majority class of the k instances. Figure 4.2 illustrates such scenario. At any rate, one of the disadvantages of the k-NN approach is that there exists no systematic way to identify the best k for a given domain. One can only iteratively apply different k to determine the best classification

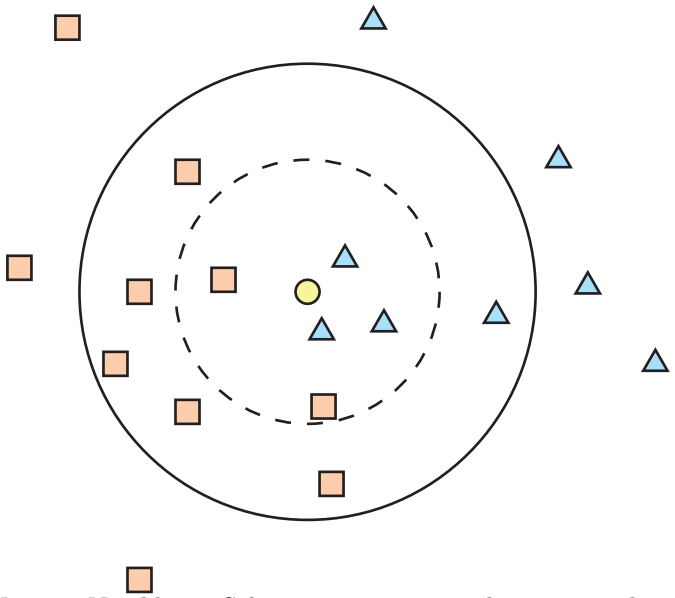


Figure 4.2: k-Nearest Neighbor. Selecting appropriate k is essential to the classification performance. If $k = 5$, the new instance (the middle circle) would be classified as triangle whereas, if k is set to 10, it would be classified as square.

performance [Kot07][WF05].

Let $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d$, the distance metrics can be defined as follows.

$$
\begin{aligned}
\textit{Euclidean}: \quad & \mathrm{Diff}_{\mathsf{Euc}}(\boldsymbol{x}, \boldsymbol{y}) = \sqrt{\left(\sum_{i=1}^{d} |x_i - y_i|^2\right)} \\
\textit{Manhattan}: \quad & \mathrm{Diff}_{\mathsf{Man}}(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=1}^{d} |x_i - y_i| \\
\textit{Chebychev}: \quad & \mathrm{Diff}_{\mathsf{Che}}(\boldsymbol{x}, \boldsymbol{y}) = \max_i |x_i - y_i| \\
\textit{Camberra}: \quad & \mathrm{Diff}_{\mathsf{Cam}}(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=1}^{d} \frac{|x_i - y_i|}{|x_i + y_i|}
\end{aligned}
$$

Figure 4.3: Approaches to define distance between $\boldsymbol{x}$ and $\boldsymbol{y}$. (Originally shown in [Kot07])

Instance-based classification algorithms are said to be *lazy* as they defer the calculation process until when it is actually needed (i.e., at the classification process) [Mit97][WF05]. *Eager* algorithms, such as a decision tree or rule learners, on the other hand, perform the generalization process as soon as the data are fed. Being lazy implies that instance-based learners would take more time than eager ones in the classification phase in which the whole process is completed. Therefore, instance-based learning might not suit our domain where the classification time is critical.

Traditional instance-based representations such as [CH67] do not employ any knowl-

edge model to explain patterns in data. All data instances are used in classifying new instances. Consequently, a large number of data might be required at classification time. To attack this problem, instead of storing a whole set of data, it is possible to store only a few prototypes of each class (i.e., the ones near the boundary between classes) [FBF77][MM99][PM00]. Instead of calculating distances from the instances' neighbors, they basically define regions that enclose instances of the same classes. These regions can be different based on the algorithms that generate them, the number of classes, or the data themselves. Generalizing instances into regions leads to another kind of knowledge representation called "clusters" which will be discussed in the next section.

In this research, we follow a popular instance-based learning approach described in [Aha97] by Aha et al. in 1997, which is derived from the classical k-NN algorithm introduced by Cover and Hart. It employs the Euclidean distance as the distance metric and the feature values are normalized to ensure that equal importance is given to the features with respect to the same distance function. Then, to classify an unseen instance, the k nearest neighbors of that instance in the dataset are determined and the majority class of the k instances is given to the unseen one. In our experiment, k is set to 10. Algorithm A.3 in Appendix A describes the approach in detail. Note that the algorithm actually describes the classification process rather than the learning process as the instance-based learner takes the lazy path and pushes all the work into the classification phase. It has been shown by Roughan et al. [RSSD04] that k-NN is very useful in a flow classification domain. They have evaluated the algorithm using several k with many trace sets and different numbers of classes. The accuracy of the system varies from 92.1% to 97.5% depending on the number of k and classes.

4.5 Clustering

All the methods that we have discussed so far are supervised learning methods aimed to predict the class of an instance with the learners learning from the given pairs of feature vectors and classes. Another breed of classification methods consist of unsupervised learning methods and is designed to be applied to domains where data instances do not belong to any predefined classes and the learners consider only the feature vectors (and not the classes of the instances in the dataset). The aim of an unsupervised learner is to determine how the data are organized, rather than to associate instances with classes. One of the most common types of unsupervised learning methods is called "clustering". Similar to the instance-based approach, an instance in clustering is thought of as a point in a d-dimensional space associating with d features. The goal of the learning is to divide the data instances into groups or "clusters", with the instances in each cluster possessing some similar properties (e.g., the feature values). How the clusters are defined and determined is different among clustering algorithms.

Although it is not originally designed for classification tasks, clustering algorithms can be used in classification problems as well. The rationale is that the instances that belong to the same class should have the similar characteristics and, thus, belong to the same clusters. To classify a new instance, it will be assigned to the nearest cluster

according to the feature values and then the class associated with that cluster will be given to the instance. Nevertheless, because the classes are not considered in the learning process, the instances that belong to different classes could still be assigned to the same cluster. If there are multiple classes in a cluster, assigning a class to the new instance could clearly become problematic. Many class-assignment heuristics have been introduced in response to this issue. Roughan et al. [RSSD04] propose to assign classes to the flows in the clusters based on specific features such as transport ports. Zander et al. [ZNA05b, ZNA05a] and Erman et al. [EMA+07], on the other hand, simply classify a flow with the dominant class of that cluster. Bernaille et al. further conduct an experiment regarding the classification heuristics in [BTS06] and conclude that classifying a flow based on its cluster and transport port is more effective than the dominant-class approach. It is important to note, however, that their results could be biased because the ground truth (i.e., the true classes of instances in the dataset) is determined by transport ports.

4.6 Discussion on Learner Selection

In addition to categorizing them into supervised and unsupervised ones as discussed above, we can also divide machine learning algorithms into white box and black box approaches. For white box approaches, which include most of the algorithms presented earlier, the classification models induced by the learner are observable, symbolic, and easy to be comprehended [Mic88][MBB98]. On the other hand, the black box learners, such as "Artificial Neural Networks (ANNs)[7]" [MP69][Wer75][Hop82][Elm90][Dav09], Random Forest [Bre01], and mathematical statistics, induce models based on their own knowledge representations, which are rather complex and difficult to interpret [MBB98][HX02]. These models typically involve weights, coefficients, and distances [MBB98].

Although neural networks are expressive and robust [WF05], the learned knowledge that is embedded in the network structure is difficult to understand and cannot be easily verified by the domain experts [HX02]. Despite many attempts to extract the comprehensible knowledge from the networks [Gal93][TS93][Fu94][HX02] and to combine connectionist systems and symbolic knowledge representation [Smo87][SA93] [BHH08], understanding the neural network structure is still a challenging research area. Another example of a black box learner is the Random Forest, which creates a number of decision trees from randomly chosen subsets of features (hence the name). A new instance will be classified by all trees and the majority class will be assigned to the instance. It is considered as a black box because there is no obvious way to extract the general knowledge from the forest [Bre01].

Because the classifier's task is essentially to map the feature vector to a service class, it is reasonable to restrict ourselves only to supervised learning algorithms. Moreover, we are also looking for algorithms whose learned classification models are easy to interpret and utilize. White-box algorithms whose learned models can be transformed into IF-THEN rules are the most preferable as they clearly present the relationships

[7] In the literature, they are also referred to in a broader term as "connectionism models".

between features and classes. To this end, we have selected four white-box, supervised algorithms to evaluate — namely, C4.5, RIPPER, PART, and Naive Bayes. In addition, k-Nearest Neighbor, which is an instance-based approach is also included. C4.5, the decision tree algorithm, follows a divide-and-conquer approach. It first examines all the features at each level of the tree and then determines which one is the most discriminative in separating the classes at their respective levels. RIPPER, on the other hand, adopts a sequential covering approach as it considers each class individually and tries to find rules that cover as many instances of that class as possible. PART is a hybrid algorithm, which makes use of both approaches. Built on Bayes theorem, the Naive Bayes classifier works by statistically classifying the flows based on background knowledge. Lastly, the k-NN algorithm classifies an instance based on its similarity (or distance) between the new instance and other instances in the dataset.

All aforementioned algorithms cover a wide range of learning algorithms, which we believe will be comprehensive enough for robust evaluations. Nevertheless, apart from the algorithms presented above, there are many other learning algorithms that have not been discussed, such as neural networks, linear/non-linear regression models, and support-vector machines. Their exclusion from our research is due to the fact that they are not intended to induce such comprehensive structures as trees or rules. Possibilities of applying them in flow classification domain will be explored in our future efforts. Machine learning literatures, e.g., [Bis08], [WF05], [TK03], [Mit97], [MBB98], and [Kot07], provide further discussions on these learners as well as other machine learning techniques.

4.7 Performance Measurement

Having discussed the learning algorithms and the classifier induction, we will now turn to the evaluation of a classifier after it is induced by a learner. Typically, in machine learning literature, the evaluation process involves two types of datasets: the "training set" and the "test set". The former is a set of instances used by the learner to induce a classifier; the latter is the dataset used to evaluate the classifier produced by the learner.

Definition 4.7 (Accuracy of a Classifier) Let $\mathcal{K}$ be a set of classifiers, $\mathfrak{D}$ the set of all datasets, and $\mathcal{C}$ a set of service class. The *accuracy of a classifier* $K \in \mathcal{K}$ *over* $\mathcal{D}$ is defined as follows:

$$\begin{aligned} \mathit{Accuracy}: \quad & \mathcal{K} \times \mathfrak{D} \to \mathbb{R} \\ & (K, \dot{\{}\langle \boldsymbol{x}_1, c_1\rangle, \ldots, \langle \boldsymbol{x}_q, c_q\rangle\dot{\}}) \mapsto \frac{\sum_{i=1}^{q} \mathit{matched}(K(\boldsymbol{x}_i), c_i)}{q}. \end{aligned}$$

Provost et al. [PFK98] suggested that using accuracy as an evaluation measure might be misleading in some cases due to the fact that classification accuracy assumes equal misclassification costs. For example, the cost of misclassifying StrConv flows as Bulk flows is the same and vice versa (both are considered as incorrect), which might not always be the case. Also, classification accuracy assumes that the classes in the dataset are equally distributed. If the classes are badly distributed, or "skewed",

however, the evaluation results could be biased. For instance, if 90% of the instances in the dataset belong to Bulk class, simply classifying all instances in the dataset as Bulk would still give 90% correctness. In our evaluation, we assume that misclassification costs are the same for all classes. Furthermore, the datasets used to evaluate the classifiers are "stratified" (i.e. the instances of all classes are drawn equally from the original datasets), which ensures that the evaluation results are not influenced by results of only any particular classes. Dataset stratification is discussed in detail in Section 5.4.2 and 5.5.3.

When the dataset is split into training and test sets, there is a chance that either of the two will not represent the actual characteristics of the data. In an extreme case, a certain class might be missing altogether from the training set. This effect could be prevented at the splitting process using stratified randomization. However, it is still very premature as it cannot guarantee that the training and test sets are really representative. This problem becomes even more pronounced in a case where the entire dataset (including both training and test sets) is small.

To address this issue, Stone [Sto77] proposes an evaluation method that repeatedly uses multiple training and test sets generated from multiple splits. This method is called "cross-validation (CV)". In cross-validation, the data are divided into k equal partitions or *folds* and each fold is held out to be used as the test set while the rest are used as the training set. A classifier is then induced from the training set and evaluated against the test set by a learner. This process is repeated over and over every fold has been used as test set (i.e., k iterations). As a result, the learner is evaluated k times over k different sets. The correctness from all iterations is then averaged to find the overall accuracy. In our evaluation, k is set to 10 as it tends to produce the best accuracy estimation, according to [Koh95]. In the following, formal description of cross-validation process will be given. We will begin with an introduction to a multiset operator, additive union, follows by definitions of folds and cross-validated performance of a classifier respectively.

Definition 4.8 (Additive Union) Given a set X, the *additive union* of two multisets M_1 and M_2 over X is a multiset M, denoted by $M = M_1 \uplus M_2$, such that $\forall x \in X$, $M(x) = M_1(x) + M_2(x)$.

Basically, additive union is a multiset operation that joins two multisets together like ordinary set union. In additive union, however, the multiplicities of similar elements in both sets are also added together. The following is the formal definition of folds:

Definition 4.9 (Folds) A *set of k-folds of a dataset* $\mathcal{D}$ is a set $\mathbb{F}_{\mathcal{D}} = \{\mathcal{D}^1, \ldots, \mathcal{D}^k\}$ such that $\mathcal{D}^1 \uplus \ldots \uplus \mathcal{D}^k = \mathcal{D}$, and $\forall i, j \in \{1, \ldots, k\}$ there exists no $\mathcal{D}^i, \mathcal{D}^j \in \mathbb{F}_{\mathcal{D}}$ such that $abs(|\mathcal{D}^i| - |\mathcal{D}^j|) > 1$. An element in $\mathbb{F}_{\mathcal{D}}$ is called *a fold of* $\mathcal{D}$.

In other words, the dataset is composed of k folds such that the differences of the cardinalities of the folds are not greater than one. The condition on the number of cardinality ensures that the sizes of the folds are as close as possible. Finally, we can formulate the cross-validation as follows.

Definition 4.10 (Cross-Validation) Let $\mathfrak{D}$ be a set of all datasets, $\mathcal{D} \in \mathfrak{D}$, $\mathbb{F}_{\mathcal{D}} = \{\mathcal{D}^1, \ldots, \mathcal{D}^k\}$ a set of k-folds of $\mathcal{D}$, and $\mathcal{L}$ a set of learners. A *k-folds cross-validation performance of L over $\mathcal{D}$ with respect to $\mathbb{F}_{\mathcal{D}}$* is defined as

$$\begin{aligned} \mathit{CrossValidate}: \quad & \mathcal{L} \times \mathfrak{D} \times \mathcal{P}(\mathfrak{D}) \to \mathbb{R} \\ & (L, \mathcal{D}, \mathbb{F}_{\mathcal{D}}) \mapsto \frac{\sum_{i=1}^{k} \mathit{Accuracy}(L(\mathcal{D} \backslash \mathcal{D}^i), \mathcal{D}^i)}{k}. \end{aligned}$$

4.8 Conclusion

In this chapter, an overview of supervised white box machine learning methods is provided. We have explained in detail how several supervised learning schemes work as well as investigated their strengths and weaknesses. As a result, five learning algorithms — C4.5, RIPPER, PART, Naive Bayes, and k-Nearest Neighbor — are selected to be used for evaluation in our research. A discussion on measuring the performance of a classifier is also given along with the formal definitions of accuracy and cross-validation measures. In the next chapter, we will see how these learners are integrated in a flow classification system and how they perform on the real-world datasets with respect to the performance measures described in this chapter.

Chapter 5

Toward an Adaptive Flow Classification System

We have learned in Chapter 2 that a real-time adaptive flow classification system that can classify both TCP and UDP flows is still missing. In this chapter, a novel flow classification system called "Supervised Machine learning Assisted Real-Time FCS (SMART)" is presented. The system will be described on the basis of the model that we introduced in Chapter 3. It is equipped with a learner and uses only features that consider only information that is available in both TCP and UDP protocols. We will see in this chapter the components of SMART including the service classes, the features, the learner, and how they are integrated. In addition, the system will be evaluated on the data from individual users.

5.1 Proposed Service Classes

The choice of service classes to be used in a FCS depends on the purpose of the system as it specifies how the flows would be categorized. Since SMART is developed to assist QoS support, the set of classes has to be able to capture the QoS requirements.

Currently, there are few standardized service classes available, for example, [G.101b] by ITU-T, [3GP04] by 3GPP, and service class guideline by IETF [BCB06]. They cannot be used in our scenario directly as they are designed to identify how the packets should be treated (based primarily on delay-sensitivities), and not to group similar services together. To this end, we propose a set of service classes, which is meant principally for flow classification systems, with each class being designed to be general enough to cover all services with similar QoS requirements as follows.

Strict Conversational Class Real-time audio/video applications are symmetric applications, which are sensitive to delay and delay variation as well as require relatively high data rates. Examples of application protocols in this class include the Real-time Transport Protocol (RTP) [SCFJ03], which delivers services such as Voice-over-IP (VoIP), Windows Live Messenger videoconference [Mic07], and real-time online games such as Half-Life [Val07] and Unreal [Epi07].

Relaxed Conversational Class This class of applications is quite similar to the previous class but requires less bandwidth, less delay variation sensitive and error intolerable. Applications in this class are real-time and symmetric with low delay requirement, including remote desktop, interactive games and instant messaging. Example applications are Telnet, SSH [Ylo06], Virtual Network Computing (VNC) [RSFWH98], Internet Relay Chat (IRC) [Oik93], and Yahoo Messenger [Yah08].

Streaming Class Streaming class services serve streams of data, including audio/video streaming services (such as RealMedia [Rea08] or web-based streaming services like YouTube [You08]). These applications expect high data rates but are not sensitive to delay or delay variation because the data can be buffered and do not need to be used in real-time.

Interactive Class All server access applications fall into this class. The key characteristic of the interactive class applications is request-and-response, i.e., the client application sends a request to a server and receives a response from the server. This kind of service is asymmetric, not delay sensitive, and does not require high bandwidth. The main requirement is error intolerance. Examples of applications include web browsers, email clients, as well as searching or file-transferring requests of peer-to-peer (P2P) software. It is worth noting that P2P software generally has two types of sessions. One is for searching and requesting files and the other is for the actual data transmission. The set of classes described here categorize the two types of sessions into two classes.

Background Class In background class traffic, the other side of the transmission does not expect the data within a certain period of time and the application will use network resources as they are available, i.e., in best-effort manner. Examples of services in this class are email delivery, SMS, and bulk data transfer such as FTP and Server Message Block (SMB) protocols as well as P2P data transmissions.

The set of applications or services of each class that are used to evaluate SMART, our new FCS, are listed in Appendix C.1. Because the components of the flow classification system do not depend on any the set of classes, the suggested set of classes could be effortlessly replaced if SMART is used for other purposes.

5.2 Features

Features are characteristics of flows that are used to categorize the flows into classes. Different feature functions are employed by different flow classification approaches as briefly discussed in Section 2.5. In this section, feature functions that are used by several existing FCSs will be explored using the model introduced in the Chapter 3. Then, we will proceed to discussions on a novel feature, throughput difference, as well as other feature functions used by SMART.

5.2.1 Features in the Literature

In this section, the characteristics and properties of the existing features in the literature are reviewed, using our model introduced in the previous chapter. The advantages and disadvantages of each kind of feature will be discussed. We will also see that our model can explain all existing features that are available in the literature.

The simplest kind of flow classification system is the port-based approach that classifies flows based solely on their transport ports. The feature functions of such systems are thus:

$$\begin{aligned} \mathit{srcPort}: \quad & \mathcal{F} \to \mathbb{N} \\ & (p_i \mid 1 \le i \le n) \mapsto \mathit{srcPort}(p_1) \\ \mathit{dstPort}: \quad & \mathcal{F} \to \mathbb{N} \\ & (p_i \mid 1 \le i \le n) \mapsto \mathit{dstPort}(p_1). \end{aligned}$$

The feature extraction function can, in turn, be defined as:

$$\begin{aligned} E': \quad & \mathcal{F} \times \mathbb{N}^+ \to \mathcal{X}_2 \\ & (f, l) \mapsto \langle \mathit{srcPort}'(f, l), \mathit{dstPort}'(f, l) \rangle \end{aligned}$$

where $\mathcal{F}$ is a set of managed flows, $\mathbb{N}^+ = \{1, 2, 3, \ldots\}$, $\mathcal{X}_2 = \mathbb{N} \times \mathbb{N}$ a set of feature vector, and *srcPort*′ and *dstPort*′ are length-restricted features corresponds to *srcPort* and *dstPort*, respectively. By definition, the source and destination ports are the same among all packets in the flow. Therefore, $l = 1$ is sufficient. Examples of such systems include [MKK+01], [LC03], and [PN97]. As discussed in Section 2.5, transport protocol ports are not reliable, hence, in our FCS, the use of transport port is avoided. Also, because of that, other techniques such as signature-based and flow-behavior based approaches are developed. Signature-based approaches consider packet and flow payload as features and classify flows according to the contents of the payloads. Hence, feature functions employed in signature-based FCSs are functions that extract payload contents from the given flows. A study on signature-based FCS including the employed feature functions and feature extraction functions is presented in Section 5.5.2.

Another kind of classification approach is called flow-behavior-based approach. It employs the features that are aimed to capture more generic characteristics of the given flows such as average packet size or throughput. One of the earliest of such FCSs, which is proposed by Zhang and Paxson [ZP00], tries to distinguish flows based on the ratios of specific packet sizes and IATs. Its feature functions can be described under our model as follows.

Let P a set of packet models and $p \in P$. We define the following auxiliary functions:

$$\begin{aligned}
\mathit{smallPkt}: \quad & P \times \mathbb{N}^+ \to \{0,1\} \\
& (p,s) \mapsto \begin{cases} 1 & \mathit{size}(p) \leq s, \\ 0 & \text{otherwise.} \end{cases} \\
\mathit{timeGap}: \quad & P \times P \times \mathbb{N}^+ \to \{0,1\} \\
& (p,p',s) \mapsto \begin{cases} 1 & \mathit{smallPkt}(p,s) \text{ - } \mathit{smallPkt}(p',s) \neq 0, \\ 0 & \text{otherwise.} \end{cases} \\
\mathit{shortIAT}: \quad & P \times P \times \mathbb{R} \to \{0,1\} \\
& (p,p',t) \mapsto \begin{cases} 1 & \mathit{iat}(p,p') \leq t, \\ 0 & \text{otherwise.} \end{cases}
\end{aligned}$$

Intuitively, the function *smallPkt* checks if the packet is larger than the given packet size. The function *timeGap* checks if the given packets are both small. *shortIAT* checks if the IAT between two packets are shorter than a given value. Then, let $\mathcal{F}$ be a set of flows, $(p_i \mid 1 \leq i \leq n) \in \mathcal{F}$, $s \in \mathbb{N}^+$, and $t \in \mathbb{R}$, we can characterize the following feature functions:

$$\begin{aligned}
\mathit{smallPktRatio}: \quad & \mathcal{F} \to \mathbb{R} \\
& (p_i \mid 1 \leq i \leq n) \\
& \mapsto \frac{\sum_{i=1}^{n} \mathit{smallPkt}(p_i,s) - \sum_{i=1}^{n-1} \mathit{timeGap}(p_i,p_{i+1},s) - 1}{n} \qquad (5.1) \\
\mathit{shortIATRatio}: \quad & \mathcal{F} \to \mathbb{R} \\
& (p_i \mid 1 \leq i \leq n) \mapsto \frac{\sum_{i=1}^{n-1} \mathit{shortIAT}(p_i,p_{i+1},t)}{n-1} \qquad (5.2)
\end{aligned}$$

Here, the feature functions compute the ratio of bursts of *small* packets against the total number of packets and the ratio of *short* IATs against all IATs respectively. The thresholds of small and short are defined as s and t, which must be manually specified to suite the target protocols or classes. Furthermore, in [ZP00], the features are calculated from all packets in the flow. Therefore, it is considered to be non-real-time approach and, in turn, the feature extraction function is defined as follows:

$$\begin{aligned}
E: \quad & \mathcal{F} \to \mathcal{X}_2 \\
& f \mapsto \langle \mathit{smallPktRatio}(f), \mathit{shortIATRatio}(f) \rangle
\end{aligned}$$

where $\mathcal{X}_2 = \mathbb{R} \times \mathbb{R}$. Then, to classify a flow, a rules set that classifies the flows based on the predefined ratio of small packets and short IATs is introduced to be used as a classifier. Similar techniques have been extended by [ML05], [DWF03], and [TAO07]. However, because the thresholds of the packet size and IAT as well as their ratios have to be predefined, these approaches are categorized as non-adaptive.

In 2004, Roughan et al. [RSSD04] introduced flow classifier systems that are equipped with different clustering-algorithms-based learners. In their paper, average packet size, root mean square of packet size, flow connection time, data volume, number of packets, and standard deviation (SD) of packet inter-arrival time are employed

as features. In addition, they also introduced another feature, the inter-arrival time variability, which can be categorized as follows:

$$\begin{aligned} iatVar: \quad & \mathcal{F} \to \mathbb{R} \\ & f \mapsto \frac{iatSD(f)}{iatAvg(f)} \end{aligned} \tag{5.3}$$

Essentially, the function states that the inter-arrival time variability is the ratio of the SD to the average of the IATs throughout the flow. This feature was shown in [RSSD04] that it could be used to separate streaming traffic and bulk-data traffic from each other. Furthermore, it was shown that the average packet size and the flow connection time can be used to distinguish relaxed conversational, interactive and streaming classes from each other. These features are also employed by Zander et al. in 2005 [ZNA05b, ZNA05a] as well as SMART [AS07a][Ana10]. In 2006, Williams et al. [WZA06] evaluated the aforementioned features along with the following: transport protocol, minimum and maximum packet size, and inter-arrival time.

All approaches mentioned above are non-real-time flow classification systems, which abstract the features from the entire flows. The flow extraction function can thus be characterized as follows:

$$\begin{aligned} E: \quad & \mathcal{F} \to \mathcal{X}_d \\ & f \mapsto \langle V_1(f), \ldots, V_d(f) \rangle \end{aligned}$$

where $\mathcal{F}$ is a managed set of flow, $V_1, \ldots, V_d$ are feature functions, and $\mathcal{X}_d$ is a set of feature vectors corresponds to $V_1, \ldots, V_d$.

Non-real-time approaches are not suitable for some domains, such as QoS support, where the classification has to be carried out in a limited time period so that the flow can benefit from the given class. Thus, Bernaille et al., [BTA+06, BTS06] have proposed a flow classification system that can classify flows in using only partial flows allowing the classification to be done after seeing only specific number of packets. In [BTA+06, BTS06], it is observed that when two hosts interact with each other using different application protocols, their communication initialization behaviors are also different. These interactions are defined over the sizes and the direction of the packets. The packet sizes of each flow and its coflow are captured and compared with respect to their orders. That is, packet size of the first packet of a flow is compared with the size of the first packet of its coflow, the sizes of second packets in both flows are then compared, and so on. After the comparison, the differences of the packet sizes of each pair are stored in a feature vector of length l where l is the number of packet pairs to be compared. The k-th component in the vector is the difference of the sizes of the k-th packet pair. To precisely characterize the feature, we first define an auxiliary function that computes the size of the specified packet:

$$\begin{aligned} sizeOfPacket: \quad & \mathcal{F} \times \mathbb{N}^+ \to \mathbb{N} \\ & ((p_i \mid 1 \le i \le n), j) \mapsto \begin{cases} size(p_j) & 1 \le j \le n, \\ 0 & j > n. \end{cases} \end{aligned}$$

The feature function is given by:

$$\begin{aligned} sizeCompare_k : \quad & \mathcal{F} \to \mathbb{R} \\ & f \mapsto \frac{sizeOfPacket(f,k)}{sizeOfPacket(coflow(f),k)} \end{aligned}$$

$$\begin{aligned} sizeCompare'_k : \quad & \mathcal{F} \times \mathbb{N}^+ \to \mathbb{R} \\ & (f,l) \mapsto sizeCompare_k(prefix(f,l)) \end{aligned}$$

Intuitively, the feature function compares the size of the last packet of the given flow with the one from its coflow. The feature extraction function can be characterized as:

$$\begin{aligned} E' : \quad & \mathcal{F} \times \mathbb{N}^+ \to \mathbb{R}^l \\ & (f,l) \mapsto \langle sizeCompare_1(f,l), \ldots, sizeCompare_l(f,l) \rangle. \end{aligned}$$

In the function *sizeOfPacket*, the case where $j > n$ occurs when the length of the flow is smaller than specified length j. Although we specify it as zero here, in the papers, the authors have not mentioned such cases.

Since the preferred flow length can be specified, this method is considered real-time. The authors also apply and evaluate a number of clustering algorithms as the learners, making the method adaptive. At any rate, using this technique, the coflow of each flow is always required. Thus, it can only be applied to TCP flows where coflows always exist and the order or the packets in both flows have to be kept in sync. Furthermore, the employed features might not be able to distinguish between the classes whose the initialization behaviors are the same but the actual characteristics are different. One example is a case where streaming data are encapsulated by HTTP protocol. This downside is also confirmed by the experiments carried out by the authors themselves. The induced classifiers exhibit performance drop in the dataset where a single application protocol conveys multiple types of services.

5.2.2 Discriminative Features

In order to find the features that can effectively distinguish flows from different classes (or, in other words, highly "discriminative" features), it is necessary to first understand the nature of the flows. The strict and relaxed conversational classes share some characteristics such as symmetry (both sides of connection communicate to each other equally) and connection time (e.g., a conversation would take longer than loading a web page) but they are different in error tolerance, data volume, and delay sensitivity. Strict conversational flows consume much more bandwidth, are tolerable to errors, and are extremely sensitive to delay, while the relaxed conversational class behaves diametrically. Therefore, we believe that the transport protocol, data volume, and packet inter-arrival time (IAT) could be the key differences between them. Streaming connection is typically carried out in a request-and-response manner. Once the connection is established, the client does not send anything back to the server apart from TCP packet acknowledgements. This is also true for the interactive class. We

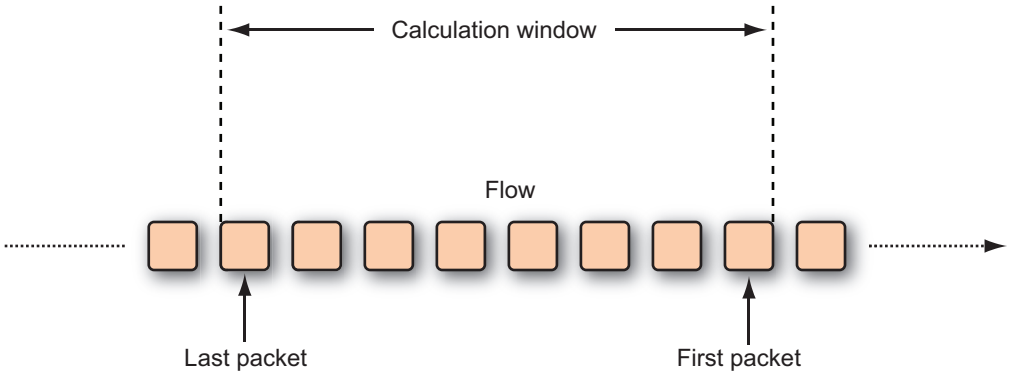


Figure 5.1: Throughput calculation window. The width of a calculation window is the number of packets used to calculate the throughput at a point in time. The first throughput will be calculated using packets 1 to ω. The second throughput will be calculated using packet 2 to $\omega + 1$, and so on. While $\omega = 8$ in this figure, ω is set to 10 in the actual implementation.

can thus use this asymmetry factor to separate streaming and interactive classes from the conversational ones. The symmetry might be captured by ratio of data volume transferred by a flow to that of its coflow.

The streaming and interactive classes can then be distinguished from each other by data volume and burstiness. In the next section, a new feature, which is designed to capture burstiness of a flow will be described.

5.2.3 Throughput Difference — A New Feature

Flow burstiness characterizes how uniform the packet IAT of the flow is. Packet IATs of streaming flows, which transfer data in streams, would be more stable than those of interactive flows, which have to wait for interaction from the other sides of the transmissions. To capture such characteristic, we employ a new feature function called *TPUTDiff*, which captures the changes of the throughput along the flow[1]. Although throughput is affected by both IATs and packet sizes, our experiments show that the feature can be effectively used to distinguish the flows.

Throughput is calculated by dividing the size of the data by the time duration that the data are transferred. Generally, it is done by fixing a time-window and the amount of data transferred within the window is divided by the window width. In doing so, however, a timer is required for each flow[2]. Instead, we take another approach to calculate throughout. When a flow is seen, the classification system stores the size and the timestamp of each packet up to a specific number of packets, say ω. The throughput is calculated by dividing the sum of the packet sizes with the differences between the timestamps of the last (i.e., the ω-th) and the first packets. Interval between first and last packets is called a "calculation window". By moving the window through the flow, the sum of the differences of the throughput along the flow can be obtained. Figure 5.1 illustrates the calculation window.

[1]This work has already been reported in [AS07a].

[2]Also, because WinPcap triggers the flow capturing component every time a packet is seen, calculating this way is more suitable than maintaining the timer.

Let *dataTPUTAvg* and *packetCount* be functions that calculate average data throughput and packet counts of a flow, respectively[3]. The following is the definition of the corresponding auxiliary function, *sumTPUTDiff*, which calculates the sum of differences of throughput along the flow, and the throughput difference feature function.

$$\begin{aligned}
\mathit{sumTPUTDiff}: \quad & \mathcal{F} \times \mathbb{N}^+ \to \mathbb{R} \\
& ((p_1, \ldots, p_n), \omega) \mapsto \begin{cases} |\mathit{dataTPUTAvg}((p_1, \ldots, p_\omega)) \\ \quad -\mathit{dataTPUTAvg}((p_2, \ldots, p_{\omega+1}))| \\ \quad +\mathit{sumTPUTDiff}((p_2, \ldots, p_n)) & n \geq \omega + 1, \\ 0 & \text{otherwise.} \end{cases} \\
\mathit{TPUTDiff}_\omega: \quad & \mathcal{F} \to \mathbb{R} \\
& f \mapsto \frac{\mathit{sumTPUTDiff}(f, \omega)}{\mathit{packetCount}(f)}
\end{aligned}$$

Intuitively, throughput of a flow is calculated using a moving window of ω packets. To ensure accurate throughput computation in our experiments, we use $\omega = 10$. After at least two windows are obtained, the difference of throughputs of two adjacent windows are computed and summed. In effect, to compute the differences, at least $\omega + 1$ packets are required. Also, the sum of the differences is affected by the overall number of packets (i.e., the flow length) and, therefore, has to be normalized by the flow length.

Apart from high discriminative power, the designate features should not require deep-packet information (i.e., the payload data) and calibration. Some features, such as [ZP00], [ML05], [DWF03], and [TAO07], require some calibrations or fine-tuning by experts. It thus begs the question whether such features can be used in general or whether the induced classifier would be able to handle unseen services. In consequence, they will not be considered here. For $\mathit{TPUTDiff}_\omega$, although the size of the calculation window ω has to be identified beforehand, it signifies only how the throughput is measured but does not directly influence the classification. Features that are not well-defined such as maximum or minimum throughput are also avoided. This is because they require additional parameters, such as the interval that the throughput is observed. As a result, a set of 32 features shown in Table 5.1 is used. Some of these features have already been employed or introduced in the literature. Precise definitions of all feature functions in this thesis are given in Figure B.1 in Appendix B.

In this chapter, to empirically prove that SMART is feasible, preliminary experiments will be conducted using the entire set of features. In Chapter 6, further scrutiny will be on the discriminability of each feature. We will also see how the features behave with respect to the number of packets observed. The investigation would lead to an efficient, real-time and accurate flow classification system.

5.3 Learners

As discussed in the previous chapter, supervised learners are suitable for flow classification problem because the set of classes is distinctively defined and the training

[3]Precise definitions of these functions are defined in Figure B.1 in Appendix B.

Table 5.1: Descriptions of features.

	Features	Description
1	*protocol*	transport protocol of the flow
2	*srcPort*	source port
3	*dstPort*	destination port
4	*connTime*	flow run time (in seconds)
5	*connTimeCF*	connTime of the coflow
6	*dataVolume*	sum of the sizes of all packets in the flow
7	*dataVolumeCF*	dataVolume of the coflow
8	*dataVolumeRatio*	ratio between the data volume of the flow and its coflow
9	*pktCount*	number of packets in the flow (flow length)
10	*pktCountCF*	number of packets in the coflow
11	*pktCountTotal*	total number of packets in the flow and its coflow
12	*pktCountRatio*	ratio between number of packets in the flow and its coflow
13	*pktSizeAvg*	average packet size
14	*pktSizeAvgCF*	pktSizeAvg of the coflow
15	*TPUTDiff*	sum of differences of throughputs along the flow
16	*TPUTDiffCF*	TPUTDiff of the coflow
17	*pktSizeSD*	standard deviation of the sizes of packets in the flow
18	*pktSizeSDCF*	pktSizeSD of the coflow
19	*pktSizeRMS*	root mean square of the sizes of packets in the flow
20	*pktSizeRMSCF*	pktSizeRMS of the coflow
21	*dataTPUTAvg*	average data throughput (data rate)
22	*dataTPUTAvgCF*	dataTPUTAvg of the coflow
23	*pktTPUTAvg*	average packet throughput (packet rate)
24	*pktTPUTAvgCF*	pktTPUTAvg of the coflow
25	*iatAvg*	average packet inter-arrival time
26	*iatAvgCF*	iatAvg of the coflow
27	*iatSD*	standard deviation packet inter-arrival time
28	*iatSDCF*	iatSD of the coflow
29	*iatRMS*	root mean square packet inter-arrival time
30	*iatRMSCF*	iatRMS of the coflow
31	*iatVar*	ratio of the iatSD and iatAvg
32	*iatVarCF*	iatVar of the coflow

data that are fed into the learner come with service classes. In turn, our research is focused on evaluating several supervised learners including C4.5, RIPPER, PART, Naive Bayes, and k-NN. These learners represent a wide range of learners based on different disciplines and should be able to show how well our flow classification methodology is performed across different learners.

5.4 Empirical Evaluation - Individual Users

In this section, empirical evaluations of our flow classification methodology will be presented. The aim is to see if adaptive flow classification that uses the proposed service classes, features, and machine learning techniques, is feasible in end-user devices. This would allow us to see how our methodology performs when it is used to classify flows from single user. Also, we would like to know if the classifier induced by observing flows from a user can be effectively used to classify flows from another user. We will first begin with the data collection and preparation, followed by a discussion over the evaluation strategy, and evaluation results respectively. The experiments and results presented in this section have been previously published in [AS07b] and [AS07a].

5.4.1 Data Collection with Live-Capturing Sensor

To investigate the feasibility of our classification methodology as well as finding the most suitable learner for the flow classification domain, a thorough analysis of real-world data is carried out. In doing so, we have developed two state-of-the-art data collection programs; one is designed to be deployed on end-user devices and collect live traffic data from individual users and the other is intended to analyze flow data in large offline packet traces[4]. With two diverse classes of flow data, we can investigate the possibility of utilizing our methodology in both end-user devices and in other network devices such as routers and firewalls. We call the implementations of live-packet and packet-trace data collection platforms "FlowStatLive" and "FlowStatTrace", respectively. In the following, our live-capturing software, FlowStatLive, will be discussed in detail. FlowStatTrace will be described later in Section 5.5.1.

The packet sensor of FlowStatLive is implemented over an open source packet capturing software called WinPcap. WinPcap consists of two main components, the "capturing driver" and the "capturing interface". The capturing driver is implemented as an ordinary network driver through which the applications can interact with the network interfaces. However, unlike a typical network driver, when the application sends or receives packets through WinPcap driver, it also copies the packets and sends them to any programs that are waiting (one of which, in our case, is FlowStatLive) through the capturing interface. Every time when a packet is seen, the sensor will be invoked. Then, the sensor will determine if the packet belongs to any flow that it has seen before. If so, the packet will be added to a temporary packet buffer associating

[4]A packet trace is a large collection of flow data, which is usually collected at the router of large network such as organizational or ISP gateways. Packet traces will be discussed further in Section 5.5.1.

with the flow that corresponds to the packet; otherwise, a new buffer, which is a linked-list containing the packet structure, will be created. Although WinPcap provides an excellent capturing facility, it treats the packet individually regardless of which flow it belongs to. This means that FlowStatLive has to sort the packet into flows by itself.

In this sense, each buffer represents a flow that is seen over the network interface. The packet capturing and buffering process is shown in Figure 5.2. After a flow is closed or if no packet belongs to the flow observed within the time period, the flow will be considered closed and the feature extractor will be called to further process the packets in the buffer. In the literature, this timeout period is typically set to 60 seconds, which we also adapt to our non-real-time FCS implementation.

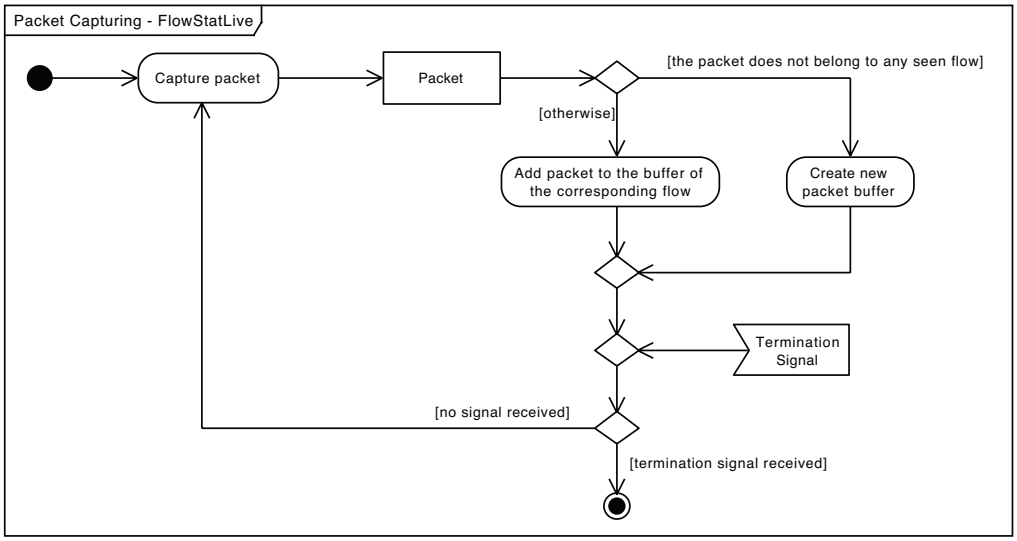


Figure 5.2: Packet-capturing process.

After the flow is closed or the timeout is reached, the extractor will invoke feature functions to abstract the corresponding feature values. All feature functions are inherited from the same feature function class, which takes a flow (i.e. linked-list of packet structures) as input and gives a feature value as output. For the extractor, each feature function is treated as a black box, as it is concerned only with the output type of the function regardless of how the output is computed. This approach closely follows the models of the feature function and extraction function defined in Definition 3.7 and 3.10. After all of them have completed the calculations, the extractor gathers the feature values and stores them in the dataset, which, in our implementation, is a comma-separated values (CSV) file. One can also store raw packet data directly without converting them to feature vectors. Doing so, however, is impractical here because it would take too much space as all network traffic on the device will be recorded. At any rate, the entire process from capturing the packets to storing the feature vectors into the dataset is carried out in real-time without noticeably slowing the system down.

Table 5.2: Dataset descriptions. Although the user's applications are summarized as, for instance, "web browsing" or "online games", each user might have different applications of the same type.

Users	User's applications	Number of flows
User 1	Mainly web browsing followed by real-time online games, video streaming, videoconference and chat	9364
User 2	Mainly real-time online games followed by web browsing, chat and streaming respectively	7928
User 3	Mainly web browsing, followed by video streaming, chat, audio conference, and online games	8992
User 4	Mainly web browsing, stock ticker, and video streaming, few online games.	8191

5.4.2 Data Preparation

To see how SMART performs in the real-world, the dataset used to evaluate the system must be collected from real-world networks and must be of a certain degree of diversity. In consequence, to capture real-world traffic and network usage, we have deployed our data collection software in different devices from different users, ranging from a high school student, whose main applications include online gaming and media streaming, to an investor who primarily uses stock ticker and web browsing applications. (See Table 5.2.) This diversity ensures that the applications, hardware and accessed networks of these users are different. Data collection consists of two main phases, flow capturing and service class identification.

In the flow capturing phrase, the flows data are collected by letting the users run their usual applications. FlowStatLive, which runs in the background, will capture the flows and extract the feature vectors automatically. Along with flow characteristics computed by the feature functions, the name of the application associated with each flow is stored in the feature vector as well[5]. In effect, the application associated to the flow is also counted as a feature. This way, we are always able to map feature vectors to the corresponding applications. This mapping is important to identify the true class of the flows as described below. Moreover, the flow observation time is restricted to 30 seconds, primarily because we want to save the system memory at the runtime. Another reason is that we would like to see if the flows can be classified without observing them entirely.

After the feature vectors are extracted, the service classes have to be assigned to them. Currently, there exists no QoS-aware application that specifies flow service classes. The networks are also not always guaranteed to support QoS. Furthermore, service class definitions are not standardized, making the information available in Type-of-Service field unreliable. Without knowing the true classes of the flows in the dataset, we cannot reliably evaluate our flow classification method. Therefore, the classes have to be assigned to the feature vectors by some other means — this is where

[5]The application name is obtained from the application-to-5-tuple mapping provided by the operating system.

the application names come into play. As the application names are stored as a feature in the feature vectors, one can identify the flow types based on the corresponding applications. Some applications can nevertheless open many classes of flows. For example, MSN Messenger can open both chatting flows and videoconference flows. To provide additional information for the identification process, after the user run a network application, she is asked to specify the class of her *activities*. The user is not expected to have background knowledge in networking or flow classification. She has to specify only what kind of actions she is doing, e.g., browsing web, playing games, or videoconferencing. These actions are considered, along with the application names, to identify the actual classes of flows. The identification is done manually by a human expert. Using activity-mapping, we can establish the ground truth (i.e., the true class of each flow in the dataset) to evaluate our classification method.

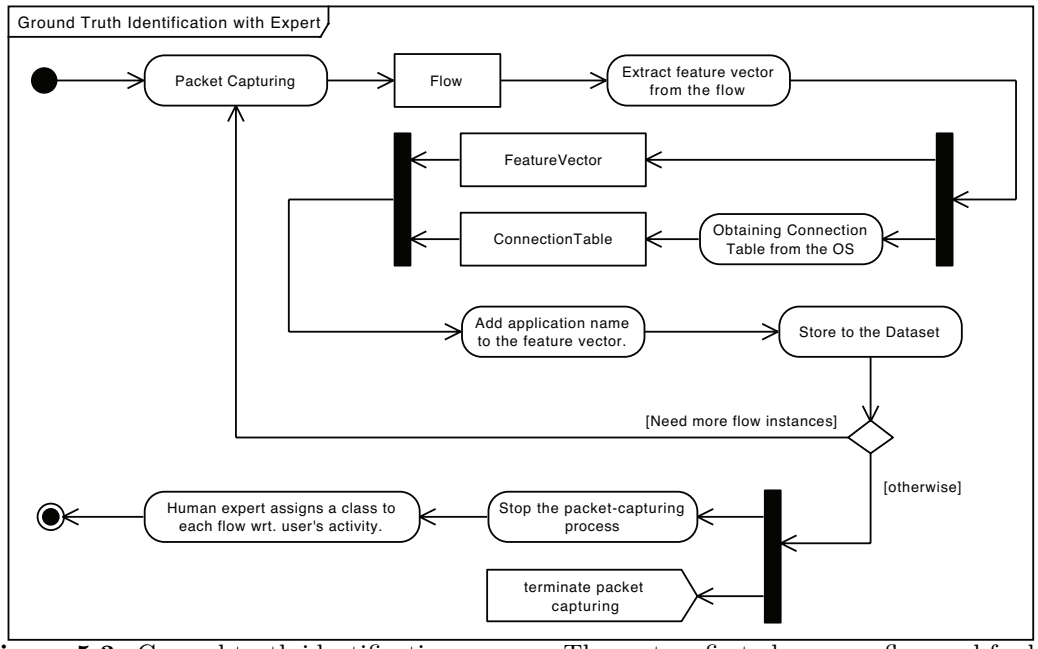


Figure 5.3: Ground truth identification process. The system first observes a flow and feed it to the extractor to extract a feature vector. The connection table is then from the operating system. The name of the application corresponding to the flow is then attached to the feature vector. After enough flows are collected, they are later classified by the human expert. The expert uses information on activities provided by the users and the application names in the feature vectors to classify each vector in the dataset.

The aforementioned service classes identification method is at least as reliable as the signature-based approach, which uses signatures to identify the service, and, based on the identified service, identifies the class of the flows. Our approach, on the other hand, uses connection-mapping tables provided by the operating system. This ensures that the mapping be error-free, unlike the signature-based approach, whose classification accuracy depends on the quality of the signatures.

5.4.3 Evaluation Strategy

This section presents the evaluations of the learners described earlier in Chapter 4, namely C4.5 decision tree learner, PART and RIPPER covering algorithms, Naive Bayes, and k-NN. The experiments are conducted on WEKA, an open source data mining platform [HFH+09]. In WEKA, C4.5 is reimplemented in Java. This Java implementation is called J4.8. The evaluations are carried out in two main phases: single-user and cross-user. In the single-user phase, the learners are applied to the dataset from each user to learn the flow behaviors of that user and use the learned knowledge to classify unseen flows from the same user. The accuracy is evaluated using 10-fold CV method. To avoid any biases, the 10-fold CV method is repeated 10 times resulting in a total of 100 individual tests. In the cross-user phase, each learner is trained using the entire data from one user and is evaluated on the data from the others. For individual user datasets, Bulk class flows are not considered because most of the applications in this class run in the background and, thus, the users are not aware of them. Therefore, we cannot be certain which flow instances belong to the class. It is important to note that none of the existing flow classification systems are aimed for end-user devices and, to the best of our knowledge, none has ever collected and performed experiments on individual-user flow data as we have done here.

5.4.4 Single-User Evaluation Results

Table 5.3 reports the evaluation results of the first phase where learners are trained and tested by the data from the same user using the cross-validation method. Despite the diversity of the users, all learners perform significantly well on every dataset, with PART performing especially better than other learners in most cases. Table 5.4 - 5.8 present per-class correctness of each learner. The results show that the learners perform equally well in all classes across datasets, and the overall correctness is not biased by correctness of any particular classes. Note that User 4 never used strict conversational flows and hence the result for that class is missing.

Table 5.3: Classification accuracy of each learner on each dataset.

Users	J4.8	PART	RIPPER	Naive Bayes	k-NN
User 1	98.73	98.54	98.46	93.37	98.25
User 2	97.14	97.98	97.56	89.52	96.47
User 3	99.31	99.43	99.52	97.73	99.09
User 4	99.11	99.33	99.32	91.98	98.52
Average correctness	98.57	98.82	98.72	93.15	98.08

We have also evaluated the learners in terms of computational time required to induce classifiers (i.e., learning time). In our observations, the classification times of all learners are extremely low and not significantly different. Thus, they are not considered here. The tests are performed on a 2.8 GHz Intel Core 2 Duo with 2 GB of RAM using Mac OS X 10.5 as the operating system. Figure 5.4 and Table 5.9 show the average learning time of each learner on each of the datasets. Although there is

Table 5.4: Per class accuracy - J4.8

Class	User 1	User 2	User 3	User 4
StrConv	99.10	99.56	100.00	N/A
RlxConv	99.18	98.20	98.41	72.83
Streaming	97.55	95.03	98.17	99.53
Interactive	99.56	98.14	99.86	99.61

Table 5.5: Per class accuracy - RIPPER

Class	User 1	User 2	User 3	User 4
StrConv	98.88	99.87	100.00	N/A
RlxConv	98.64	98.44	96.83	75.00
Streaming	97.73	95.38	98.57	99.78
Interactive	99.50	97.67	99.76	99.46

Table 5.6: Per class accuracy - PART

Class	User 1	User 2	User 3	User 4
StrConv	100.00	99.87	100.00	N/A
RlxConv	98.37	98.55	96.83	73.91
Streaming	97.38	96.08	97.88	99.85
Interactive	99.43	97.50	99.75	99.61

Table 5.7: Per class accuracy - Naive Bayes

Class	User 1	User 2	User 3	User 4
StrConv	92.81	88.44	98.77	N/A
RlxConv	95.65	98.49	98.41	82.61
Streaming	94.29	85.71	93.19	95.08
Interactive	97.44	81.79	98.49	89.60

Table 5.8: Per class accuracy - k-Nearest Neighbor

Class	User 1	User 2	User 3	User 4
StrConv	97.75	98.99	100.00	N/A
RlxConv	91.58	97.86	98.41	72.83
Streaming	96.21	95.12	96.58	98.28
Interactive	99.17	95.05	99.63	99.30

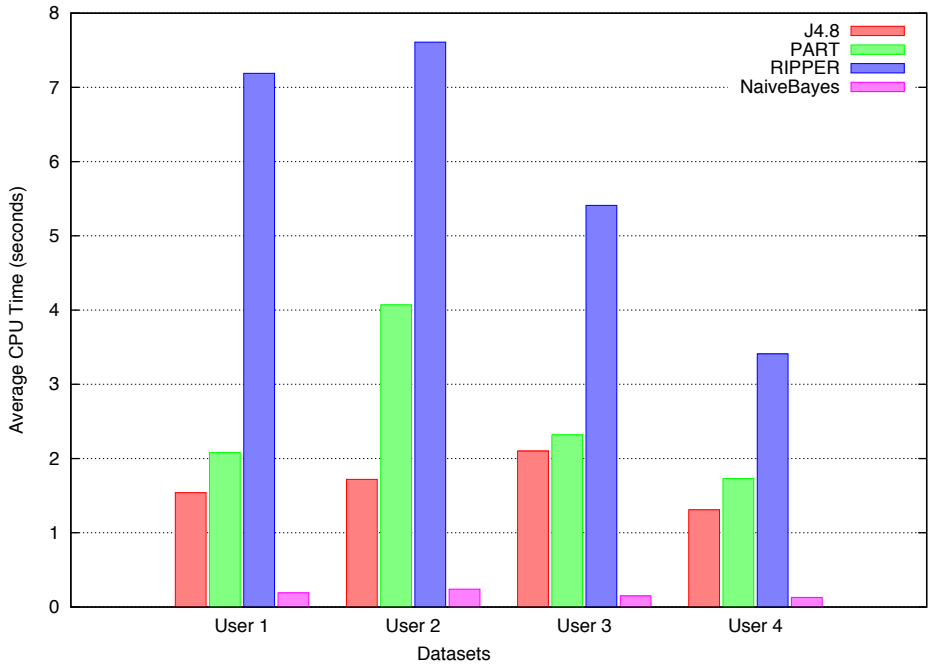


Figure 5.4: Average CPU time taken to learn. Because k-NN is a lazy algorithm, it does not construct any knowledge representation model and requires no learning time.

Table 5.9: Average CPU time taken to learn (in seconds).

Users	J4.8	PART	RIPPER	Naive Bayes
User 1	1.54	2.08	7.19	0.19
User 2	1.72	4.07	7.61	0.24
User 3	2.10	2.32	5.41	0.15
User 4	1.31	1.73	3.41	0.13

little variation in terms of classification accuracies, as seen from Table 5.4 - 5.7, huge differences in learning time are observed here. Naive Bayes learns considerably faster than the other learners on all datasets, as it simply collects the flow characteristics. Conversely, because it has to construct and revises the learned rules, RIPPER requires at least twice as much time as the other learners. This is due to the fact that RIPPER employs a slow grow-and-prune technique to generate the rule set and, after the rule set is induced, each rule in the set has to be revised again [WF05]. Naive Bayes, on the other hand, performs only simple calculations to establish the likelihood of feature values and classes. Considering the computational time alone, Naive Bayes would clearly be the preferred choice, even though its prediction accuracies are still inadequate. In comparison the other learners, J4.8 provides the best trade-off between accuracy and computational time.

In any case, the similar prediction results from all learners indicate that the features employed by our methodology are discriminative. It is thus safe to conclude that our methodology is independent not only of the applications and networks, but also of the machine learning algorithms.

5.4.5 Cross-User Evaluation Results

In the second evaluation phase, we use the data from one user as the training set and data from the other users as test sets. Each learner is trained by a dataset and then tested on all other datasets. Other combinations of the data are then selected and the process repeated until the each dataset has been used as the training set. The overall performance is the average correctness of all iterations. Table 5.10 reports the evaluation results of the four learners. It can be seen that the performance of each learner drops dramatically compared to the previous results. This may be attributed to the diversity of experience learned from the training set. Datasets collected from users who have seen many kinds of applications from different classes allow for a better classification than other datasets that contain less variety of applications. User 2, for instance, regularly plays online games as well as using other applications such as streaming. A learner using the data from User 2 can better classify other datasets than that using the data from User 4, who usually uses only web browsers. This phenomenon is even more evident if we examine deeper in dataset-wise classification results. As shown in Table 5.11 to 5.15, the classifiers that are induced from the dataset from User 1 and User 2 always perform well on other datasets, while the classifiers induced from User 3 and User 4 datasets do not. It can be observed that the classifiers induced from User 4 perform the worst because User 4 uses only a limited set of applications and does not have any StrConv applications. Notice that J4.8 performs better than other learners in most of the tests. This may thus lead us to conclude that J4.8 is more tolerable to data diversity, whereas Naive Bayes and k-NN do not.

Another reason for the low cross-user performance is that the classification rules are too specific. The flows of the same class from different users, in spite of their similar behaviors, might not be exactly the same. For example, consider the following rule obtained by training PART algorithm on User 1's data.

```
proto = TCP AND
```

Table 5.10: Cross-user evaluation result - Average correctness of all datasets.

Learners	Average Correctness
J4.8	74.15
RIPPER	72.92
PART	71.33
Naive Bayes	68.98
k-NN	66.82

```
avg_pkt_size_in > 315 AND
avg_pkt_size_in <= 1174: Interactive
```

In this case, if a flow of class Interactive has the average packet size of 1,200 and it is transferred with TCP protocol, it will be misclassified as the other class. Apparently, the rule is too strong. We believe that instead of using exact real values, transforming those values into discrete ranges of values would yield more generalized rules and thus better classification results. This issue will be investigated in our future efforts.

At any rate, the correctness of each learner in cross-user evaluation is still in an acceptable level and could still be useful. Consider a scenario where the learner on a network device is not yet well trained. One might be able to use the knowledge from another device as background knowledge until the learner is sufficiently trained.

5.5 Empirical Evaluation - Packet Traces

The next phase of our experiment is to evaluate our flow classification technique on a large dataset that is collected from a sizable network. This will allow us to see its feasibility and performance in a large domain with many kinds of applications and numerous network clients. We will begin by the description of packet traces, in which the traffic records are stored, followed by our data collection software, FlowStatTrace, as well as the experiment strategy and results, respectively.

5.5.1 The Packet Traces

In networking research, records of live network traffics are generally stored in the "packet traces". A packet trace is a collection of packet records. Depending on its format, a trace could store packet headers, payloads, other information such as the packet size, or the time when the packet is captured (i.e., the "timestamp"). Currently, there are several packet trace formats available, for instance, pcap [MLJ07], Endace Extensible Record Format (ERF) [End04], and CoralReef trace format [KMK+01]. These formats possess different features such as how much information can be stored and how precise the records could be. Despite the differences, in principle, they are just long records of packets that are captured by a sensor. That is, like in live-capture scenario, the packets are treated and stored individually and one has to sort them into flows manually. Some trace format, e.g., pcap format, can be use to *replay* the network traffic exactly the same way it is captured. In fact, FlowStatLivecan be

Table 5.11: Cross-user evaluation result (by set) - J4.8

	Tested on			
Trained by	User 1	User 2	User 3	User 4
User 1		98.42	65.10	90.94
User 2	94.54		72.30	43.37
User 3	73.16	79.03		43.05
User 4	82.12	84.36	63.64	

Table 5.12: Cross-user evaluation result (by set) - RIPPER

	Tested on			
Trained by	User 1	User 2	User 3	User 4
User 1		98.27	71.63	83.04
User 2	93.69		81.35	36.56
User 3	74.40	78.87		41.26
User 4	85.28	84.73	46.00	

Table 5.13: Cross-user evaluation result (by set) - PART

	Tested on			
Trained by	User 1	User 2	User 3	User 4
User 1		92.98	38.76	86.00
User 2	94.05		61.84	36.70
User 3	71.57	79.61		47.11
User 4	88.11	84.93	74.31	

Table 5.14: Cross-user evaluation result (by set) - Naive Bayes

	Tested on			
Trained by	User 1	User 2	User 3	User 4
User 1		76.80	61.68	64.14
User 2	92.40		76.41	59.28
User 3	68.61	74.82		55.62
User 4	80.57	78.44	39.03	

Table 5.15: Cross-user evaluation result (by set) - k-Nearest Neighbor

	Tested on			
Trained by	User 1	User 2	User 3	User 4
User 1		75.51	78.49	77.58
User 2	89.19		72.97	77.73
User 3	78.76	44.25		58.90
User 4	53.17	43.22	52.03	

modified to read the packet traces and can be further used in our experiments without much effort. However, with offline records, we do not have restrictions on memory, storage, and time to process the feature vectors. We can in turn analyze the flow characteristics in broader aspects. For example, one can extract flows using different prefixes or different set of features and determine the relationships among them as well as the classification accuracy[6]. To enable such thorough investigation on flow behaviors, a sophisticated flow analyzing software, FlowStatTrace, is implemented. In the following sections, detailed descriptions of FlowStatTrace and its components will be provided.

As mentioned earlier, various trace formats are currently available. Regardless of the format in which the packets are stored, they are in a sense huge sequences of packets that are transferred across the networks. A packet trace can thus be modeled as a sequence of packet models as follows.

Definition 5.1 (Packet Trace) Let P be a set of packet models, a *packet trace* is a finite sequence

$$(p_1, \ldots, p_n)$$

where $(\forall 1 \leq i \leq n) p_i \in P$.

The definition above indicates that the packet traces are simply sequences of *any* packets, unlike flows, which are sequences of packets with the same 5-tuples values. It is modeled as a sequence because the packets in the trace always come in order. They are generally sorted by the timestamps.

Storing the packet sequences means that entire network traffic has to be stored and that huge data storage space for each user is required. Therefore, collecting raw flow data from individual users is not practical (although this is possible through FlowStatLive). In our experiments, we use packet traces that are already available. We use the traces obtained from the Widely Integrated Distributed Environment (WIDE) traffic archive [Cho08], which contains partial packet payloads. The advantage of payload-traces is that we can use a signature-based classification system to precisely identify the flows service classes. This can be used as the ground truth to evaluate our flow classification system. The traces from the WIDE project are captured in March 2008 in both directions on a 150 Megabit per second Ethernet external link, which connects WIDE backbone and its upstream. They contain the first 96 bytes of every packet's payload. The whole traces are captured in the course of 72 hours from March 18 - 20. For each day, we selected five two-hour traces from different time periods, namely, 0:00-02:00, 08:00-10:00, 12:00-14:00, 16:00-18:00, and 20:00-22:00. In total, we have 30 hours of packet records of real-world traffic with more than 900 GB of data. Characteristics of the merged traces are summarized in Table 5.16 and 5.17. More detailed information on the traces is shown in Table D.1 in Appendix D.

5.5.2 Finding Ground-Truth Using Signature-Based FCS

As discussed in Section 3.3, the construction of a dataset requires feature vectors and corresponding service classes. Feature vectors can be extracted using information avail-

[6]These issues are important to real-time classification and will be investigated in Chapter 6.

Table 5.16: Ratio of IPv4 and IPv6 protocols in WIDE traces

Total Data Volume (both IP protocols)	912.41 GB
Percentage of IPv4 Data	99.89%
Percentage of IPv6 Data	0.11%
Total Packets (both IP protocols)	1,609,056,617
Percentage of IPv4 Packets	99.69%
Percentage of IPv6 Packets	0.31%

Table 5.17: Statistics of WIDE traces - Amount of packets and data of each trace.

Traces	IPv4 Data (MB)	IPv4 Packets	IPv6 Data (MB)	IPv6 Packets
March 18	338,849.84	581,273,033	301.39	1,427,508
March 19	318,845.56	533,481,966	382.28	1,395,621
March 20	276,612.22	489,262,589	366.55	2,215,900
Total	**934,307.62**	**1,604,017,588**	**1,050.21**	**5,039,029**

able in packet traces. This is not the case, however, for the classes of flows because QoS support is not yet well-established and most of the network applications are still QoS-unaware applications. Furthermore, service class definitions are not standardized and therefore we cannot rely on the information available in the Type-of-Service field in the IP header. In the individual-use case, we determine the applications to which the flows belong using the connection table provided by the operating system. With packet traces recorded outside of network devices, such luxury is not available, leading to the development of an entire signature-based flow classification system called PacketExtract.

Given a packet trace and a set of signatures, PacketExtract searches for the packets that contain one of the given signatures. Then, if a packet is found, all other packets in the same flow as well as its the coflow will be separated from the trace and stored in another trace. PacketExtract therefore does not only classify flows but also extracts the flows from the packet trace and stores them into a designated file. This process is iterated until all packets in the trace are analyzed. To differentiate flows, signatures of an interested service class are given to the program along with a packet trace. As a result, the target trace file will contain only the packets of the same class. The mathematical model of PacketExtract is presented below.

Recall that packet payload is defined as a string over hexadecimal alphabets. Let

Σ^* be a set of strings, we first define two auxiliary functions that manipulate strings:

$$\begin{aligned} \mathit{concat}: \quad & \Sigma^* \times \Sigma^* \to \Sigma^* \\ & (b_1 \dots b_n, c_1 \dots c_m) \mapsto \begin{cases} b_1 \dots b_n c_1 \dots c_m & n > 0, m > 0, \\ b_1 \dots b_n & n > 0, m = 0, \\ c_1 \dots c_m & n = 0, m > 0, \\ \epsilon & n = 0, m = 0. \end{cases} \end{aligned} \tag{5.4}$$

$$\begin{aligned} \mathit{substring}: \quad & \Sigma^* \times \Sigma^* \to \{0, 1\} \\ & (b_1 \dots b_n, c_1 \dots c_m) \\ & \mapsto \begin{cases} 1 & (\forall i \in \{1, \dots, n\})(\exists j \in \{1, \dots, m\}) b_i = c_{j+(i-1)}, \\ 0 & \text{otherwise.} \end{cases} \end{aligned} \tag{5.5}$$

The function *concat* concatenates two strings together. For example, the concatenation of two following strings, 48 54 54 50 and 2F 31 2E , is

$$\mathit{concat}(48\ 54\ 54\ 50, 2F\ 31\ 2E) = 48\ 54\ 54\ 50\ 2F\ 31\ 2E.$$

The function *substring* checks whether the first string is contained in the second string. For instance:

$$\mathit{substring}(2F\ 31\ 2E, 48\ 54\ 54\ 50) = 0,$$

whereas

$$\mathit{substring}(54\ 50, 48\ 54\ 54\ 50) = 1.$$

Then, let $\mathcal{F}$ be a set of flows and $(p_i \mid 1 \leq i \leq n) \in \mathcal{F}$ a flow. The payload-extraction feature is a function defined as follows:

$$\begin{aligned} \mathit{flowPayload}: \quad & \mathcal{F} \to \Sigma^* \\ & (p_i \mid 1 \leq i \leq n) \\ & \mapsto \begin{cases} \mathit{concat}(\mathit{payload}(p_1), \mathit{flowPayload}((p_j \mid 2 \leq j \leq n))) & n > 1, \\ \mathit{payload}(p_1) & n = 1. \end{cases} \end{aligned}$$

In other words, the function *flowPayload* extracts and concatenates the payloads in all packets in the flow. Consequently, the length-restricted payload-extraction feature can be characterized by:

$$\begin{aligned} \mathit{flowPayload}': \quad & \mathcal{F} \times \mathbb{N}^+ \to \Sigma^* \\ & (f, l) \mapsto \mathit{flowPayload}(\mathit{prefix}(f, l)) \end{aligned}$$

Let $\mathcal{X}_1 = \Sigma^*$ be a set of feature vectors. The feature-extraction function corresponding to the payload-extraction feature is:

$$\begin{aligned} E': \quad & \mathcal{F} \times \mathbb{N}^+ \to \mathcal{X}_1 \\ & (f, l) \mapsto \langle \mathit{flowPayload}(f, l) \rangle \end{aligned}$$

The signature-based classification considers only the signature. Therefore, the feature vector has only one element. Generally, signature-based classification systems operate in real-time as the number of considered packets can be specified beforehand.

Definition 5.2 (Signature of a Class) Let $\mathcal{C} = \{c_1, \ldots, c_k\}$ be a set of service classes and Σ^+ a set of non-empty strings. Elements of $\mathcal{C} \times \Sigma^+$ are called *signatures pairs*. A *set of signature pairs* is denoted by $SIG \subseteq \mathcal{C} \times \Sigma^+$. Let $(c, s) \in SIG$, s is called a *signature of* c.

Definition 5.3 (Signature-Based Classifier) Let $\mathcal{X}_1 = \Sigma^*$ be a set of feature vectors, SIG a set of signature pairs, and $s \in \Sigma^*$. A *signature-based classifier* is a function defined as follows:

$$K_{SIG}: \quad \mathcal{X}_1 \to \mathcal{C}$$
$$(\langle s \rangle) \mapsto \begin{cases} c' & (\exists (c', s') \in SIG) substring(s', s) \\ c_{\text{default}} & \text{otherwise.} \end{cases}$$

Intuitively, the classifier searches through the payload for each signature in SIG. If a signature is found, i.e., there exists a substring in the payload that matches the signature, the classifier returns the class corresponding to the signature. Otherwise, it returns a pre-defined *default* class. Notice that the classifier is bound to a predefined set of signatures. In the following, a simple flow classification scenario using our signature-based FCS is presented.

Example 5.4 (Signature-Based Flow Classification System)
Consider the following set of service classes from (3.1). The following is an example of a set of signature pairs corresponding to those classes.

$$\begin{aligned} SIG \;=\; & \{(StrConv, \text{4A 00 14 01}), \\ & (StrConv, \text{FF FF FF FF}), \\ & (RlxConv, \text{4E 49 43 4B}), \\ & (Streaming, \text{52 54 53 50}), \\ & (Streaming, \text{76 69 64 65 6F 2F}), \\ & (Interactive, \text{48 45 4C 4F}), \\ & (Background, \text{53 4D 42})\} \end{aligned} \tag{5.6}$$

Let K_{SIG} be a signature-based classifier that is associated with the set of signatures described in (5.6). Given a string

$$\text{CD AD 52 54 53 50 2F 31 2E 30 20 32 30 30 20 4F},$$

we have

$$K_{SIG}(\langle \text{CD AD 52 54 53 50 2F 31 2E 30 20 32 30 30 20 4F} \rangle) = Streaming.$$

The signature 52 54 53 50 is a substring of the given string, so the classifier returns the class associated to the signature. Now, suppose the default class is Background class and the input string is

$$\text{E1 6F 02 8C 9E C1 A4 E2 BF}$$

Table 5.18: Statistics of WIDE traces - Number and percentage of flows identified.

	No. of flows	Percentage
Identified	35,042,233	25.23
Unidentified	103,856,128	74.77
Total	138,898,361	100

we have

$$K_{SIG}(\langle \text{E1 6F 02 8C 9E C1 A4 E2 BF} \rangle) = \textit{Background}.$$

Here, the input string does not contain any signature. The classifier simply returns a default class, which is the Background class. In the literature, the default class is different in different implementations and is generally called the Unknown class.

Because a reliable ground truth is of utmost importance to an effective evaluation of our flow classification system, much effort is put in the signature analysis. Apart from our own investigations, signatures from other studies including [Pro08], [RG07], [Pha05], [ZP00], [Pol06], [Yah08] are also employed, resulting in a large collection of identified signatures from a variety of applications. (See Table 2.3 for a complete list of applications supported by our signature-based FCS.) This rich set of applications covers all service classes and most of the applications used thus far in other researches. Further details of the signatures of each class are provided in Appendix C.1. Using PacketExtract, 25.23% of more than 138 million flows are identified (see Table 5.18). 71.78% of the identified flows consist primarily of HTTP flows, which belongs to Interactive class. Bulk class follows with 16.01%. RlxConv class makes up 12.17% of all identified flows, followed by Streaming and and StrConv classes with 0.03% and 0.01%, respectively. Table 5.19 summarizes the proportion of each class within each packet trace. Detailed information about the traces and their statistics is provided in Appendix D.

At any rate, large networks such as the Internet are overwhelmed with extremely short flows. These flows do not contain meaningful information but rather are caused by network control protocols, failed connection requests, pings, etc. We therefore excluded the short flows from our experimental data by filtering out flows that contain less than two packets. Additionally, as shown in Table 5.16, the ratio of IPv6 packets is extremely small in the traces. We consequently chose to discard them and consider only IPv4 packets.

It is worth noting that, for RlxConv class, the numbers of the flows in each time slot vary greatly (see Table 5.19). This is because there is a large number of short simultaneous SSH connections from a few hosts to many other hosts, which are most likely to be port scanning. However, we do not remove such flows, because, even though may not be relevant, they still belong to the class. A similar observation is also found in the last trace of Streaming class. As shown in Table 5.19, in the last trace, the number of streaming flows is much higher than the other traces, which is also due to the short simultaneous flows from a few hosts. Again, as they belong to the class, they are not removed.

Table 5.19: Statistics of WIDE traces - Number of flows within each class.

Traces	StrConv	RlxConv	Str	Int	Bulk	Total
March 18						
00:00	691	1,181,179	128	1,478,524	524,327	3,184,849
08:00	152	543,973	95	1,174,395	398,927	2,117,542
12:00	196	524,260	658	1,904,665	281,186	2,710,965
16:00	260	524,999	441	2,111,387	316,451	2,953,538
20:00	491	605,375	254	1,600,020	336,284	2,542,424
March 19						
00:00	724	34,678	141	1,616,291	442,016	2,093,850
08:00	157	30,205	94	1,433,811	356,126	1,820,393
12:00	207	3,937	219	1,705,898	291,616	2,001,877
16:00	301	6,907	273	2,142,470	360,943	2,510,894
20:00	214	4,446	87	1,789,957	413,280	2,207,984
March 20						
00:00	144	8,795	88	1,663,293	414,748	2,087,068
08:00	163	5,224	175	1,193,042	346,176	1,544,780
12:00	191	245,931	60	1,700,262	291,830	2,238,274
16:00	255	393,163	77	1,789,793	362,398	2,545,686
20:00	386	151,392	6,073	1,851,004	473,254	2,482,109
Total	**4,532**	**4,264,464**	**8,863**	**25,154,812**	**5,609,562**	**35,042,233**
Total (%)	**0.01**	**12.17**	**0.03**	**71.78**	**16.01**	

5.5.3 Data Stratification

While the WIDE data set contains more than 35 million feature-vector instances, the class distributions are highly skewed (see Table 5.19). The number of flows in each class varies greatly, from only 4,532 instances in the StrConv class to more than 25 million in the Interactive class. Determining classification performance from such a skewed dataset would not be sensible. Therefore, the flow instances are equally sampled into a smaller dataset before the experiments are conducted. In any case, even if the classes are well distributed, the sampling process would still be inevitable as it would not be feasible to perform the evaluation on the dataset of 35 million instances.

We have randomly sampled 4,000 instances from each class, constituting a dataset of 20,000 instances. However, to avoid any biases, the sampling process is done 10 times. This results in 10 datasets, each of which has 20,000 instances stratified with 4,000 instances per class. Note that the number of instances per class is set to 4,000 in this case because it already covers almost all of the 4,532 instances in the StrConv class.

5.5.4 Evaluation Results

Similar to individual-user datasets, we evaluate our flow classification technique in two phases. The classification accuracies of various learners will be evaluated first followed by the time required to learn. With WIDE dataset, it can still perform well in a large dataset consisting of large number of users.

As shown in Table 5.20, the results are consistent with those from individual-user sets — i.e., all but one method yield remarkable accuracy in our experiments. PART, RIPPER and J4.8 perform particularly well with more than 99% average correctness compared to the 92.97% achieved by k-NN. Naive Bayes, on the other hand, has much lower average accuracy of just 42.08%. Moving to per-class correctness, while other methods can classify flow instances of all classes equally well, Naive Bayes performs poorly on all classes except Bulk as it classifies most of the instances as Bulk. This shows that the employed features are correlated and the feature-independence assumption of Naive Bayes does not hold in our domain. Table 5.21 presents per-class accuracy results, which contradict those of the experiments conducted by Moore and Zuev [MZ05b] and Williams et al. [WZA06], whereby the Naive Bayes achieved more than 80% accuracy. It is conjectured that the different between their results and ours are contributed by the following factors:

- The service class definition: [WZA06] defines service classes over application protocols. That is, the FCSs have to classify flows into different protocols, such as FTP, SMTP, HTTP, etc. In our case, service classes are defined by QoS requirements. Common behaviors within the same protocols might be more obvious than those within different protocols that belong to the same service classes defined over QoS requirements.

- Unreliable data: In [MZ05b], the datasets used are unreliable for two reasons. The first is that about 87% of the instances belong to the "WWW" class (i.e., highly skewed); the other reason is that the Naive Bayes learner classifies virtually all instances to this class. In [WZA06], the ground truth is established based on protocol ports. Therefore, one can never be certain if the results are reliable.

Nevertheless, given the excellent performance of the other methods, we can still safely conclude that the features are discriminative in both small and large sets of data, depending on how the relationships between the features and classes are modeled.

In terms of computational time, the results are also consistent with the individual-user ones. Naive Bayes outperforms other methods with only half a second learning time followed by J4.8, PART, and RIPPER respectively. The huge difference in learning time between individual-user and WIDE datasets is caused by the size of the data. The results are reported in Figure 5.5 and Table 5.22.

The evaluation results of the WIDE dataset show that SMART is usable, versatile, and the accuracy is also comparable with other the techniques in the literature. Although the dataset used in the evaluations are not the same the others, most of the applications and services present in other researches are also present in our dataset (as shown in Table 2.3). Moreover, the actual types of flows (i.e., the ground truth) in our dataset are identified by a sophisticated and reliable signature-based FCS. Thus, our results are more robust than many other methods, especially the ones that rely on protocol ports to find the ground truths, such as [WZA06], [ZNA05b] and [ML05].

Table 5.20: Accuracy of learners - evaluated on WIDE dataset.

Learner	Accuracy
J4.8	99.35
RIPPER	99.51
PART	99.66
Naive Bayes	42.08
k-NN	92.97

Table 5.21: Per class accuracy

Class	J4.8	RIPPER	PART	Naive Bayes	k-NN
StrConv	99.98	99.82	99.91	38.86	99.84
RlxConv	99.15	99.47	99.57	20.95	95.74
Streaming	98.76	99.21	99.36	41.29	92.33
Interactive	98.75	99.02	99.39	19.10	84.31
Bulk	99.97	99.94	99.97	99.39	91.73

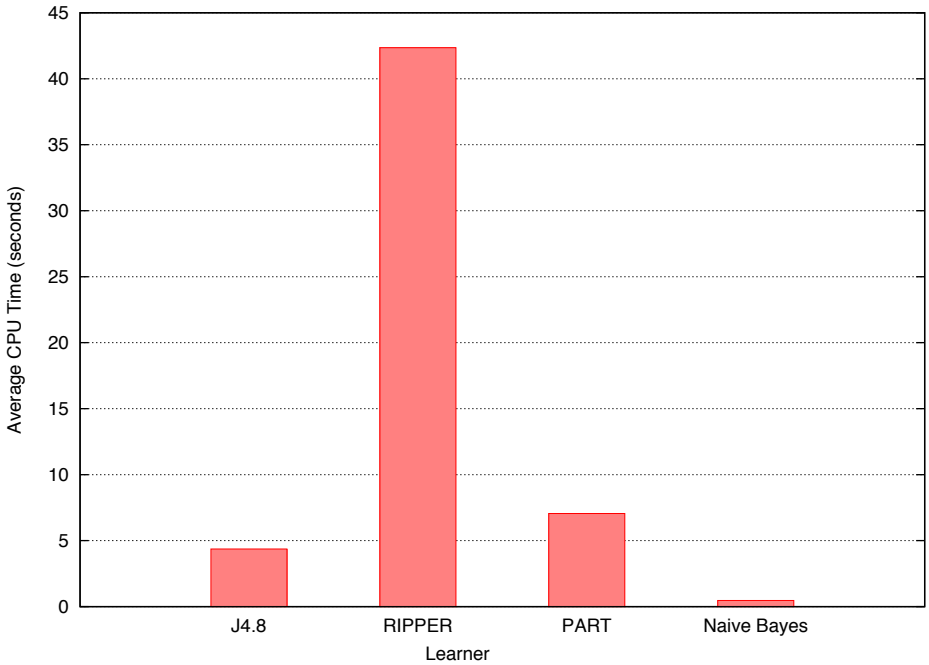


Figure 5.5: Average CPU time taken to learn from WIDE dataset. The learning time of all algorithms is consistent with that evaluated in individual datasets whereby Naive Bayes performs the best followed by J4.8, PART, and RIPPER respectively. They all take much longer time to learn in general, however, because the size of the WIDE dataset is much larger than the individual ones.

Table 5.22: Average CPU time taken to learn from WIDE dataset. k-NN learning time is not included as it is a lazy algorithm.

Learners	CPU Time (seconds)
J4.8	4.37
RIPPER	42.35
PART	7.06
Naive Bayes	0.47

5.6 Conclusion

In this chapter, we have described a set of service classes, features, and machine learning algorithms that are used by our new SMART FCS. Particularly, we have introduced a new feature, *TPUTDiff*, which is intended to capture the burstiness of flows. Our classification methodology is then evaluated in datasets from individual users and large packet traces to see if it is feasible in both end-user and lower-level network devices. The results from our individual-user evaluations show that exceptional accuracies can be achieved in all our diverse datasets, indicating that the employed features are discriminative. The low cross-user prediction accuracies imply that knowledge learned by a user cannot yet be effectively used in other devices. We believe, however, that the classification results could be improved by implementing more generalized classification models. These results have been reported in [AS07a]. It is worth noting that such individual-users-level evaluations have not been carried out anywhere before. For the datasets obtained from large packet traces, the results are consistent with the individual-user sets; apart from Naive Bayes, all learners yield exceptional correctness. We can thus safely conclude from those experiment results that our choices of features and learners are sound and can effectively be used in any flow classification task in general. Nevertheless, since the features used in our experiments are computed using entire flows, they are not yet suitable for real-time classification.

In the next chapter, the issue of an optimal observation period will be discussed. We will see how real-time classification can be carried out and what would be the minimum amount of information required to identify the service class of a flow. We will also explore ways to improve our *TPUTDiff* feature such that it can be used in real-time classification. Furthermore, we will investigate relationships between features and the required prefix that needs to be observed.

Chapter 6

Classifying Flows in Time

To effectively assist QoS support to QoS-unaware flows, the flow classification has to be carried out in real-time. It also has to be adaptive to be able to handle the flows from unknown services. In the previous chapter, we have shown that adaptive classification is feasible in the real world. In this chapter, we will systematically study the nature of Internet connections, their features, as well as the possibility of extracting and classifying flows in real-time.

6.1 Towards Real-Time Flow Classification

Before a flow is analyzed and classified, it has to be abstracted into a feature vector. Each component in the feature vector is the feature value representing a property or characteristic of the flow. To be able to capture the true characteristics (i.e., the true feature values) of the flow, the feature values must be calculated using every packet in the flow. Doing so, however, is not suitable for real-time classification. We conjecture that instead of extracting features after the entire flows have been observed, it should be possible to capture the feature values using only partial flows. In other words, **the flow features should be able to be computed using only a specific prefix**.

To verify our hypothesis, the possibility of capturing the characteristics using only partial flows will be explored in this chapter. We will begin by defining the "distribution" of feature values and discussing how to compare and analyze the distributions. Next, the distributions will be computed using different prefixes and these will then be compared to the one computed using the entire flows. By studying the distribution differences of various prefixes, one can see how the distributions behave with respect to the flow prefixes. Then, we will turn to analysis of accuracies achieved by the learners with respect to the prefixes. Lastly, we will investigate the relationship between prefixes and discriminability of each feature through an extensive series of empirical evaluations.

6.2 Feature Values Divergences

In this section, the relationship between feature values and prefixes is studied. The distributions of the feature values are computed using different prefixes and are compared to the one using the entire flows. If the distributions of the feature values computed using up to a prefix are *close* to those computed using the entire flows, it means that the feature values should be also able to be estimated using only that prefix.

The focus of this section is to examine the relationship between feature values and prefixes. A key concept here is the distribution of feature values, which we will formally define as follows.

6.2.1 Frequency Distribution

Definition 6.1 (Sequence of Observations) Let D be a set and $m \in \mathbb{N}^+$. A *sequence of observations s over D* is a non-empty finite sequence

$$s = (s_1, \ldots, s_m)$$

such that $(\forall 1 \leq i \leq m) s_i \in D$. We call m the *length of s* or the *number of observations in s* and each component in s is called an *observation*. A set of sequence of observations over D is denoted by $\mathcal{S}(D)$.

Essentially, a sequence of observations is a sequence all of whose components belong to the same domain. In our context, a sequence of observations is a sequence of observed feature values.

Definition 6.2 (Frequency) Let D be a set and $d \in D$. The *frequency of d in a sequence* $(s_1, \ldots, s_m) \in \mathcal{S}(D)$ is a function

$$\begin{aligned} \mathit{freq}: \quad & D \times \mathcal{S}(D) \to \mathbb{N} \\ & (d, (s_1, \ldots, s_m)) \mapsto \begin{cases} 1 + \mathit{freq}(d, (s_2, \ldots, s_m)) & \text{if } d = s_1, m > 1, \\ \mathit{freq}(d, (s_2, \ldots, s_m)) & \text{if } d \neq s_1, m > 1, \\ 1 & \text{if } d = s_1, m = 1, \\ 0 & \text{if } d \neq s_1, m = 1. \end{cases} \end{aligned}$$

Definition 6.3 (Relative Frequency) Let D be a set and $d \in D$. The *relative frequency of d in a sequence* $(s_1, \ldots, s_m) \in \mathcal{S}(D)$ is a function

$$\begin{aligned} \mathit{RelFreq}: \quad & D \times \mathcal{S}(D) \to [0, 1] \\ & (d, (s_1, \ldots, s_m)) \mapsto \frac{\mathit{freq}(d, (s_1, \ldots, s_m))}{m}. \end{aligned}$$

Intuitively, the frequency of an element d in a sequence s is the number of components in s whose value equals d. The relative frequency of d is the frequency of d normalized by the length of the sequence. Consequently, the maximum value of the relative frequency is one.

Example 6.4 (Relative Frequency)
Given a sequence of observations

$$s = (2.4, 0.9, 2.4, 5.8, 0.8),$$

the relative frequencies of the components in s are::

$$\begin{aligned} \mathit{RelFreq}(2.4, s) &= 0.4 \\ \mathit{RelFreq}(0.9, s) &= 0.2 \\ \mathit{RelFreq}(5.8, s) &= 0.2 \\ \mathit{RelFreq}(0.8, s) &= 0.2 \end{aligned}$$

whereas

$$\mathit{RelFreq}(10, s) = 0.$$

With relative frequency, one can compute the distribution of each feature value relative to other values in the observed sequence. However, analyzing the observed feature values directly is not practical because individual values could occur a few times or only once although they are very close to each other (e.g., the values 0.9 and 0.8 in the example above). Therefore, we would like to group the contiguous values together. We call this grouping a "histogram", which is a collection of adjacent, non-overlapping intervals defined over a set of usually real numbers. Each interval is called a "bin" and the size of a bin is called "width".

Definition 6.5 (Histogram) Let D be a set, $w \in \mathbb{R}^+$, and $b \in \mathbb{N}^+$, a *wb-histogram* is a function

$$H_{w,b} : \quad \mathbb{R} \to \mathbb{N}$$
$$d \mapsto \begin{cases} \left\lfloor \frac{d}{w} \right\rfloor & \text{if } \left\lfloor \frac{d}{w} \right\rfloor < b - 1, \\ b & \text{if } \left\lfloor \frac{d}{w} \right\rfloor \geq b - 1. \end{cases}$$

For an element $d \in D$, we call $H_{w,b}(d)$ the *bin of* d with *bin-width* w and *total number of bin* b.

Essentially, a histogram is a function that maps an element in $d \in \mathbb{R}$ to a bin. Because a bin must be an interval and the histogram must have at least one bin, the bin-width and the total number of bin cannot be zero. The bin to which a given value d is assigned is determined by dividing d by the bin-width w. As a result of the flooring function, if the division result has a remainder, the resulting bin is the (integer) quotient of the division (i.e., the division remainder is dropped). If the resulting bin is larger than the preferred total number of bin b, d will be mapped to the last bin. Therefore, there are b possible bins: $0, \ldots, b-1$ and the maximum value that the histogram can effectively express is $w \times (b-1)$ (as all the higher values belong to bin $b-1$). At any rate, if it is clear from the context or it is irrelevant, the subscript w, b will be dropped.

Definition 6.6 (Histogram Sequence) Given a sequence $(s_1, \ldots, s_m) \in \mathcal{S}(D)$, a *histogram sequence induced by $H_{w,b}$ with respect to s* is a sequence

$$(h_1, \ldots, h_m)$$

such that $(\forall 1 \leq i \leq m) h_i = H_{w,b}(s_i)$

Example 6.3 (continued)
Consider the sequence

$$s = (2.4, 0.9, 2.4, 5.8, 0.8)$$

presented earlier and the histogram $H_{1,5}$, the histogram sequence induced by $H_{1,5}$ with respect to s is

$$h = (2, 0, 2, 4, 0).$$

The relative frequencies of values in h are:

$$\begin{aligned} \mathit{RelFreq}(0, h) &= 0.4 \\ \mathit{RelFreq}(2, h) &= 0.4 \\ \mathit{RelFreq}(4, h) &= 0.2 \end{aligned}$$

Definition 6.7 (Frequency Distribution) Given a sequence of observation $s = (s_1, \ldots, s_m)$ and a histogram function H, the *frequency distribution of H with respect to s* is a function defined as

$$\begin{aligned} \mathrm{Fr}_{H,s} : \quad & \mathbb{N} \to [0, 1] \\ & k \mapsto \mathit{RelFreq}(k, (H(s_1), \ldots, H(s_m))) \end{aligned}$$

In other words, the frequency distribution is the relative frequency of a given value k with respect to a histogram sequence H. Figure 6.1 illustrates the frequency distribution of the histogram sequence specified in Example 6.3. To avoid ambiguity, we clearly show the histogram function and sequence of observation from which each frequency distribution is induced. In most cases, however, such pedantry is not necessary. Therefore, we will use Fr to denote a frequency distribution where it is either unnecessary or clear from the context which histogram function and observation sequence are associated with the frequency distribution.

6.2.2 Frequency Distribution Difference

To be able to systematically study the convergence of distributions, the distribution difference must be precisely described. In the literature, a number of metrics have been proposed to measure the difference between two distributions. One of the most popular measures is called the Kullback-Leibler divergence [CT91] or KL divergence in short. It is characterized as follows:

$$\mathrm{Diff}_{KL}(\mathrm{Fr}_{H,s} || \mathrm{Fr}_{H,s'}) = \sum_{k \in \mathbb{N}} \mathrm{Fr}_{H,s}(k) \log_2 \frac{\mathrm{Fr}_{H,s}(k)}{\mathrm{Fr}_{H,s'}(k)}.$$

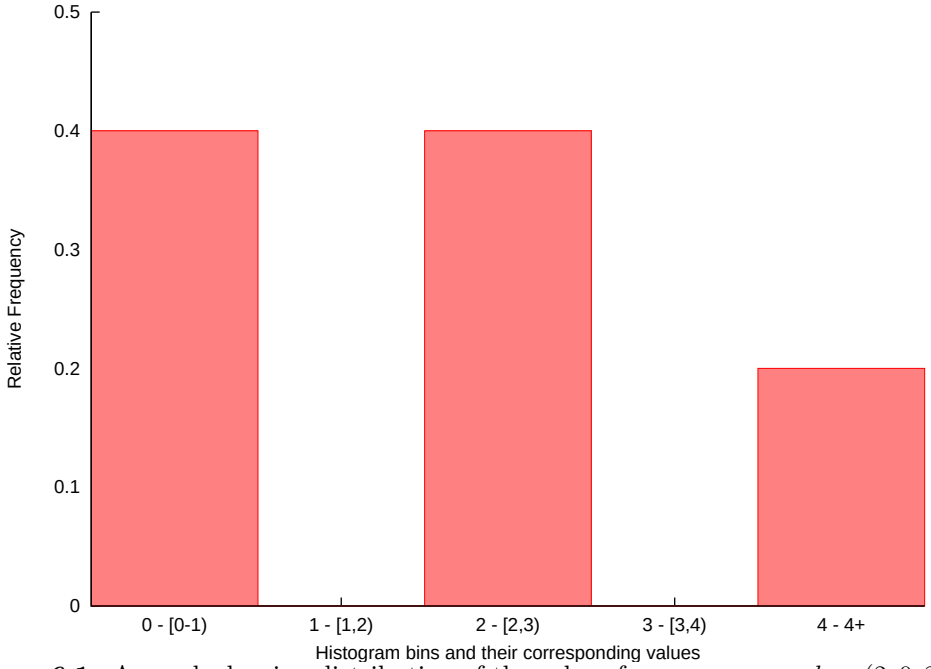


Figure 6.1: A graph showing distribution of the values from a sequence $h = (2, 0, 2, 4, 0)$

KL divergence measures expected bits[1] required to code the samples from distribution $\mathrm{Fr}_{H,s}$ using $\mathrm{Fr}_{H,s'}$. The higher the bits required, the higher the difference. KL divergence derived from the information theory discussed earlier in Section 4.1 is termed *divergence* instead of *distance* because it is not symmetric. It is important to note also that the two distribution functions must be induced by the same histogram function. Otherwise, they would not be comparable. The main drawback of the KL divergence is that the maximum divergence is infinity [CT91]. Precisely, if there exists a k such that $\mathrm{Fr}_{H,s}(k) > 0$ and $\mathrm{Fr}_{H,s'}(k) = 0$, then $\mathrm{Diff}_{KL}(\mathrm{Fr}_{H,s}||\mathrm{Fr}_{H,s'}) = \mathrm{Fr}_{H,s}(k) \log \frac{\mathrm{Fr}_{H,s}(k)}{0} = \infty$. Therefore, even though KL divergence is widely used, it is not preferable.

Another metric used to measure the distance between two distributions is called the Bhattacharyya distance [Bha43], which is given by:

$$\mathrm{Diff}_B(\mathrm{Fr}_{H,s}||\mathrm{Fr}_{H,s'}) = -\ln\left(\sum_{k\in\mathbb{N}} \sqrt{\mathrm{Fr}_{H,s}(k) \times \mathrm{Fr}_{H,s'}(k)}\right).$$

Because $\ln 0 = \infty$, $\mathrm{Diff}_B(\mathrm{Fr}_{H,s}||\mathrm{Fr}_{H,s'}) = \infty$, if $\mathrm{Fr}_{H,s}$ is orthogonal to $\mathrm{Fr}_{H,s'}$. Thus, like KL divergence, Bhattacharyya distance is not preferable. In turn, we propose a difference metric that does not involve infiniteness:

Definition 6.8 (Frequency Distribution Difference) Given a histogram function H and sequences of observation s and s', the *difference between two frequency distributions*

[1] In general, the difference is measured in bits because the base of the logarithm is 2. However, the base of the logarithm is not restricted to two. For instance, the base could also be e, then the unit would be nats.

$\mathrm{Fr}_{H,s}$ *and* $\mathrm{Fr}_{H,s'}$ is defined as

$$\mathrm{Diff}(\mathrm{Fr}_{H,s}||\mathrm{Fr}_{H,s'}) = \sum_{k\in\mathbb{N}} |\mathrm{Fr}_{H,s}(k) - \mathrm{Fr}_{H,s'}(k)|.$$

Intuitively, the distribution difference is the difference of the areas under the two distribution functions. The area difference will be zero only when the two distributions are exactly the same. The maximum difference is two and will arise when the distributions do not overlap at all. Again, we would like to emphasize that two distributions are comparable only when they are induced from the same histogram function. In the following example, we will see how the distribution differences are computed. We will also see how the distributions from different prefixes are related.

Example 6.9 (Distribution Differences)
Consider a set of flow

$$\mathcal{F} = \{f_1, \ldots, f_m\} \tag{6.1}$$

with a feature *dataTPUTAvg*, its length-restricted counterpart, *dataTPUTAvg*$'$, and length $l \in \mathbb{N}^+$. The following sequences can be obtained:

$$s^{\text{full}} = (dataTPUTAvg(f_1), \ldots, dataTPUTAvg(f_m)) \tag{6.2}$$
$$s^{[l]} = (dataTPUTAvg'(f_1, l), \ldots, dataTPUTAvg'(f_m, l)). \tag{6.3}$$

The sequence s^{full} contains values of the features *dataTPUTAvg* of all flows in $\mathcal{F}$. Each feature value is calculated using the entire flow. The values in $s^{[l]}$, on the other hand, are calculated from only l packets. Note that each sequence contains only values of only one feature.

Let us instantiate the flows $f_1, \ldots, f_m$ in Equation (6.1) with the flows of Streaming class from WIDE dataset. Then, given a histogram function, $H_{300,30}$, we obtain the frequency distribution shown in Figure 6.2. We call a distribution obtained from the entire flows a "true distribution" as it is calculated using all packets in the flows and, thus, represents the actual characteristics of the flows.

Now, to see how the prefixes are related to the feature value distributions, we instantiate the sequence of observations $s^{[l]}$ presented in Equation (6.3) with three following sequences $s^{[4]}$, $s^{[8]}$, and $s^{[16]}$:

$$s^{[4]} = (dataTPUTAvg'(f_1, 4), \ldots, dataTPUTAvg'(f_m, 4)) \tag{6.4}$$
$$s^{[8]} = (dataTPUTAvg'(f_1, 8), \ldots, dataTPUTAvg'(f_m, 8)) \tag{6.5}$$
$$s^{[16]} = (dataTPUTAvg'(f_1, 16), \ldots, dataTPUTAvg'(f_m, 16)) \tag{6.6}$$

Figures 6.3 shows the distributions obtained from several flow lengths in comparison to the true distribution. These distributions are computed from all flows (totaling more than 8,800) in the WIDE dataset that belong to the Streaming class. As shown in the figure, as more packets are observed, the distributions will get *closer* to the true distribution. In other words, the distributions corresponding to different prefixes are "converging" toward the true distribution.

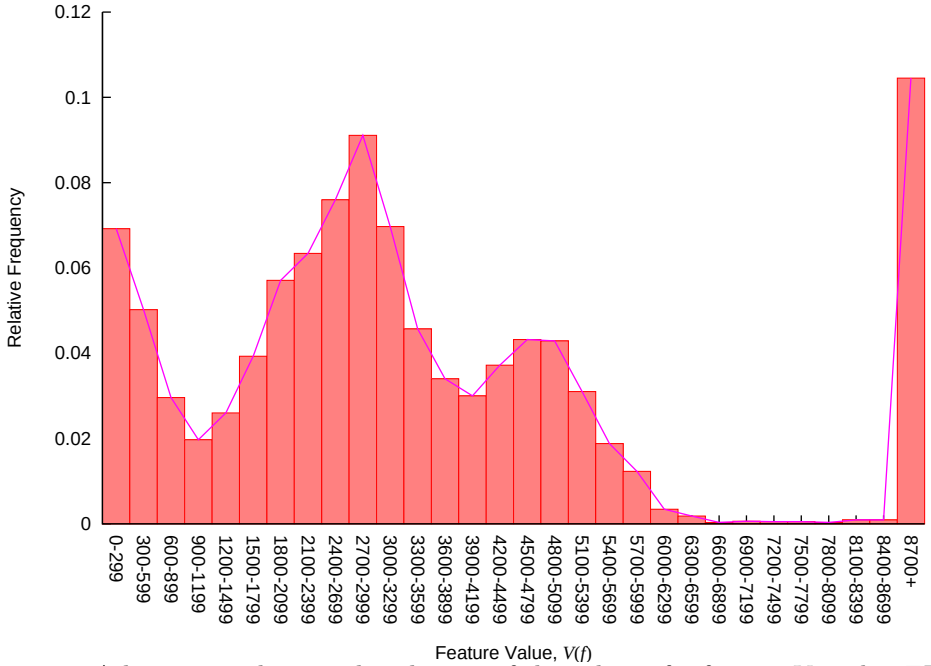


Figure 6.2: A histogram showing distribution of the values of a feature $V = dataTPUTAvg$ where the width and number of bins are 300 and 30 resp. The pink line is the frequency polygon of representing the distribution.

Figure 6.4(a)-(c) depict the areas between distributions induced from sequences (6.4) - (6.6) and the one induced by (6.2), respectively. As shown in the figures, the differences between the distributions are getting smaller. The difference-prefix plot is also shown in Figure 6.4(d).

6.3 Packet-Size Difference — A New Real-Time Feature

In the previous chapter, a new feature, *TPUTDiff*, has been introduced to capture the burstiness, which is calculated by measuring the differences of throughputs along the flows. This feature is, however, not suitable for real-time classification as it requires a certain number of packets per calculation window to ensure the correct estimations of the throughputs.

Therefore, we introduce here features that, like *TPUTDiff*, try to capture the changes of the flow characteristics over time while at the same time exhibit greater flexibility. These are *pktSizeDiff* and its coflow counterpart, *pktSizeDiffCF*, whose

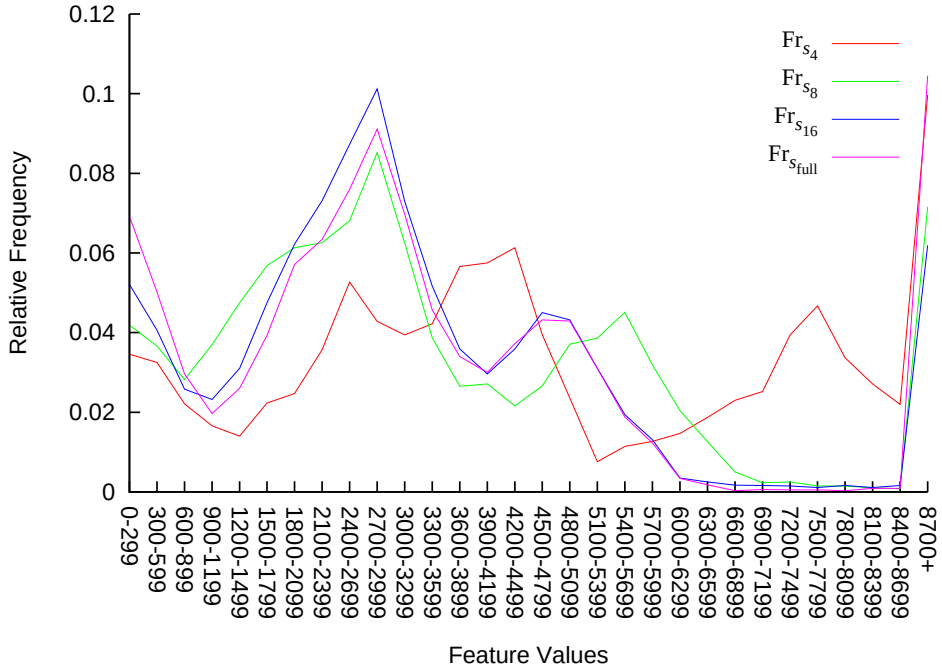


Figure 6.3: Frequency polygons showing the distributions of feature values using different flow prefixes. $Fr_{s_{full}}$ represents the same distribution as the one shown in Figure 6.2. Notice that the distributions converge on the true distribution as more packets are observed.

precise definitions are given by:

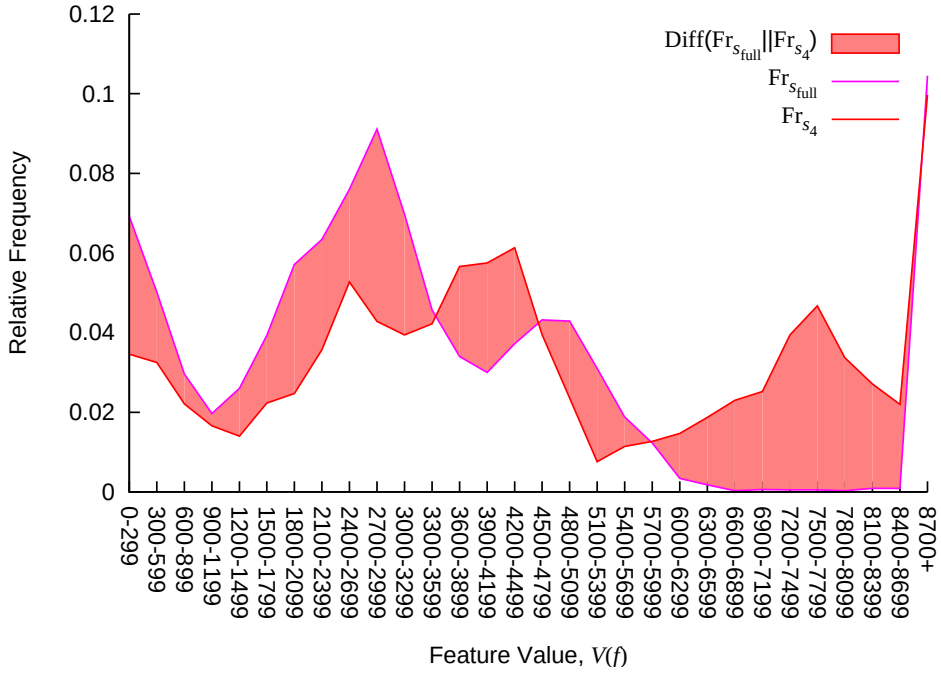
$$\begin{aligned} \mathit{pktSizeDiff}: \quad & \mathcal{F} \to \mathbb{R} \\ & (p_i \mid 1 \le i \le n) \mapsto \frac{\sum_{i=1}^{n-1} |\mathit{size}(p_i) - \mathit{size}(p_{i+1})|}{n} \\ \mathit{pktSizeDiffCF}: \quad & \mathcal{F}' \to \mathbb{R} \\ & f' \mapsto \begin{cases} \mathit{pktSizeDiff}(f'') & \text{if } (\exists f'' \in \mathcal{F})(f', f'') \in \mathit{Coflow}, \\ 0 & \text{otherwise.} \end{cases} \end{aligned}$$

Intuitively, both functions try to capture the differences of packet sizes throughout the flow. Unlike *TPUTDiff*, they can be computed at any prefix that is more than two.

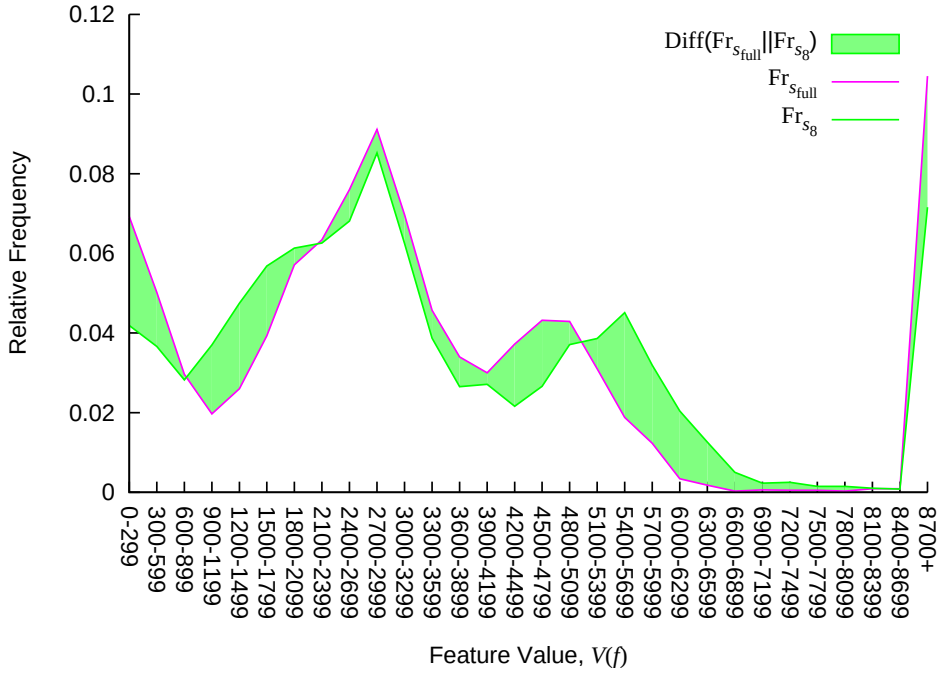
6.4 Frequency Distributions in Real-World Data

In Example 6.9, we have seen that the distributions of the feature values converge on the true distribution. However, each of them is the distribution of only one feature and is obtained from only one class of flow. In this section, we will see if this phenomenon holds in general through a series of empirical experiments, in which the distributions of all features are computed using different prefixes. In addition, the distribution difference corresponding to each feature will also be analyzed.

In Section 5.4, the dataset is obtained directly from multiple users using FlowStatLive. The feature vectors are calculated from the observed flows and are stored in

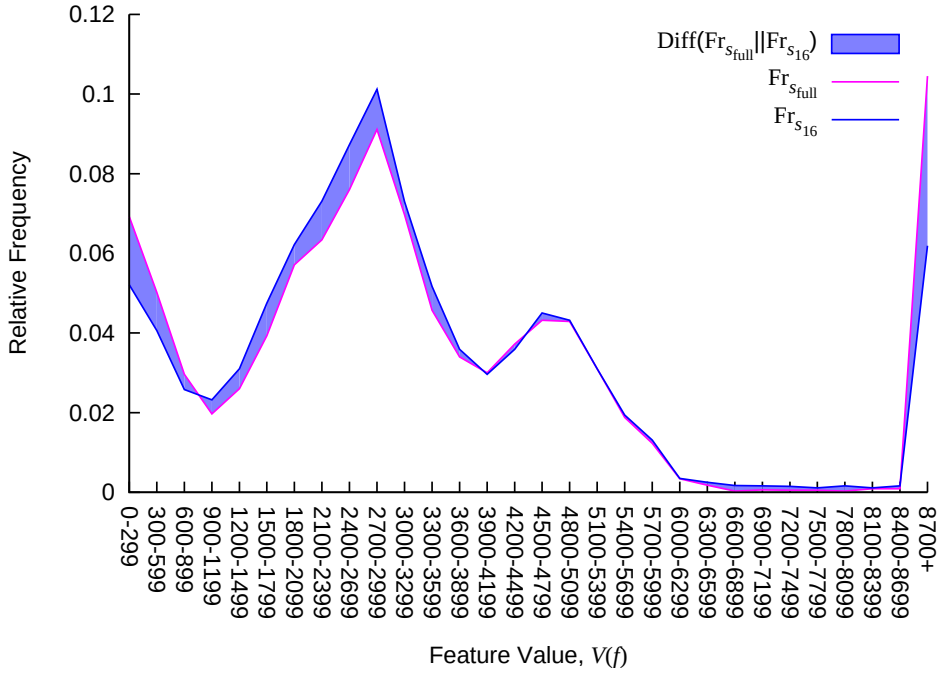


(a)

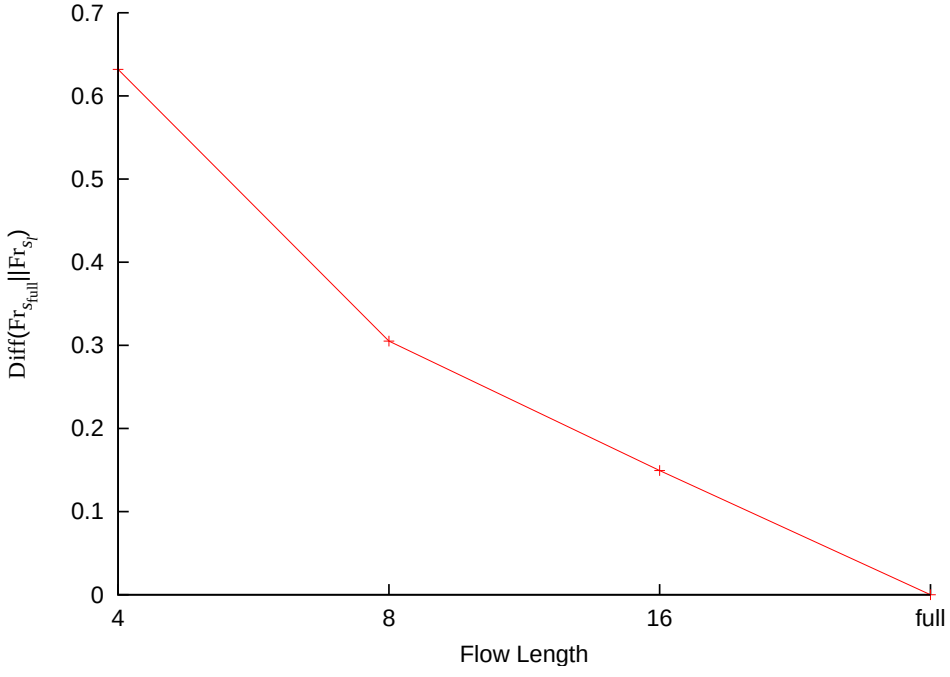


(b)

Figure 6.4: Subfigure (a)-(c) show the frequency distributions differences $\mathrm{Diff}(\mathrm{Fr}_{s_{\mathrm{full}}}||\mathrm{Fr}_{s_l})$ where $l = 4, 8, 16$ resp. Subfigure (d) presents the differences wrt. the lengths of the flows.



(c)



(d)

Figure 6.4 (continued)

a database on-the-fly. This approach, however, is not suitable here because, in order to study the relationship between flow prefix and feature values, the feature values have to be calculated from multiple prefixes. In other words, multiple feature vectors corresponding to different flow lengths have to be extracted from the same flow. Doing so in ordinary mobile devices is not feasible due to restricted storage and computational power. As a result, the individual-users dataset cannot be used anymore in our evaluations. We will focus only on the WIDE packet traces and the datasets that are generated from them.

In the following experiments, two series of datasets are generated: linear-increment and exponential-increment series. In linear-increment series, a set of length-restricted datasets is generated such that the corresponding length is increased linearly from 4 to 20. Precisely, from a given raw dataset $\mathcal{R}$, a set of length-restricted features $\mathcal{V}' = \{V'_1, \ldots, V'_d\}$, and a dataset generator function G' corresponding to $\mathcal{V}'$, we generate:

$$\mathcal{D}_{[l],\mathcal{R}} = G'(\mathcal{R}, l), l \in \{4, \ldots, 20\}.$$

That is, after the generation, we have:

$$\mathcal{D}_{[4],\mathcal{R}}, \mathcal{D}_{[5],\mathcal{R}}, \ldots, \mathcal{D}_{[20],\mathcal{R}}. \tag{6.7}$$

For exponential-increment series, the length is increased exponentially:

$$\mathcal{D}_{[l],\mathcal{R}} = G'(\mathcal{R}, l), l \in \{2^2, 2^3, \ldots, 2^{16}\}.$$

The following datasets are then created:

$$\mathcal{D}_{[2^2],\mathcal{R}}, \mathcal{D}_{[2^3],\mathcal{R}}, \ldots, \mathcal{D}_{[2^{16}],\mathcal{R}}. \tag{6.8}$$

In addition, a dataset using entire flows is also generated[2]:

$$\mathcal{D}_{\text{full},\mathcal{R}} = G(\mathcal{R}).$$

After the all datasets are generated, the distribution difference of each feature is computed using the distributions of feature values in $\mathcal{D}_{[l],\mathcal{R}}$ and $\mathcal{D}_{\text{full},\mathcal{R}}$. The following is a step-by-step description of the process.

Consider the series of linear-increment datasets shown in (6.7). Each dataset $\mathcal{D}_{[l],\mathcal{R}}, 4 \leq l \leq 20$, consists of the following elements:

$$\mathcal{D}_{[l],\mathcal{R}} = \dot{\{}\langle x^{[l]}_{11}, \ldots, x^{[l]}_{d1}, c_1\rangle, \ldots, \langle x^{[l]}_{1n}, \ldots, x^{[l]}_{dn}, c_n\rangle\dot{\}}$$

where n is the cardinality of $\mathcal{D}_{[l],\mathcal{R}}$, $x^{[l]}_{ij}, 1 \leq l \leq d$, denotes the value of feature V'_i of flow instance j, $1 \leq j \leq n$, which is calculated using prefix of length l, and c_j denotes the class of the j-th instance. Then, for each feature V'_i, the value of the feature of each instance is projected into a sequence of observation as follows:

$$s^{[l]}_i = (x^{[l]}_{i1}, \ldots, x^{[l]}_{in}).$$

[2]Note that the dataset generator, in this case, is an ordinary generator function and not the length-restricted counterpart.

Performing the projection on all prefixes, we have

$$\begin{aligned} s_i^{[4]} &= (x_{i1}^{[4]}, \ldots, x_{in}^{[4]}), \\ s_i^{[5]} &= (x_{i1}^{[5]}, \ldots, x_{in}^{[5]}), \\ &\vdots \\ s_i^{[20]} &= (x_{i1}^{[20]}, \ldots, x_{in}^{[20]}). \end{aligned}$$

A sequence of observation corresponding to $\mathcal{D}_{\text{full},\mathcal{R}}$ is also obtained:

$$s_i^{\text{full}} = (x_{i1}^{\text{full}}, \ldots, x_{in}^{\text{full}}).$$

Next, after the frequency distributions of the sequences are computed through a histogram H, we have

$$\text{Fr}_{H,s_i^{[4]}}, \ldots, \text{Fr}_{H,s_i^{[20]}}$$

and

$$\text{Fr}_{H,s_i^{\text{full}}}.$$

Finally, the distribution corresponding to each flow length can be compared to that corresponding to the entire flows.

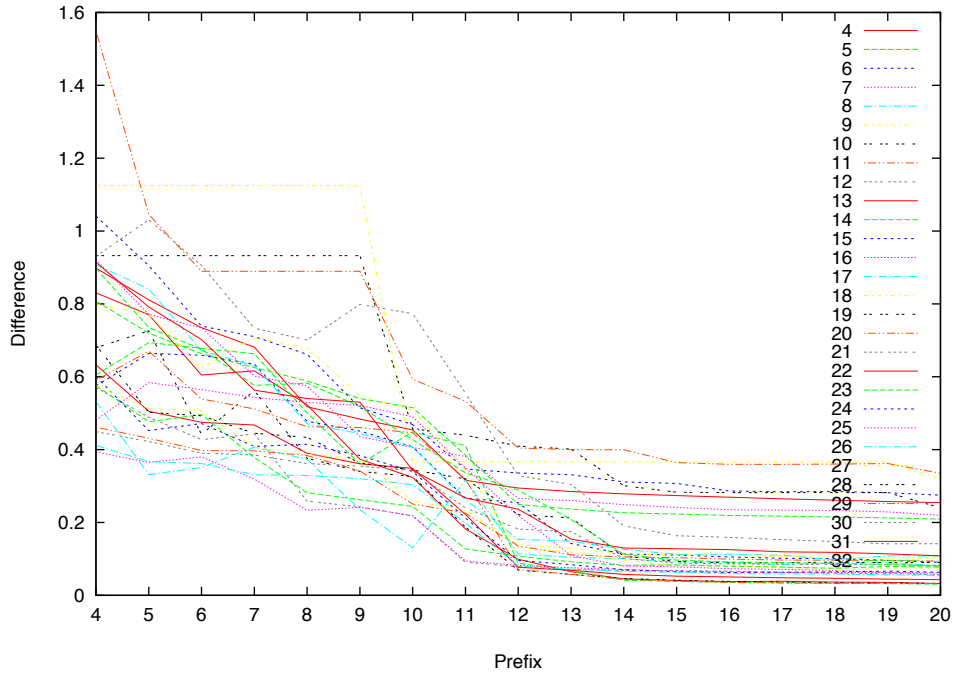
$$\text{Diff}(\text{Fr}_{H,s_i^{[4]}} || \text{Fr}_{H,s_i^{\text{full}}}), \ldots, \text{Diff}(\text{Fr}_{H,s_i^{[20]}} || \text{Fr}_{H,s_i^{\text{full}}}) \tag{6.9}$$

Note though that the differences shown in (6.9) correspond to only one feature. Figure 6.5 illustrate how the distribution differences behave with respect to the prefixes. The distributions are obtained from all flows of all classes in the WIDE dataset. In total, more than 32 million flow instances are considered. As shown in the figure, the differences tend to converge as more prefixes are observed. At prefix 11, differences of most of the features drop to less than 0.4[3].

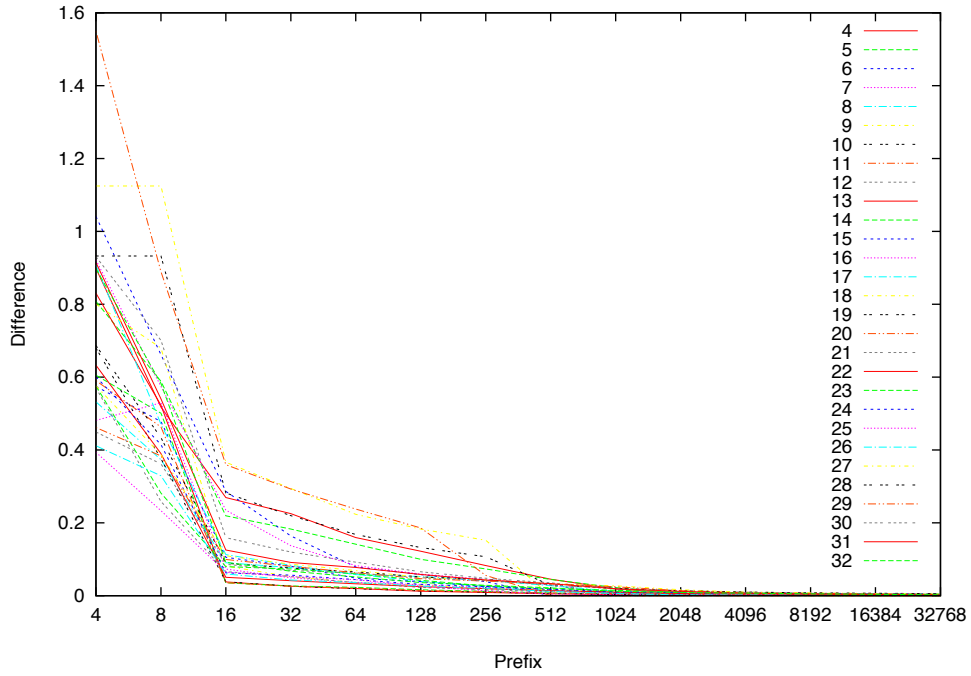
Let us consider the convergence of feature values more closely. Figures D.1 - D.5 in Appendix D present distribution differences within each class. As shown in the figures, the convergences of the distribution differences are relatively similar within the same class but differ greatly among classes. This could be explained by the average length of the flows in each class (see Table 6.1). The distributions corresponding to the classes consist of large flows and take more time to converge, which means that the flows need to be observed longer in order to effectively estimate their true characteristics. For the Strict Conversational class, the differences of features drop to < 0.2 at around prefix of length 512, which is much lower than average flow length of 2165.81. This is also true for the Streaming class whose average flow length is 3880.90 while the differences get lower than 0.2 at around prefix of length 512.

Figure D.6 - D.34 illustrate the convergences of each feature. Each graph in each figure represents the distribution difference with respect to number of prefix and class, i.e., the graph is obtained from the values of a feature extracted from the flows belonging to the same class. This way, one can analyze how the flows of different classes

[3]The overall area under the two distributions is 2 (each of them has an area of 1). Therefore, difference of 0.4 from overall area of 2 is equal to 0.4/2 or 20% estimation error.



(a)



(b)

Figure 6.5: Difference/Prefix plots of all features computed using all flows in WIDE dataset. The number of each plot indicates which feature is associated to the plot. It corresponds to the numbers of the features listed in Table 5.1.

Table 6.1: Average length of flows in each class.

Class	Average Flow Length
StrConv	2165.82
RlxConv	12.68
Streaming	3880.90
Interactive	29.40
Bulk	11.00

behave with respect to different prefixes. From the figures, it could be established that, considering the convergence of feature values, some features are more suitable for some specific classes while some features are not. For instance, the statistics of packet sizes of the flows in Strict Conversational, Interactive and Bulk classes (such as, average, SD, and RMS) are relatively consistent throughout the flows. Consequently, the differences of the feature values are low regardless of how many prefixes are used to calculate the feature values (see Figure D.15, D.16, and D.19 - D.21). On the other hand, for streaming flows, those features do not converge as quickly whereas the packet inter-arrival time (IAT) and its statistics do. The distribution differences of the corresponding feature values are very low (see Figure D.27 - D.34). This means that the packet sizes of streaming flows often vary while packet IATs are steady. This phenomenon is consistent with previous findings. Kuang and Williamson [KW02] have shown that media streaming using variable-bit-rate (VBR) codecs such as RealMedia [Rea08] have fluctuated bit rate at short timescales whereby the fluctuation appears only in the packet sizes, not in the inter-arrival times [RSSD04]. Other works including [GW94] and [BSTW95] also report similar behaviors.

Our study here provides a useful insight of how features behave over observed flow length in an entirely new aspect that has not been carried out before in the literature. In the following sections, we will see if the prefix and the distribution difference also affect the classification accuracy. Furthermore, we will also see if the features that do not converge can still be useful in flow classification.

6.5 Prefix and Accuracy

As we have emphasized earlier, real-time classification requires a FCS that can classify flows within a restricted period of time. Up to now, only little research in real-time flow classification, such as [BTA$^+$06], [NA06], and [EMA$^+$07], has been carried out with high accuracy and the flows can be classified after some restricted number of packets is observed. Due to the choices of features, however, their methods always require coflows and none of them can classify UDP flows. There is therefore an obvious need for a real-time classification technique that addresses those limitations. In the following sections, an analysis of the relationships between observed flow lengths and the accuracies will be carried out. Firstly, a formal definition of the shortest flow length will be given. Secondly, we will determine the relationship of the prefixes and classification accuracies. Finally, further analysis on the smallest set of features that

constitute the shortest prefix will be presented. With such analysis, the features that are actually important and useful in real-time flow classification can be identified.

6.5.1 Saturation Point

In a non-real-time classification setting, the learner induces a classifier from a set of data, which is generated by a dataset generator function G corresponding to a set of features $\mathcal{V}$. With G and $\mathcal{V}$, all packets in the flows are used by the feature functions to generate feature vectors. In real-time classification, instead of using the entire flow, only l prefix is considered and a length-restricted dataset generator function G' and a set of length-restricted feature functions $\mathcal{V}'$ are employed. Because we want to classify the flow as soon as possible, the smallest l is preferred. Nevertheless, certain classification accuracy has to be maintained. The following is the formal definition of the shortest prefix required by a learner to induce a classifier that can achieve the given accuracy.

Definition 6.10 (Saturation Point) Let $\mathcal{F}$ be a set of flows, $\mathcal{C}$ a set of service classes, $\mathfrak{D}$ set of all datasets, $G' : \mathcal{P}(\mathcal{F} \times \mathcal{C}) \times \mathbb{N}^+ \rightarrow \mathfrak{D}$ a length-restricted dataset generator, $\mathcal{K}$ a set of classifiers, and $K \in \mathcal{K}$. Given a learner $L : \mathfrak{D} \rightarrow \mathcal{K}$, a threshold $\alpha \in \mathbb{R}$, and $\mathcal{D} \in \mathfrak{D}$, a flow length $l \in \mathbb{N}^+$ is said to be the *saturation point corresponding to* α, denoted l^*_α, iff l is the minimal flow length such that

$$Accuracy(K, \mathcal{D}) \geq \alpha$$

where $K = L(G'(\mathcal{R}, l))$.

The flow length l is called the saturation point if the accuracy of the classifier induced from the l-dataset reaches the given threshold α. The l-dataset is generated from a given raw dataset $\mathcal{R}$ by the length-restricted dataset generator G'. The learner then uses the generated dataset to induce a classifier, which will later be evaluated on a separate test set $\mathcal{D}$.

6.5.2 Determining Preferred Accuracy

The results of the evaluations conducted in Chapter 5 show that all learners except Naive Bayes, using entire flows and all feature functions, could achieve more than 99% accuracy (see Table 5.3, 5.10, and 5.20). Also, in Section 6.2.1, we have learned that the feature values converge on true characteristics, though fast convergences do not imply better classification accuracy. This, in turn, begs the question whether the learners can still achieve such performances using only partial flows. To answer this question, a series of analyses — described below — are carried out.

In the following investigations, series of datasets are generated the same way described in Section 6.4. That is, for a given raw dataset $\mathcal{R}$, series of datasets similar to those in (6.7) and (6.8) are generated. We extend the analysis conducted earlier by inducing classifiers from datasets by feeding them into a learner to obtain the following sets of classifiers:

$$\begin{aligned} L(\mathcal{D}_{[4],\mathcal{R}}), L(\mathcal{D}_{[5],\mathcal{R}}), \ldots, L(\mathcal{D}_{[20],\mathcal{R}}) &\Rightarrow K_{[4],\mathcal{R}}, K_{[5],\mathcal{R}}, \ldots, K_{[20],\mathcal{R}} \\ L(\mathcal{D}_{[2^2],\mathcal{R}}), L(\mathcal{D}_{[2^3],\mathcal{R}}), \ldots, L(\mathcal{D}_{[2^{16}],\mathcal{R}}) &\Rightarrow K_{[2^2],\mathcal{R}}, K_{[2^3],\mathcal{R}}, \ldots, K_{[2^{16}],\mathcal{R}}. \end{aligned}$$

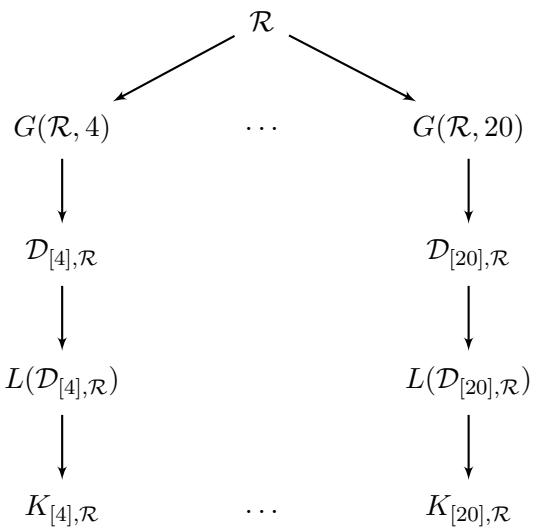


Figure 6.6: Inducing classifiers from the datasets corresponding to different prefixes. From the same raw dataset, multiple length-restricted datasets are generated. Here, the length grows linearly from 4 to 20 as the same learner L is used to induce the classifiers. The only difference among classifiers are thus the prefixes corresponding to them. In an experiment not shown here, another series of classifiers is induced by increasing the prefix exponentially. Also, in the actual experiments, 10 raw datasets are used in order to avoid any biases.

Figure 6.6 illustrates the classifier induction process. The linear and exponential series consist of 17 and 15 classifiers, respectively. To investigate whether the classifiers induced from partial flows can be used to classify full-length flows, they are evaluated on the dataset that is extracted from the entire flow (i.e., the $\mathcal{D}_{\text{full},\mathcal{R}}$). Five learners evaluated in the previous chapter — including J4.8 (an implementation variant of C4.5), RIPPER, PART, Naive Bayes, and k-Nearest Neighbor — will be evaluated again here, leading to five groups of classifiers for each raw dataset with each corresponding to a learner. Each group consists of two series of classifiers that are linearly and exponentially generated. Furthermore, to avoid any biases from the uncertainty, the flows are randomized 10 times resulting in 10 different raw datasets. Thus, in our experiments, including all 10 raw datasets,

$$5 \times 10 \times (17 + 15) = 1,600$$

classifiers are considered. The exact experiment steps are described in the following listing:

Randomize $\mathcal{R}$ into 10 small subsets $\mathcal{R}_1, \ldots, \mathcal{R}_{10}$.
for $j = 1$ to 10 **do**
  generate $\mathcal{D}_{\text{full},\mathcal{R}_j} = G(\mathcal{R}_j)$
  for $l = 4$ to 20 **do**
    generate $\mathcal{D}_{[l],\mathcal{R}_j} = G'(\mathcal{R}_j, l)$
    induce $K_{[l],\mathcal{R}_j} = L(\mathcal{D}_{[l],\mathcal{R}_j})$
    evaluate $a_{j,l} = Accuracy(K_{[l],\mathcal{R}_j}, \mathcal{D}_{\text{full},\mathcal{R}_j})$
  end for

end for

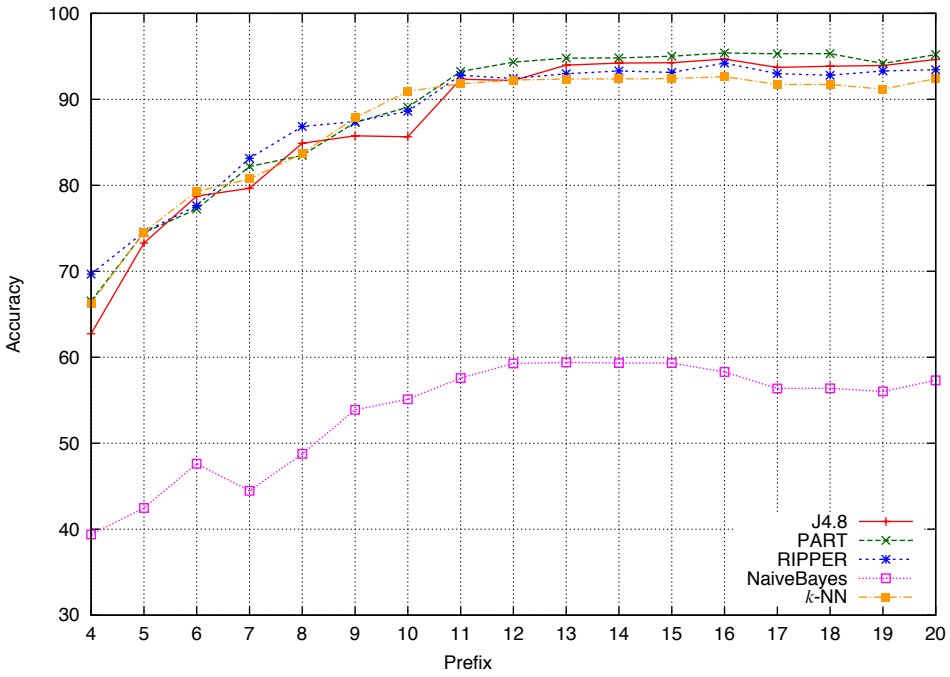
find the average accuracies of $a_{j,4}, \ldots, a_{j,20}$, $1 \leq j \leq 10$

The listing above describes the experiment process corresponding to the linear-increment dataset series. Nevertheless, as in the previous experiments, the evaluation is conducted for exponentially-increasing prefixes (from prefix 2^2 to 2^{15}) as well.

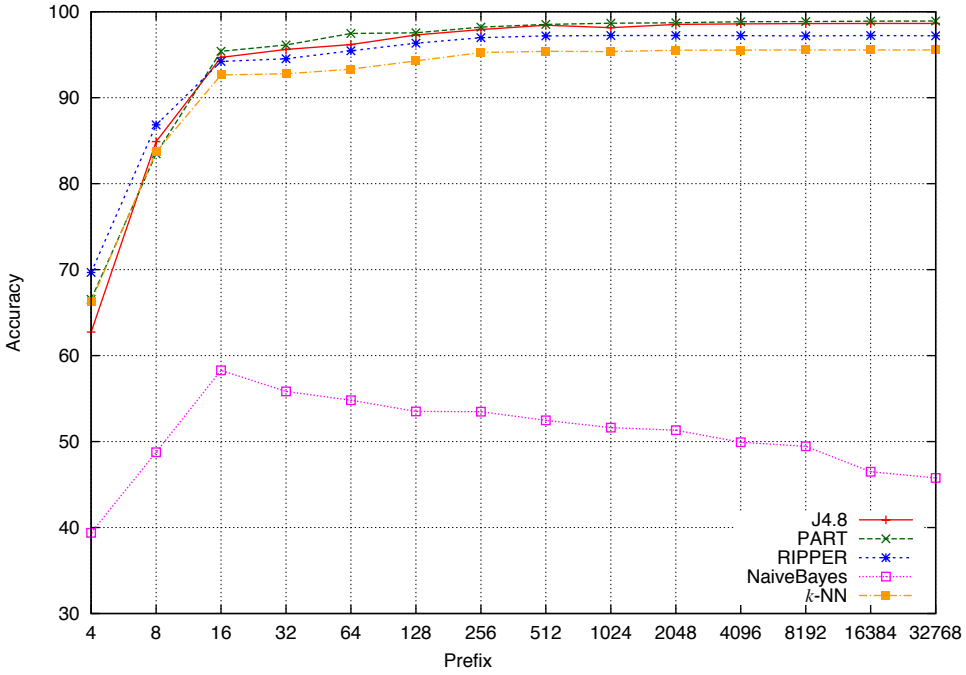
Figure 6.7 reports the results of both evaluations. As shown in the figure, after prefix 11, all learners except Naive Bayes achieve more than 90% accuracy. As shown in Figure 6.7(b), the accuracies stay above 90% afterwards. After prefix 128, the accuracies go above 95% and remain so throughout the evaluations. In other words, if $\alpha = 90\%$, the saturation point $l^*_\alpha = 11$. It is important to note that the saturation point is the *earliest* point that the preferred accuracy α is reached. Thus, it is not required that the accuracy has to monotonically increase proportional to the prefixes. The results suggest that, after prefix 11, an acceptable accuracy is already realizable and we do not have to observe the flow longer than 128 packets as it will not enhance the performance any further. This means that accurate real-time classification is possible within only 11 packets, according to our experiments. Moreover, unlike previous real-time classification approaches such as [BTS06] and [EMA+07], our method, thanks to its employed features, does not require coflows and can classify both TCP and UDP flows.

The evaluations above assess the performance of the classifiers with respect to the prefixes. The objective is to determine whether the learners can tolerate the variation of values caused by the partial-flow observations. In practice, however, this is not necessarily the case because, when the classifier classifies an unseen flow, it is also observed only partially. In other words, since the classifier is induced from the dataset of length l, there is no need to observe the unseen flow longer than l. It is thus interesting to see how the classifiers induced from a training set of length l perform over the test set of the same length. Consequently, we conduct another set of evaluations, in which the classifiers are induced from l-datasets and evaluated on the same sets using the cross-validation method.

Figure 6.8 reports the experiment results, which indicate that the performance of all classifiers except Naive Bayes are extremely high regardless of the prefix used. This is, nevertheless, not surprising as the classifiers are optimized for specific flow lengths. Erman et al. [EMA+07] utilize this property and propose a real-time classification method that employs multiple classifiers, which are trained by the training sets of some particular lengths. After l packets of a flow are observed, the classifier that corresponds to l will be invoked to classify the flow. One advantage of such method is that the classification system benefits from the classifiers specifically optimized for particular flow lengths. Such approach is, however, rather redundant. As shown in Figure 6.8, with an appropriate learner, high classification accuracy can be achieved right from the beginning of observation. Furthermore, even if the entire flow is concerned, Figure 6.7 shows that observing flows after a certain point would not increase the accuracy any further. The performance of Naive Bayes in fact worsens as prefixes are increased. This is true when we test the classifier with full flow length as well as in specific prefixes alike. We believe that, as the prefix increases, different flows (within the same classes) exhibit higher dissimilarities, which Naive Bayes cannot handle effectively. Also, it

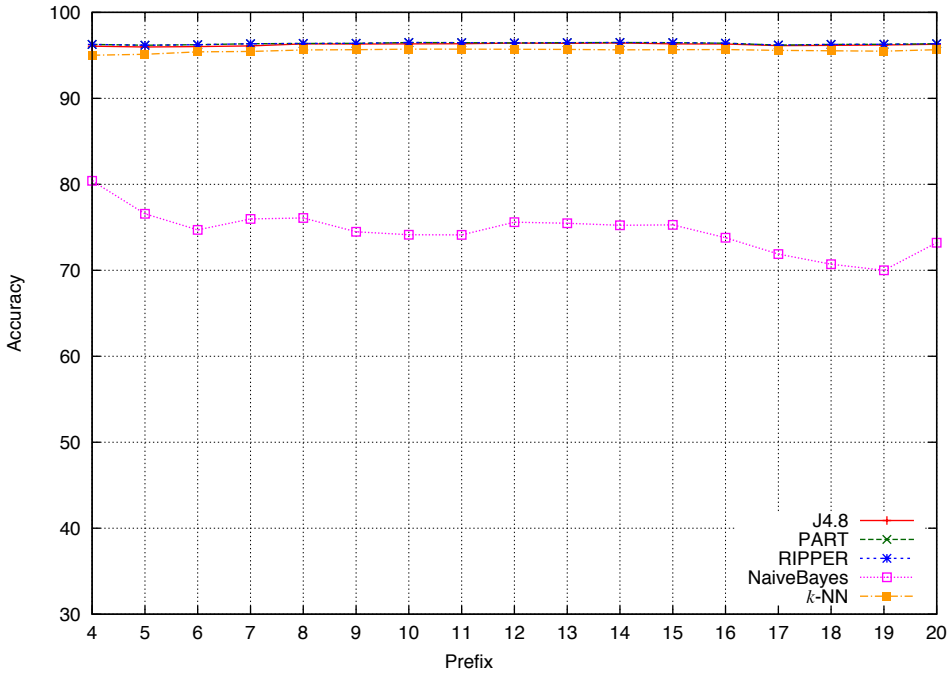


(a) Linear Scale

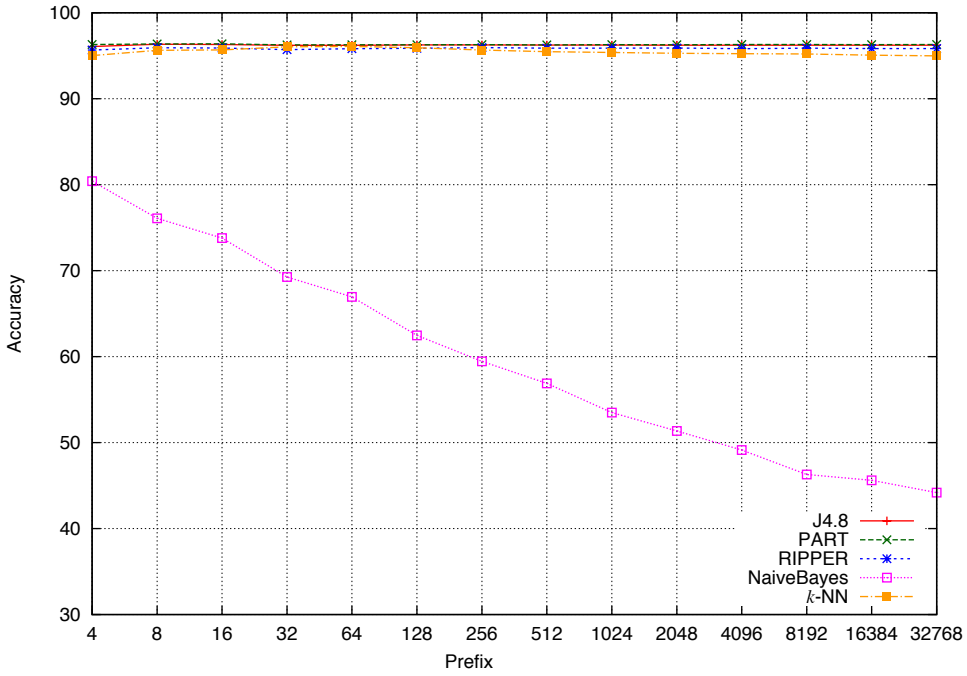


(b) Exponential Scale

Figure 6.7: Accuracy/prefix plots - Tested on full flow lengths.



(a) Linear Scale



(b) Exponential Scale

Figure 6.8: Accuracy/prefix plots - Tested on the same prefixes.

could be contributed by the bias of the learner, which assumes the independency of features.

6.6 Selecting Features with Respect to Prefixes

In the previous sections, we have established that the high accuracy can be achieved using only partial flow and the smallest prefix that yields more than 90% accuracy is 11. Here, we will investigate further which features are actually "relevant" with respect to classification accuracy and required prefix. The goal of the investigation is to identify which features actually contribute to the classification and to determine if they do so for all prefixes.

We will begin with an introduction to "features selection algorithms", which are techniques designed to select a set of optimal features from the whole set of features with respect to a certain criterion. Then, we will employ one of them to determine which features are relevant. Finally, the selected set of features will be evaluated to see if they really are useful.

6.6.1 Introduction to Feature Selection

Feature selection is a method to select a subset of relevant features from a given set of features with respect to a given criterion. It is one of the most active fields in machine learning community because, by removing redundant and irrelevant features, the learning time could be reduced and the learned concept could be more general leading to a better classification performance [DL97]. Feature selection plays very important roles in many learning tasks, especially the ones involving large sets of features such as, text categorization [NMTM00][LK02][YP97], image retrieval [RHC99][SW95], signal processing [TLY04][Thi07], and genomic analysis [XJK01][YL04].

In essence, feature selection is an instance of a search problem, where each state in the search space is associated to a candidate feature subset and the search objective is the best combination of features according to the given criterion [SS98][LM98]. Like other search problems, it consists of three main aspects: search strategy, evaluation criterion, and stopping decision. The search strategy determines which feature subset to be evaluated at each step. It is usually referred to as subset generation process. After a subset of features is selected, it will be evaluated by an evaluation function, which measures the "goodness" of the set. If the current set of features is better than the best one seen so far, the current one will be kept instead. The process of select-and-evaluate could be iterated until a stopping criterion is met. Feature selection process is depicted in Figure 6.9.

The feature selection algorithms can be divided, according to their evaluation criteria, into "wrapper" and "filter" models [Lan94][KJ97]. In the wrapper model, a predetermined learner is used to induce a classifier using the dataset with selected set of feature. Then, the performance of the induced classifier will be used as the evaluation criterion. The advantage of this approach is that one will get a feature set that works best with the given learner. Filter approaches, on the other hand, select the features based on other criteria, which are independent from the learner. Because, in the evaluation process, filter approaches do have to wait for the learner to induce a classifier, they are generally faster than the wrapper ones.

In our research, a filter method called Correlation-Based Feature Selection (CFS)

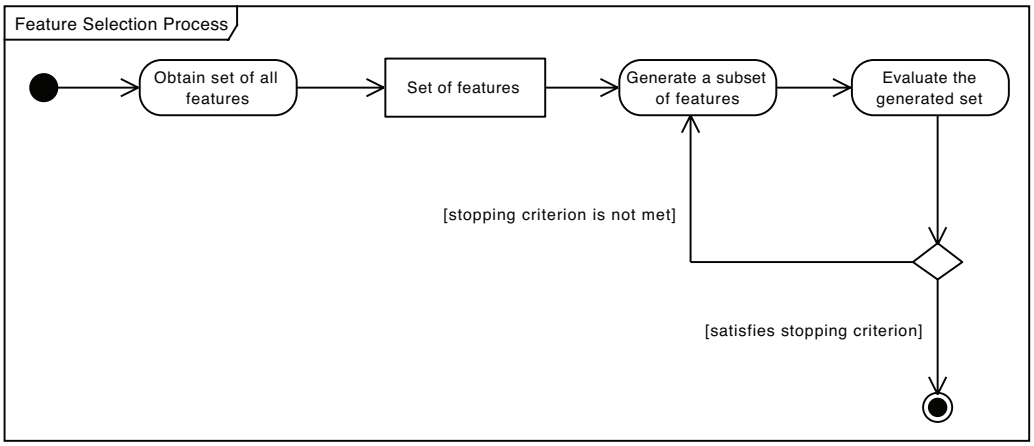


Figure 6.9: Feature Selection Process.

[Hal98][Hal00] is employed to select the optimal set of features. CFS measures the goodness of a feature based on rationale that good feature subsets contain features highly correlated with the class, yet uncorrelated with each other [Hal98]. This means that CFS measures the goodness of feature sets entirely and not only individual features. According to [Liu05], the CFS algorithm suits best to our need as it is aimed directly at eliminating irrelevant and redundant features. This would provide us a better insight to characteristics of the feature set. The method was also employed by Williams et al. [WZA06] to evaluate their sets of features. Also, as a filter algorithm, it is independent to any specific learner. More importantly, it is shown to be superior over other selection techniques [Hal00][LLW02]. The CFS algorithm and its selection criteria will be explained in the following section. Further information in feature selection can be found in, e.g., [DL97], [LM98], [GE03], and [Liu05].

6.6.2 Determining the Smallest Set of Features

The Correlation-based Feature Selection method or CFS evaluates the goodness of feature sets based on a test theory developed by Ghiselli to design the most effective composite test (i.e., a set of tests) for predicting an external value [Ghi64]. The rationale behind the theory is that the values of the features in the set should be correlated to the classes while they should not be correlated to each other. According to Hall [Hal98], the most suitable criteria to estimate degree of correlation between two random variables X and Y is the "symmetrical uncertainty", given by:

$$SU(X,Y) = 2.0 \times \left(\frac{Gain(X,Y)}{H(X) + H(Y)} \right)$$

where H and $Gain$ are entropy and information gain functions, respectively. The entropy and information gain correspond to those explained in Section 4.1. Intuitively, the symmetrical uncertainty measures the information gain of two variables normalized by the entropy of each variable (in our case, the variables are either a pair of feature and feature or feature and class) — similar to the concept of information gain used in C4.5. Now, given a set of features $\mathcal{V} = V_1, \ldots, V_d$, the goodness of $\mathcal{V}$ is given by

[Hal98]:

$$Eval_{\mathsf{CFS}}(\mathcal{V}) = \frac{d \times r_{\mathsf{class}}}{\sqrt{d + d(d-1) \times r_{\mathsf{features}}}}$$

where r_{class} and r_{features} are the averages of feature-class and feature-feature correlations respectively. In the feature-feature case, the correlation is calculated pair-wise among all features in the feature set. While symmetrical uncertainty is employed as the evaluation criterion, best-first search strategy is employed as the search strategy.

To determine the most relevant features, we apply CFS algorithm on the datasets generated from WIDE packet traces, which were used before in previous evaluations. Then, we will see how often the features are selected among all classes. The results will be shown in percentage — if a feature is selected four times out of 10 datasets, its selection rate is 40%. We will begin by applying the feature selection algorithms on datasets generated from full flow length. Then, we move on to scrutinizing the features that are selected from l-datasets generated from prefixes of various lengths.

Table 6.2 presents the results of feature selection using CFS. As shown in the table, 10 features are always selected from 10 different sets sampled from the entire packet traces. These features include *protocol*, *connTime*, *dataVolume* and its coflow counterpart, *pktCount* and related statistics, and statistics of packet sizes. It is, however, not always the case in short-prefix scenarios. As shown in Table 6.3, only *protocol*, *pktCount*, and statistics of packet sizes are selected. This means that connection time, data volume, and overall and ratio of number of packets are less discriminative when the flows are not fully observed. This makes sense because, within only short observation period, differences of connection time and total data volume are not apparent. The sizes of the packets, on the other hand, can be consistently used to classify flows with different classes. This observation is consistent with other researches that suggested that packet sizes could be effectively used to distinguish flows from different classes [WZA06][BTS06]. It is important to note that our new feature, *pktSizeDiff*, is also always selected. Interestingly, packet inter-arrival time and its statistics are rarely selected. According to an inspection, this is because the distributions of inter-arrival times are relatively similar among all classes. (Although, as discussed in Section 6.4, they tend to converge early in Streaming class.) Such phenomenon was also discerned by Bernaille [Ber07]. Other studies including [EMA+07] and [WZA06] also obtain the similar results. This finding may shine negative light on classification approaches which are based mainly on inter-arrival time, such as [MLK06]. We would like to emphasize that evaluations conducted here are focused on the relationship between prefix and accuracy, which has not been carried out before in the literature.

6.6.3 Applying the Selected Features in Real-Time Classification

In the previous section, the smallest set of features containing the most relevant and non-redundant features is determined. The features that are consistently discriminative throughout various prefixes are also identified. Here, we will see whether the selected features can be used to discriminate the flows. To do so, the experiments that are carried out earlier will be repeated again using only the selected set of features. More precisely, we will let the learners induce classifiers using the datasets generated

Table 6.2: Features selected by CFS algorithm from 10 set of full-flow-length datasets. The percentage in the second column indicate the rate that the features are selected from different sets. In total, out of 31 features, 10 are always selected. Notice that connTime, dataVolume, and pktCount and its statistics are always selected.

Selected Features	Selection Rate
protocol	100%
connTime	100%
dataVolume	100%
dataVolumeCF	100%
pktCount	100%
pktCountTotal	100%
pktCountRatio	100%
pktSizeSD	100%
pktSizeSDCF	100%
pktSizeRMS	100%
pktSizeDiff	90%
avgPktSize	40%
iatVar	40%
pktSizeRMSCF	30%

Table 6.3: Features selected by CFS algorithm from 10 set of datasets generated from prefix 4-20. The features that are selected less than 50% are discarded.

Selected Features	Selection Rate
protocol	100%
pktCount	100%
pktSizeDiff	100%
pktSizeSD	100%
pktSizeRMS	100%
pktSizeDiffCF	98.24%
dataVolume	88.82%
pktSizeRMSCF	65.88%

from different prefixes. Then, as before, the induced classifiers will be evaluated on the datasets generated from full flow lengths. This time, however, only the selected features, which are listed in Table 6.3, are utilized.

Figure 6.10 presents the graphs of accuracy over number of prefixes. As shown in the Figure, performance of Naive Bayes is dramatically increased for more than 50% at almost every flow length. Such improvement may be contributed by the strong bias of Naive Bayes: As it assumes independence among features, removing redundant features should improve the performance of the algorithm [LS94]. To this aspect, CFS would be a perfect complement to Naive Bayes as it selects only features that are the least-correlated to each other [Hal98]. Apart from that, despite CFS's aggressive feature reduction, accuracies of other learners are similar to those using full set of features (see Figure 6.7(a)). Not only that the learners can still maintain their outstanding performance, between prefix 9 and 11, the accuracies of all learners are actually improved. At prefix 10, RIPPER, PART, and k-NN already reach 90% line, followed closely by J4.8 with 88.42%. This shows that, the features are discriminative, not only at any specific prefixes, but throughout the flow. Furthermore, removing irrelevant and redundant features results in more meaningful data, which further leads to an improved learning capability.

The major benefit from the feature selection, however, is not the accuracy of the classifiers, but the learning time. As the features that have to be concerned by the learners are reduced, the learning time is also reduced. This is consistent with the theoretical computational time. C4.5 decision tree algorithm, in particular, requires $O(dn \log n)$, where d is number of features and n is the number of instances, to build a decision tree [FW98]. The reason why its complexity is closely related to the number of features is that it has to compute the gain ratio of each feature every time it expands a node in the tree. As a result, as shown in Table 6.11, 72.65% improvement in computational time is gained. Naive Bayes method, whose complexity is linear in the number of features and instances, also benefits directly from the feature reduction. Its learning time is reduced by 81.70%. Algorithms whose time complexity does not directly depend on the number of features also benefit from the reduction. The learning time of PART, which requires $O(kdn \log n)$ time[4] to construct a rule set of size k [FW98], reduces to 64.78%. RIPPER, whose time complexity corresponds to number of instances ($O(n \log n)$) [Coh95], also gains 27.31% improvement because it requires less time to compute much fewer feature values.

As shown in Figure 6.10, the accuracies of J4.8, PART, and RIPPER are similar, but the learning time required by J4.8, which is an implementation of C4.5, is much smaller than other algorithms — It takes only 1.2 seconds to learn from dataset of 20,000 instances. Therefore, we select C4.5 as the learner for our SMART flow classification system.

In this section, we have identified most relevant and non-redundant features, which are the core features that contribute most to the classification. Our evaluations revealed that flows can be characterized by statistics the packet size, while packet IATs

[4]The time complexity of PART is rather similar to that of C4.5 because it uses a decision tree to generate each rule.

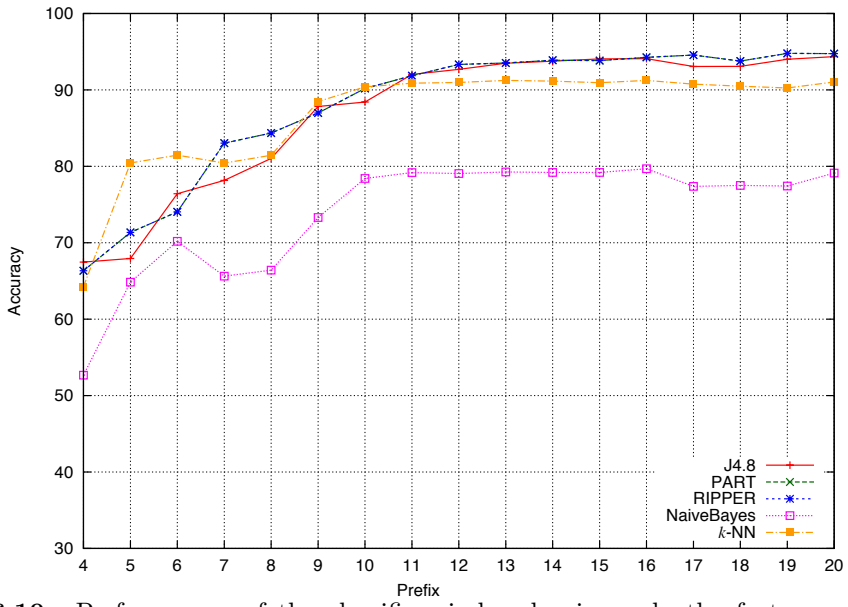


Figure 6.10: Performances of the classifiers induced using only the features selected by CFS. All classifiers maintain their performances despite aggressive feature reduction. Overall performance of Naive Bayes improves significantly. Also, between prefix 9 and 11, the performance of other algorithms noticeably increased — the accuracy of RIPPER, PART, and k-NN reaches 90% at prefix 10.

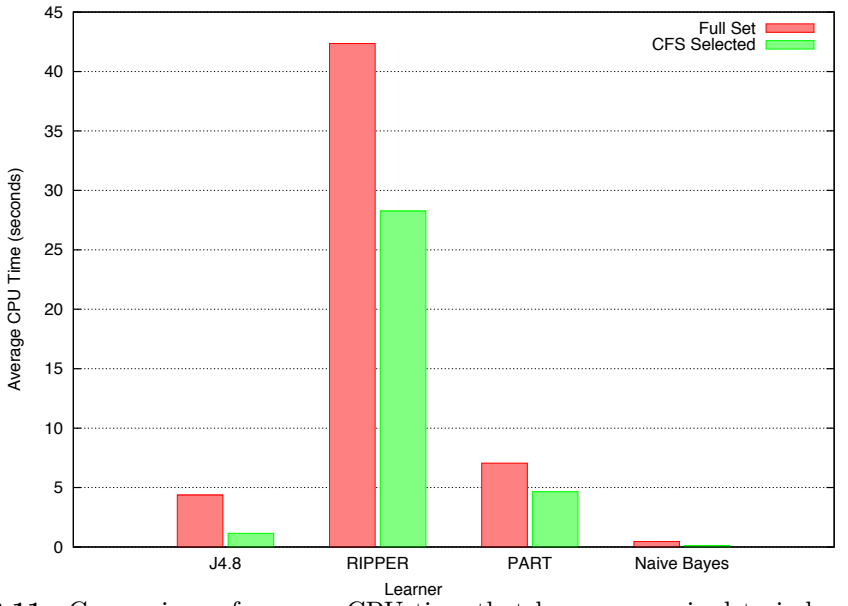


Figure 6.11: Comparison of average CPU time that learners required to induce classifiers between full and the selected feature sets. The CPU time of the full set correspond to that presented in Figure 5.5. As shown in the figure above, the computational time of all algorithms, especially J4.8, are improved dramatically. Again, as k-NN is a lazy algorithm, it requires no learning time.

Table 6.4: Comparison of the time required to induce classifiers. The average CPU time corresponds to the full set and the selected set of features is presented along with the ratio between learning time before and after applying feature selection.

Learners	CPU Time (seconds)		Difference (%)
	Full Feature Set	CFS Selected	
J4.8	4.37	1.20	72.65
RIPPER	42.35	27.44	27.31
PART	7.06	5.13	35.22
Naive Bayes	0.47	0.09	81.70

and other features are as useful. Also, as shown in our experiment results, using only appropriate features leads to in better classification performance, especially for Naive Bayes, as well as much shorter learning time.

6.7 Conclusion

In Chapter 5, we have shown that, using a machine learning algorithm, adaptability and self-updatable facilities can be integrated into flow classification system. In this chapter, we have extended our study further to the real-time aspect of flow classification by focusing on possibility of determining the flow characteristics using only specific prefixes. In doing so, we have analyzed the differences between the feature values distributions observed from various prefixes and the full flow lengths. The results show that the distributions corresponding to different prefixes tend to converge on the true distribution and the characteristics of a flow might be able to be captured without observing it entirely. Such investigations have never been carried out before in the literature. Apart from the distribution convergence, we have shown through a series of empirical evaluations that the classifiers induced from length-restricted datasets can be effectively used to classify full-length flows. With all learners except Naive Bayes, 90% accuracy can be achieved using dataset of length 11. Also, novel features that can be computed at any prefix, *pktSizeDiff* and *pktSizeDiffCF*, were introduced. We have employed a feature selection method, CFS, to select features that are relevant to the class and non-redundant to each other. It is shown in our experiments that our new features are relevant to the classification at every prefix. Thanks to the much-reduced size of the selected set, the learning time is dramatically decreased. The accuracies are also improved, especially that of Naive Bayes.

Chapter 7

Conclusion and Future Work

As discussed in Chapter 1, a flow classification system (FCS) assisting quality-of-service (QoS) support has to be accurate, robust, generic, and able to operate in real-time. Due to the dynamic nature of the Internet, to which new applications are constantly introduced, static FCSs, such as port-based and signature-based FCS, cannot be used effectively. Furthermore, most of the current flow-behavior-based FCS does not address real-time classification, which is necessary for QoS-support. Although FCSs such as [BTA+06] and [EMA+07] can carry out the classification using only partial flows, they do not support UDP connections. This thesis presents Supervised Machine learning Assisted Real-Time (SMART) Flow Classification System, an innovative real-time adaptive flow classification system, that addresses all of these limitations.

This chapter summarizes the contributions of our research as well as discussion on its limitations and future research directions.

7.1 Contributions

SMART, a real-time adaptive FCS has been introduced. SMART is suitable for QoS-support scenario as addresses all the requirements listed in Chapter 1:

- **Accuracy**: Evaluation results presented in Chapter 5 and 6 show that our methodology can be used to classify unseen flows with high accuracy. This is true in both individual users and large-network cases. The main reason behind such impressive results is the choice of features, which can be used to effectively distinguish the flows.

- **Robustness**: Thanks to its equipped learner, the FCS can learn and adapt itself to recognize unseen flows without human intervention. Evaluation results show that machine learning can really be used to in a flow classification scenario.

- **Generality**: Unlike previous works, our technique is designed from the start to support UDP flows. It relies only on the common information available in the IP and transport protocol headers. Therefore, we are convinced that it can be used in any TCP/IP networks, including the upcoming IPv6 network.

- **Real-time capability**: The strongest advantage of our method is its real-time classification capability. With the innovative concept of flow prefix, we have shown that only 11 packets are sufficient to achieve more than 90% accuracy. Our analysis also shows that the distributions of feature values converge on true distributions and it is not necessary to observe the entire flows to obtain the true characteristics.

A unified framework for flow classification. In the literature, all previous flow classification systems are developed separately without a general model that can describe them. We have proposed instead a solid unified mathematical model, which describes the flow classification components and processes on an abstract level. As a result, not only can our model describe and compare the previous approaches, but it can also precisely define the components of our FCS, especially the feature functions. Apart from mathematical model, the implementation, deployment and system decomposition of SMART are also described using Unified Modeling Language (UML). Thus, our research and experiments can be easily and correctly replicated.

Relevant features and appropriate prefixes are identified. Empirical evaluations carried out in this thesis show that packet size and its statistics, including our new features, throughput difference and the packet size difference, are discriminative features, whereas, interestingly, packet inter-arrival time is not. Such phenomenon holds across all prefixes evaluated in our experiments. Furthermore, as mentioned above, we are able to pinpoint that 11 packets are enough to deliver an outstanding classification performance.

The best candidate for flow classification system is identified. Our analysis suggests that C4.5 offers the best trade-off between classification accuracy and computational time. In non-real-time classification, it achieves up to more than 99% correctness in both individual-user and large-network evaluations. In real-time case, its accuracy reaches 92.34% after observing only 11 packets, while taking only 1.2 second to learn from a dataset of 20,000 instances. Also, its induced classification model, a decision tree, is a white-box classifier, which is easy to comprehend and utilized.

SMART is evaluated on end-user data. None of the existing flow classification systems are aimed for end-user devices and, to the best of our knowledge, none has ever collected and performed experiments on individual-user flow data as we have done here.

A complex analytical framework for flow classification is developed. To make sure that our experiments are reliable, a sophisticated signature-based FCS is developed to verify the ground-truths of our data.

7.2 Future Research Directions

Although SMART, our new flow classification system, has so far proven to be highly accurate, it is not without limitations. This section provides a discussion on such restrictions as well as some insights into potential future research.

Embedded device implementation Although SMART is shown to be feasible on a user device and can capture and process live packets seamlessly, our current system is still running only on a personal computer. An actual system that is deployed in embedded devices such as routers and smartphones is yet to be implemented.

Improving cross-user performance From the individual-user and large-networks evaluations, our technique has empirically proved to be very usable. However, in case of cross-user classification, the performance is still inadequate. We believe that, by pre-processing the feature values or adjusting the learner to produce a more generic classifier, a lucid cross-user classification can be realized.

Automatic prefix adjustment We have so far focused on a single prefix that is the most appropriate observation point. Although such approach has been shown to be useful, it should be possible to identify some applications even earlier. A flow classification system that can determine an appropriate prefix at runtime without human intervention would indeed be advantageous.

Flow with dynamic QoS-requirements In our research, we have assumed that each flow has only one class and will not be changed. It does not recognize the flows that change their QoS requirements during the course of the communication. This issue may be partially circumvented by deploying multiple classifiers with each corresponding to a specific prefix. Once the flow reaches such prefix, the classifier will be activated to reclassify the flow. As long as the appropriate classifier is not activated, the new class will not be assigned to the flow.

Extending our methodology to other domains SMART is aimed directly at assisting QoS-support. We believe though that it can also be employed in other flow classification domains such as network security and administration. Moreover, the proposed methodology is designed to be used only in networks that are based on DiffServ model. Studying the possibility and feasibility of extending our work to IntServ networks would be a compelling and challenging research.

Appendix A

Machine Learning Algorithms

Algorithm A.1 C4.5

Procedure C4.5($\mathcal{D}, \mathcal{V}, \mathcal{C}$)
Input: $\mathcal{D}$: Dataset
$\mathcal{V}$: Set of features
$\mathcal{C}$: Set of classes
Output: T: Decision Tree

1: $T \leftarrow$ Empty tree.
2: **if** For all $\langle \boldsymbol{x}, c \rangle, \langle \boldsymbol{x}', c' \rangle \in \mathcal{D}, c = c'$ **then**
3: Construct a node N with label c
4: Add N to T
5: **return** T
6: **end if**
7: **if** $\mathcal{V} = \emptyset$ **then**
8: Construct a node N with label c^* where c^* is the dominant class in $\mathcal{D}$
9: Add N to T
10: **return** T
11: **end if**
12: $\mathcal{V}_{\text{original}} \leftarrow \mathcal{V}$

▷ Continue on the next page.

13: **for all** $V \in \mathcal{V}$ **do**
14: **if** codomain D of V is continuous **then**
15: Extract unique values of feature V in $\mathcal{D}$ and store in a sequence s
16: Sort the sequence s
17: $MaxGain \leftarrow 0$
18: **for all** component s_i in s **do**
19: Create a feature $\hat{V} : \mathcal{F} \to \{[-\infty, s_i], (s_i, +\infty]\}$
20: $Gain \leftarrow GainRatio(\mathcal{D}, \hat{V})$
21: **if** $Gain > MaxGain$ **then**
22: $\hat{V}_{\max} \leftarrow \hat{V}$
23: **end if**
24: **end for**
25: Replace V with $\hat{V}_{\max}$ in $\mathcal{V}$
26: **end if**
27: **end for**
28: $V_{\max} \leftarrow \underset{V \in \mathcal{V}}{\operatorname{argmax}} GainRatio(\mathcal{D}, V)$
29: Construct a node N with label $V_{\max}$
30: **for all** $x \in D$ **do**
31: Add new branch with label x below N
32: $\mathcal{D}^x \leftarrow \dot{\{}\langle x_1, \ldots, x_j, \ldots, x_d, c\rangle \in \mathcal{D} \mid x_j = x\dot{\}}$
33: **if** $\mathcal{D}^x = \emptyset$ **then**
34: Construct a node N' with label c^* where c^* is the dominant class in $\mathcal{D}$
35: Add N' to the branch
36: **else**
37: $\mathcal{V} \leftarrow \mathcal{V}_{\text{original}}$
38: Add a subtree C4.5($\mathcal{D}^x$, $(\mathcal{V} \setminus V_{\max})$) to the branch
39: **end if**
40: **end for**
41: **return** N

Algorithm A.2 RIPPER

Procedure RIPPER($\mathcal{D}$)
Input: $\mathcal{D}$: Dataset
Output: R: Set of rules

$R \leftarrow \emptyset$
$d \leftarrow 64$
$SmallestR \leftarrow 0$
split $\mathcal{D}$ into $\mathcal{D}^c$ and $\mathcal{D}^{\bar{c}}$
while $\mathcal{D}^c \neq \emptyset$ **do**
    randomly split $\mathcal{D}^c$ into $Grow^c$ and $Prune^c$
    randomly split $\mathcal{D}^{\bar{c}}$ into $Grow^{\bar{c}}$ and $Prune^{\bar{c}}$
    $NewRule \leftarrow []$
    $NewRule \leftarrow GrowRule(NewRule, Grow^c, Grow^{\bar{c}})$
    $NewRule \leftarrow PruneRule(NewRule, Prune^c, Prune^{\bar{c}})$
    if $(MDL((R \cup \{NewRule\})) - SmallestR) \geq \delta$ **then**
        return R
    else
        $R \leftarrow R \cup \{NewRule\}$
        $\mathcal{D}^c \leftarrow \mathcal{D}^c \backslash \{d^c \in \mathcal{D}^c | CoveredByRule(NewRule, d^c)\}$
        $\mathcal{D}^{\bar{c}} \leftarrow \mathcal{D}^{\bar{c}} \backslash \{d^{\bar{c}} \in \mathcal{D}^{\bar{c}} | CoveredByRule(NewRule, d^{\bar{c}})\}$
        if $MDL(R) < SmallestR$ **then**
            $SmallestR \leftarrow MDL(R)$
        end if
    end if
end while
for all $r \in R$ **do**
    randomly split $\mathcal{D}^c$ into $Grow^c$ and $Prune^c$
    randomly split $\mathcal{D}^{\bar{c}}$ into $Grow^{\bar{c}}$ and $Prune^{\bar{c}}$
    $Prune^{\bar{c}} \leftarrow Prune^{\bar{c}} \backslash \{d^{\bar{c}} \in Prune^{\bar{c}} | CoveredByRuleSet((R \backslash \{r\}), d^{\bar{c}})\}$
    $r' \leftarrow []$
    $r' \leftarrow GrowRuleMDL(r', Grow^c, Grow^{\bar{c}})$
    $r' \leftarrow PruneRuleMDL(r', Prune^c, Prune^{\bar{c}})$
    $r'' \leftarrow GrowRuleMDL(r, Grow^c, Grow^{\bar{c}})$
    $R' \leftarrow (R \backslash \{r\}) \cup r'$
    $R'' \leftarrow (R \backslash \{r\}) \cup r''$
    $R \leftarrow ShortestMDL(\{R, R', R''\})$
end for
return R

Algorithm A.3 k-Nearest Neighbor

Procedure k-NN($\mathcal{D}, \boldsymbol{y}$)
Input: $\mathcal{D}$: Dataset
$\boldsymbol{y}$: Data instance to be classified
Output: c: the predicted class

for all $i = 1$ to d **do**
  $v_{i,\max} \leftarrow$ largest value of feature V_i in $\mathcal{D}$
  $v_{i,\min} \leftarrow$ smallest value of feature V_i in $\mathcal{D}$
  for all $\boldsymbol{x} = \langle x_1, \ldots, x_i, \ldots, x_m, c \rangle \in \mathcal{D}$ **do**
    $x_i \leftarrow \dfrac{x_i - v_{i,\min}}{v_{i,\max} - v_{i,\min}}$
  end for
end for
for $j = 1$ to k **do**
  Initialize $\boldsymbol{z}_j$ such that all feature values in it is equal to 1 (i.e. the largest possible feature value).
end for
for all $\boldsymbol{x} \in \mathcal{D}$ **do**
  $\Delta \leftarrow \mathrm{Diff}_{\mathrm{Euc}}(\boldsymbol{x}, \boldsymbol{y})$
  if Δ is smaller than any element in $\{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_k\}$ **then**
    Replace the largest instance of $\{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_k\}$ with $\boldsymbol{x}$
  end if
end for
$c \leftarrow$ the majority class of all elements in $\{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_k\}$.
return c

Appendix B

Feature Functions

Some feature functions require additional functions to compute. We call those functions "auxiliary functions". Let $\mathcal{F}$ be a set of flows, $\mathcal{F}'$ a set of managed flows, $(p_i \mid 1 \leq i \leq n) \in \mathcal{F}$, and $f' \in \mathcal{F}'$. The auxiliary and features functions are defined as follows:

Auxiliary Functions:

$$
\begin{aligned}
\mathit{length}: \quad & \mathcal{F} \to \mathbb{N}^+ \\
& (p_i \mid 1 \leq i \leq n) \mapsto n \\
\mathit{iat}: \quad & P \times P \to \mathbb{N} \\
& (p, p') \mapsto |(\mathit{timestamp}(p') - \mathit{timestamp}(p))|
\end{aligned}
$$

Features Functions:

$$
\begin{aligned}
\mathit{protocol}: \quad & \mathcal{F} \to \mathbb{P} \\
& (p_i \mid 1 \leq i \leq n) \mapsto \mathit{protocol}(p_1) \\
\mathit{srcPort}: \quad & \mathcal{F} \to \mathbb{N} \\
& (p_i \mid 1 \leq i \leq n) \mapsto \mathit{srcPort}(p_1) \\
\mathit{dstPort}: \quad & \mathcal{F} \to \mathbb{N} \\
& (p_i \mid 1 \leq i \leq n) \mapsto \mathit{dstPort}(p_1)
\end{aligned}
$$

Figure B.1: Features functions and their auxiliary functions. The feature functions whose names end with CF are the functions that compute the abstraction of the coflow. Observe that the length of the coflow could exceed the length of the given flow.

$$connTime: \quad \mathcal{F} \to \mathbb{R}$$
$$(p_i \mid 1 \le i \le n) \mapsto \frac{(timestamp(p_n) - timestamp(p_1))}{1000}$$

$$connTimeCF: \quad \mathcal{F}' \to \mathbb{R}$$
$$f' \mapsto \begin{cases} connTime(f'') & \text{if } (\exists f'' \in \mathcal{F})(f', f'') \in Coflow, \\ 0 & \text{otherwise.} \end{cases}$$

$$dataVolume: \quad \mathcal{F} \to \mathbb{N}$$
$$(p_i \mid 1 \le i \le n) \mapsto \sum_{i=1}^{n} size(p_i)$$

$$dataVolumeCF: \quad \mathcal{F}' \to \mathbb{N}$$
$$f' \mapsto \begin{cases} dataVolume(f'') & \text{if } (\exists f'' \in \mathcal{F})(f', f'') \in Coflow, \\ 0 & \text{otherwise.} \end{cases}$$

$$dataVolumeRatio: \quad \mathcal{F} \to \mathbb{R}$$
$$f \mapsto \frac{dataVolume(f)}{dataVolumeCF(f)}$$

$$pktCount: \quad \mathcal{F} \to \mathbb{N}$$
$$(p_i \mid 1 \le i \le n) \mapsto n$$

$$pktCountCF: \quad \mathcal{F}' \to \mathbb{N}$$
$$f' \mapsto \begin{cases} pktCount(f'') & \text{if } (\exists f'' \in \mathcal{F})(f', f'') \in Coflow, \\ 0 & \text{otherwise.} \end{cases}$$

$$pktCountTotal \quad \mathcal{F}' \to \mathbb{N}$$
$$f' \mapsto pktCount(f') + pktCountCF(coflow(f'))$$

$$pktCountRatio \quad \mathcal{F}' \to \mathbb{N}$$
$$f' \mapsto \begin{cases} \dfrac{pktCount(f')}{pktCountCF(f'')} & \text{if } (\exists f'' \in \mathcal{F})(f', f'') \in Coflow, \\ 0 & \text{otherwise.} \end{cases}$$

$$pktSizeAvg: \quad \mathcal{F} \to \mathbb{R}$$
$$(p_i \mid 1 \le i \le n) \mapsto \frac{dataVolume((p_i \mid 1 \le i \le n))}{n}$$

$$pktSizeAvgCF: \quad \mathcal{F}' \to \mathbb{R}$$
$$f' \mapsto \begin{cases} pktSizeAvg(f'') & \text{if } (\exists f'' \in \mathcal{F})(f', f'') \in Coflow, \\ 0 & \text{otherwise.} \end{cases}$$

Figure B.1 (continued)

$$pktSizeDiff: \quad \mathcal{F} \to \mathbb{R}$$
$$(p_i \mid 1 \le i \le n) \mapsto \frac{\sum_{i=1}^{n-1} |size(p_i) - size(p_{i+1})|}{n}$$

$$pktSizeDiffCF: \quad \mathcal{F}' \to \mathbb{R}$$
$$f' \mapsto \begin{cases} pktSizeDiff(f'') & \text{if } (\exists f'' \in \mathcal{F})(f', f'') \in Coflow, \\ 0 & \text{otherwise.} \end{cases}$$

$$TPUTDiff_{\omega}: \quad \mathcal{F} \to \mathbb{R}$$
$$f \mapsto \frac{sumTPUTDiff(f, \omega)}{packetCount(f)}$$

$$TPUTDiffCF_{\omega}: \quad f' \mapsto \begin{cases} TPUTDiff_{\omega}(f'') & \text{if } (\exists f'' \in \mathcal{F})(f', f'') \in Coflow, \\ 0 & \text{otherwise.} \end{cases}$$

$$pktSizeSD: \quad \mathcal{F} \to \mathbb{R}$$
$$(p_i \mid 1 \le i \le n)$$
$$\mapsto \begin{cases} 0 & \text{if } n = 1, \\ \sqrt{\frac{\sum_{i=1}^{n}(size(p_i) - pktSizeAvg((p_i \mid 1 \le i \le n)))^2}{(n-1)}} & \text{otherwise.} \end{cases}$$

$$pktSizeSDCF: \quad \mathcal{F}' \to \mathbb{R}$$
$$f' \mapsto \begin{cases} pktSizeSD(f'') & \text{if } (\exists f'' \in \mathcal{F})(f', f'') \in Coflow, \\ 0 & \text{otherwise.} \end{cases}$$

$$pktSizeRMS: \quad \mathcal{F} \to \mathbb{R}$$
$$(p_i \mid 1 \le i \le n) \mapsto \sqrt{\frac{\sum_{i=1}^{n} size(p_i)^2}{n}}$$

$$pktSizeRMSCF: \quad \mathcal{F}' \to \mathbb{R}$$
$$f' \mapsto \begin{cases} pktSizeRMS(f'') & \text{if } (\exists f'' \in \mathcal{F})(f', f'') \in Coflow, \\ 0 & \text{otherwise.} \end{cases}$$

$$dataTPUTAvg: \quad \mathcal{F} \to \mathbb{R}$$
$$(p_i \mid 1 \le i \le n) \mapsto \frac{dataVolume((p_i \mid 1 \le i \le n))}{connTime((p_i \mid 1 \le i \le n))}$$

$$dataTPUTAvgCF: \quad \mathcal{F}' \to \mathbb{R}$$
$$f' \mapsto \begin{cases} dataTPUTAvg(f'') & \text{if } (\exists f'' \in \mathcal{F})(f', f'') \in Coflow, \\ 0 & \text{otherwise.} \end{cases}$$

Figure B.1 (continued)

$$pktTPUTAvg: \quad \mathcal{F} \to \mathbb{R}$$
$$(p_i \mid 1 \le i \le n) \mapsto \frac{n}{connTime((p_i \mid 1 \le i \le n))}$$

$$pktTPUTAvgCF: \quad \mathcal{F}' \to \mathbb{R}$$
$$f' \mapsto \begin{cases} pktTPUTAvg(f'') & \text{if } (\exists f'' \in \mathcal{F})(f', f'') \in Coflow, \\ 0 & \text{otherwise.} \end{cases}$$

$$iatAvg: \quad \mathcal{F} \to \mathbb{R}$$
$$(p_i \mid 1 \le i \le n) \mapsto \frac{\sum_{i=1}^{n-1} iat(p_i, p_{i+1})}{n-1}$$

$$iatAvgCF: \quad \mathcal{F}' \to \mathbb{R}$$
$$f' \mapsto \begin{cases} iatAvg(f'') & \text{if } (\exists f'' \in \mathcal{F})(f', f'') \in Coflow, \\ 0 & \text{otherwise.} \end{cases}$$

$$iatSD: \quad \mathcal{F} \to \mathbb{R}$$
$$(p_i \mid 1 \le i \le n)$$
$$\mapsto \begin{cases} 0 & \text{if } n < 3, \\ \sqrt{\frac{\sum_{i=1}^{n-1}(iat(p_i, p_{i+1}) - iatAvg((p_i \mid 1 \le i \le n)))^2}{(n-1)-1}} & \text{otherwise.} \end{cases}$$

$$iatSDCF: \quad \mathcal{F}' \to \mathbb{R}$$
$$f' \mapsto \begin{cases} iatSD(f'') & \text{if } (\exists f'' \in \mathcal{F})(f', f'') \in Coflow, \\ 0 & \text{otherwise.} \end{cases}$$

$$iatRMS: \quad \mathcal{F} \to \mathbb{R}$$
$$(p_i \mid 1 \le i \le n) \mapsto \begin{cases} iat(p_1, p_2) & \text{if } n < 3, \\ \sqrt{\frac{\sum_{i=1}^{n-1}(iat(p_i, p_{i+1}))^2}{(n-1)}} & \text{otherwise.} \end{cases}$$

$$iatRMSCF: \quad \mathcal{F}' \to \mathbb{R}$$
$$f' \begin{cases} iatRMS(f'') & \text{if } (\exists f'' \in \mathcal{F})(f', f'') \in Coflow, \\ 0 & \text{otherwise.} \end{cases}$$

$$iatVar: \quad \mathcal{F} \to \mathbb{R}$$
$$f \mapsto \frac{iatSD(f)}{iatAvg(f)}$$

$$iatVarCF: \quad \mathcal{F}' \to \mathbb{R}$$
$$f' \mapsto \begin{cases} iatVar(f'') & \text{if } (\exists f'' \in \mathcal{F})(f', f'') \in Coflow, \\ 0 & \text{otherwise.} \end{cases}$$

Figure B.1 (continued)

Appendix C

Signature-Based Flow Classification System

C.1 Payload Signatures

In our experiments, FlowStatTrace, which is a signature-based FCS, is used to find the true classes of instances (i.e., the ground-truth) in the datasets. To ensure the reliability of the data, countless hours of efforts were spent to extract and verify the signatures of various application protocols used in the evaluations. Most of the signatures are obtained through a deep-packet inspection approach, which involves capturing packets of the target protocols and scrutinizing each of them to find common signatures. Apart from our own analysis, the signatures are derived from other the signature-based classification and protocol structure studies, including [Pro08], [RG07], [Pha05], [ZP00], [EMA+07], [Pol06], and [Yah08].

In total, 54 signatures of 24 application protocols are obtained and presented below. Each signature is located in the first four bytes of the packet. Some signatures are printable[1] protocol syntaxes or commands while some signatures are not. For the signatures that are printable, they will be represented in two parts as follows.

```
61 62 63 64 / abcd
```

The first part, `61 62 63 64`, is the hexadecimal representation of the signature. The second part, `abcd`, is the protocol signature in represented in ASCII characters. If there exists a byte in the signature that is not printable, it will be shown as '`.`'. If the signature is shorter than four bytes or some bytes can be arbitrary values, we use `??` to represent the arbitrary value. For instance:

```
61 ?? ?? 64 / a??d
```

[1] "Printable" means that the characters of a signature can be shown using ASCII code.

C.2 Strict Conversational Class

Real-time Transport Protocol (RTP)

It is a standardized protocol for delivering video and audio data [SCFJ03]. The RTP flows typically run on even UDP ports. Its signature is:

```
80 ?? ?? ?? / .???
```

The second byte in the signature indicates the media that the RTP packet is carried. The last two bytes are the packet sequence number. Since the signature is too general, we also check if the flow is run on even UDP port.

Windows Live Messenger Video

The Windows Live Messenger video protocol is a propriety protocol used in Microsoft Windows Live Messenger software [Mic07] to perform video or audio conferences. According to the protocol structure documented in [Pol06] and our own analysis, the following signatures can be used to identify the protocol's videoconference packets.

```
4A 00 14 01 / ....
62 80 ?? ?? / ....
44 60 00 00 / ....
```

Real-Time Online Games

Not only videoconferences or audio calls, recent real-time online games are also sensitive to delay and jitter as well. Therefore, they are also categorized as Strict Conversational. Half-Life, a famous online first-person shooter game, is chosen for evaluation. This is because the game is extremely popular and all versions of it as well as the other third-party games that uses its engine share the same signature. The games that employ Half-Life game engine include Counter-Strike, Quake, and Day Of Defeat. The following is the protocol signature:

```
FF FF FF FF / ....
```

C.3 Relaxed Conversational Class

Secure Shell (SSH)

Secure Shell protocol is a network protocol that enables encrypted data transmission between two hosts. It is generally used for remote access in Unix or Linux systems.

```
53 53 48 2D / SSH-
```

Instant Messengers

Instant messengers provide user-to-user private chat sessions. They are one of the most popular services on the Internet. Our signature-based FCS recognizes a number of such services, including, Microsoft Windows Live Messenger, Yahoo Messenger, and AOL Instant Messenger.

Microsoft Windows Live Messenger

```
56 45 52 20 / VER.
41 4E 53 20 / ANS.
55 53 52 20 / USR.
80 78 68 2D / .xh-
49 4D 45 2D / IME-
50 4E 47 0D / PNG.
57 4E 47 20 / WNG.
```

Yahoo Messenger

```
59 4D 53 47 / YMSG
59 50 4E 53 / YPNS
59 48 4F 4F / YHOO
```

AOL Instant Messenger

```
2A ?? ?? ?? / *...
```

Internet Relay Chat (IRC)

Internet Relay Chat or IRC, like instant messengers, provides text-based chat sessions through a network. Unlike instant messenger, however, it is aimed for multi-user communication, where users have to connect to a designated server and join a discussion "channel" before the chat sessions can take place.

```
4E 49 43 4B / NICK
4D 4F 44 45 / MODE
```

Virtual Network Computing (VNC)

VNC is a remote-desktop sharing service, which allows a user from a remote terminal to share the graphical screen as well as inputs, such as keyboard and mouse, of another host. It is implemented using Remote FrameBuffer (RFB) protocol [Ric09], whose packet payloads begin with the following signature.

```
52 46 42 20 /  RFB.
```

C.4 Streaming Class

Streaming class consists of applications that provide live or on-demand digital media content such as live video or audio broadcasts. Examples of such applications include: Apple QuickTime [Qui08], Microsoft Windows Media Services [MMS08], and RealNetworks [Rea08]. Most streaming services are implemented over the Real Time Streaming Protocol (RTSP) [SRL98], which was standardized by the Internet Engineering Task Force (IETF) in 1998. It offers media streaming controls such as play and pause to the streaming clients. There are also other streaming services that do not use RTSP to transfer their streams. One of them is Nullsoft SHOUTcast [SHO08] protocol which we include in our experiment.

Real Time Streaming Protocol (RTSP)

We can distinguish RTSP from other RTSP flows by looking for the RTSP syntax. According to the [Pro08], the RTSP protocol usually begin with "RTSP" or "rtsp", as well as media-control commands. The RTSP packets can be easily captured by screening the packet payload for the following signatures:

```
52 54 53 50 / RTSP
72 74 73 70 / rtsp
50 4C 41 59 / PLAY
53 45 54 55 / SETU
4F 50 54 49 / OPTI
```

Nullsoft SHOUTcast

SHOUTcast protocol command always begin with "icy". Thus, streams can be captured with the following signature [Rad08, Pro08]:

```
69 63 79 2D / icy-
```

Media Streaming Through HTTP

In the past few years, streaming services using Adobe's Flash Video technology [Ado] has grown very popular. Example of these services are YouTube [You08], Google Video [Goo08], Metacafe [Met08], and many other websites. Such video streams are encapsulated by HTTP and sent through standard HTTP ports. To distinguish those services from other services that are also transferred through HTTP, we rely on the content-type [FB96a, FB96b] specified in the HTTP header.

```
43 6F 6E 74 65 6E 74 2D 54 79 70 65 3A 20 76 69 64 65 6F 2F /
Content-Type: video/
43 6F 6E 74 65 6E 74 2D 54 79 70 65 3A 20 61 75 64 69 6F 2F /
Content-Type: audio/
```

C.5 Interactive Class

Interactive services include, for instance, the world-wide-web, email receiving and sending, and news reading. Examples of the protocols corresponding to those services are webpages, Post Office Protocol (POP), Simple Mail Transfer Protocol (SMTP), Network News Transfer Protocol (NNTP).

Webpages

Currently, contents of a webpage do not include only texts and images anymore. Other contents such as video and audio streaming are also found embedded in the ordinary HTML pages. For Interactive class, only text and image contents are concerned. Therefore, only the flows with the following content-types are considered.

```
43 6F 6E 74 65 6E 74 2D 74 79 70 65 3A 20 74 65 78 74 2F /
Content-type: text/
43 6F 6E 74 65 6E 74 2D 74 79 70 65 3A 20 69 6D 61 67 65 2F /
Content-type: image/
```

Post Office Protocol (POP)

The signature for the Post Office Protocol (POP) are:

```
50 41 53 53 / PASS
4E 4F 4F 50 / NOOP
```

These signatures are some of the POP standardized commands. Since the commands are shared by other protocols such as FTP, we also have look into the transportation port as well. Apart of matching the signatures above, the packets will be identified as POP packet when it is sent through the TCP port 110, which is POP standard port.

Simple Mail Transfer Protocol (SMTP)

The Simple Mail Transfer Protocol (SMTP) is a text-based mail transfer protocol. Its standard port is TCP port 25 and its signatures are:

```
48 45 4C 4F / HELO
45 48 4C 4F / EHLO
32 32 30 2D / 220-
32 35 30 2D / 250-
```

As in the case of POP, the signatures are standardized commands, which can also be found in other protocols. Consequently, the transport port has to be checked as well.

Network News Transfer Protocol (NNTP)

The protocol is mainly used to access news articles on the Internet.

```
4D 4F 44 45 / MODE
47 52 4F 55 / GROU
3B 40 3F 2D / ;@?-
2A 58 57 2D / *XW-
42 60 58 2D / B‘X-
3F 47 36 2D / ?G6-
```

C.6 Background Class

Background class consists of services that are not sensitive to delay or delay variation. This includes data transferring services such as File Transfer Protocol (FTP), Server Message Block (SMB) protocol, Simple Mail Transfer Protocol (SMTP), and peer-to-peer file sharing services. Peer-to-peer file sharing services (or P2P) are services that allow users to download preferred files from more than one users at once. They are called peer-to-peer because they do not require centralized servers to manage the file distributions. More information on P2P service can be found in, e.g., [SG05] or [ATS04]. In our experiments we focus on four popular P2P services: Bittorrent, Gnutella, DirectConnect, and Manolito. The reason why we focus only on these four P2P services is because their protocols contain clear syntaxes can be clearly distinguished. Other services either encapsulate themselves in HTTP or do not have consistent signatures. Apart from the P2P protocols, FTP, SMB, and SMTP are also concerned.

Bittorrent

It is currently one of the most popular P2P protocols. The clients in the torrent networks initiate the connection between each other with the command that contains following signature.

```
13 42 69 74 / .Bit
```

Gnutella

Gnutella is a pure peer-to-peer protocol. The network hosts serve as both clients and servers simultaneously. The protocol could also be encapsulated in HTTP protocol. If not, the Gnutella packets usually contains the following signatures:

```
47 4E 55 54 / GNUT
A3 86 97 2D / ...-
47 49 56 20 / GIV
```

DirectConnect

It is a P2P framework that the clients always have to connect to a server to exchange files (although files are not provided by the server itself). The DirectConnect protocol messages always starts with $ follows by a protocol command. The commands that

we have seen in our data are: MyNick, Lock, and Sup. They can be captured using the following signatures:

```
24 4D 79 4E / $MyN
24 4C 6F 63 / $Loc
24 53 75 70 / $Sup
```

Manolito

It is another peer-to-peer protocol. It uses UDP ports 41170-41350 for network messages and a random TCP port between 10240 and 20480 to transfer a file. The signature of Manolito in one of its network messages is:

```
3D 4B D9 2D / =K?-
```

File Transfer Protocol (FTP)

FTP is a classical file transferring service. Its protocol is standardized so it can be easily detected. The signatures that we use to capture its packets are:

```
50 41 53 53 / PASS
4E 4F 4F 50 / NOOP
4D 4F 44 45 / MODE
32 31 31 2D / 211-
32 32 36 2D / 226-
```

Since some commands are also used by other protocols, we also use the transport port to assist the identification.

Server Message Block (SMB) Protocol

Server Message Block (SMB) Protocol is used by Microsoft Windows to provide various services such as file and printer sharing. The signature to capture SMB packet is:

```
53 4D 42 ?? / SMB?
```

Simple Mail Transfer Protocol (SMTP)

The protocol is generally used to transfer emails across the Internet. The SMTP signatures are as follows.

```
48 45 4C 4F / HELO
45 48 4C 4F / EHLO
32 32 30 2D / 210-
32 32 30 2D / 220-
32 35 30 2D / 250-
35 35 30 2D / 550-
```

Appendix D

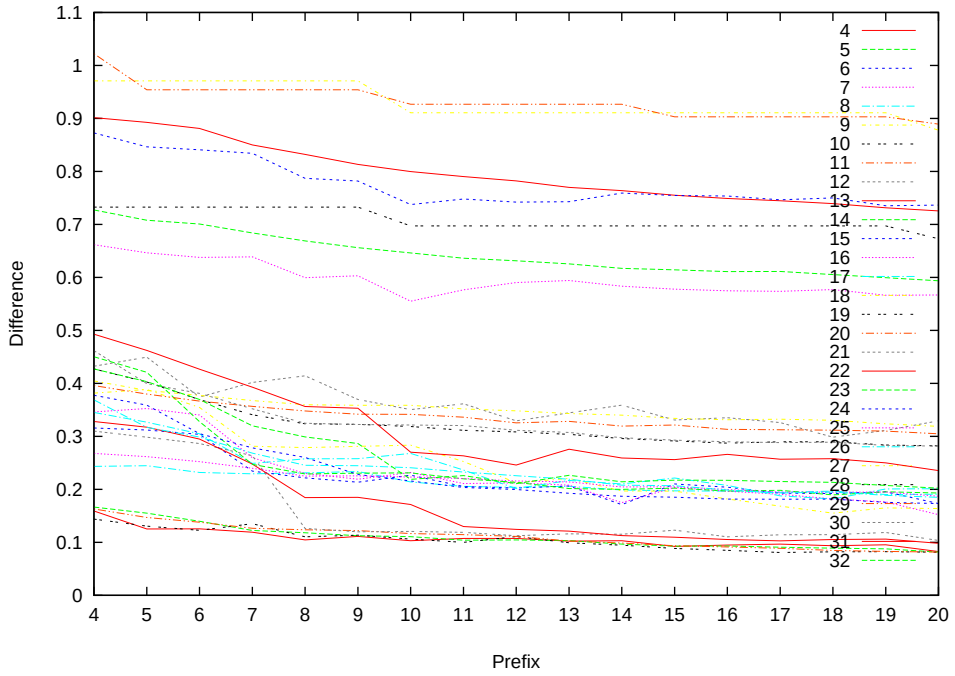
Flow Characteristics and Prefixes

Table D.1: Statistics of WIDE traces - Amount of packets and data of each trace.

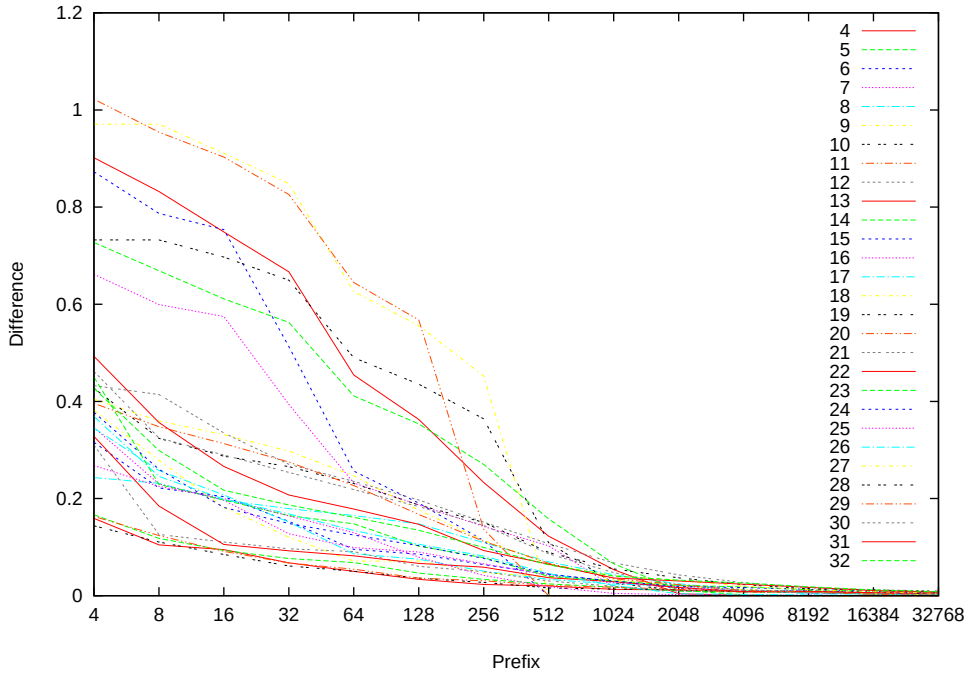
Traces	IPv4 Data (MB)	IPv4 Packets	IPv6 Data (MB)	IPv6 Packets
March 18				
00:00	60,476.20	119,292,166	43.89	335,820
08:00	54,980.23	91,867,380	28.70	182,531
12:00	70,357.34	118,436,674	60.31	263,392
16:00	87,682.00	137,723,441	75.33	282,507
20:00	65,354.08	113,953,372	93.16	363,258
March 19				
00:00	62,530.49	110,691,621	85.06	271,566
08:00	49,828.59	83,500,722	65.12	241,241
12:00	62,671.58	101,909,997	83.78	356,329
16:00	80,463.85	128,968,144	79.39	277,967
20:00	63,351.05	108,411,482	68.92	248,518
March 20				
00:00	58,527.78	98,624,605	131.89	1,366,254
08:00	47,579.19	77,043,197	48.23	195,156
12:00	55,501.56	96,138,934	64.38	226,803
16:00	64,861.18	116,615,339	84.04	227,406
20:00	50,142.51	100,840,514	38.01	200,281
Total	**934,307.62**	**1,604,017,588**	**1,050.21**	**5,039,029**

Table D.2: Statistics of WIDE traces - Number and percentage of flows identified in each trace.

Traces	Total	Identified	Unidentified	Identified (%)	Unidentified (%)
March 18					
00:00	10,646,926	3,184,849	7,462,077	29.91	70.09
08:00	8,633,413	2,117,542	6,515,871	24.53	75.47
12:00	9,310,384	2,710,965	6,599,419	29.12	70.88
16:00	9,633,095	2,953,538	6,679,557	30.66	69.34
20:00	10,067,802	2,542,424	7,525,378	25.25	74.75
March 19					
00:00	9,667,477	2,093,850	7,573,627	21.66	78.34
08:00	7,694,564	1,820,393	5,874,171	23.66	76.34
12:00	7,644,032	2,001,877	5,642,155	26.19	73.81
16:00	10,181,988	2,510,894	7,671,094	24.66	75.34
20:00	10,001,050	2,207,984	7,793,066	22.08	77.92
March 20					
00:00	9,291,889	2,087,068	7,204,821	22.46	77.54
08:00	7,055,083	1,544,780	5,510,303	21.90	78.10
12:00	8,017,126	2,238,274	5,778,852	27.92	72.08
16:00	10,551,872	2,545,686	8,006,186	24.13	75.87
20:00	10,501,660	2,482,109	8,019,551	23.64	76.36
Total	**138,898,361**	**35,042,233**	**103,856,128**	**25.23**	**74.77**

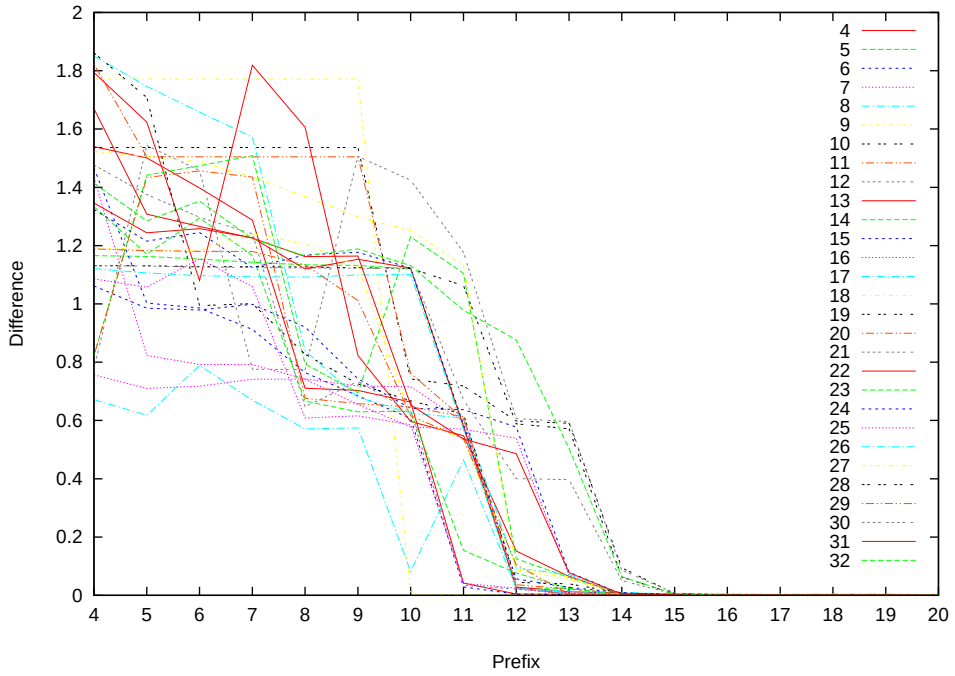


(a)

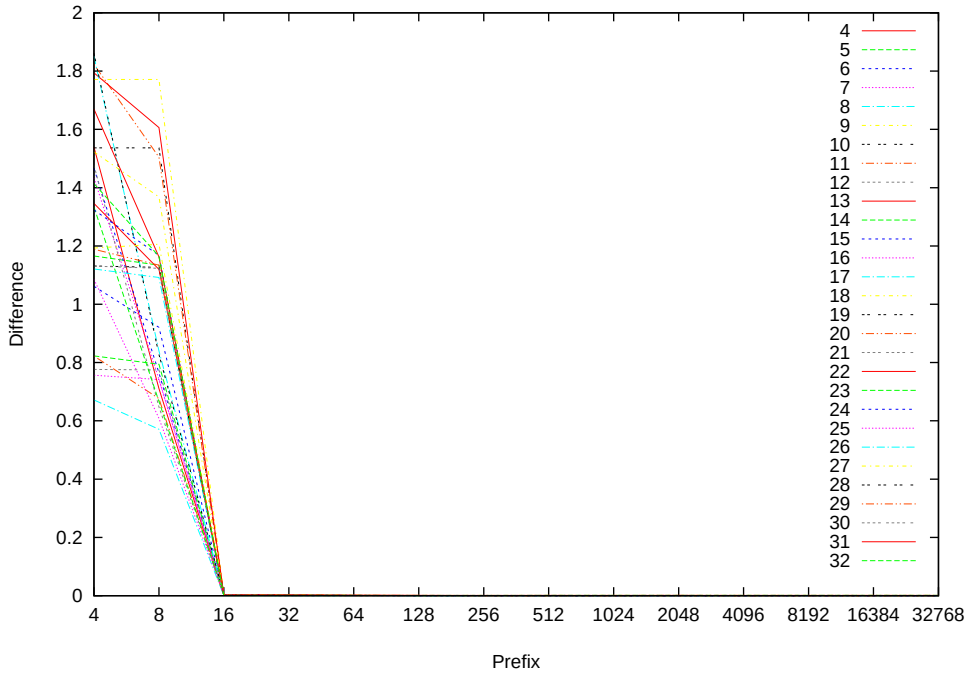


(b)

Figure D.1: Difference/Prefix plots of all features - StrConv. The number of each plot indicates which feature is associated to the plot. It corresponds to the numbers of the features listed in Table 5.1.

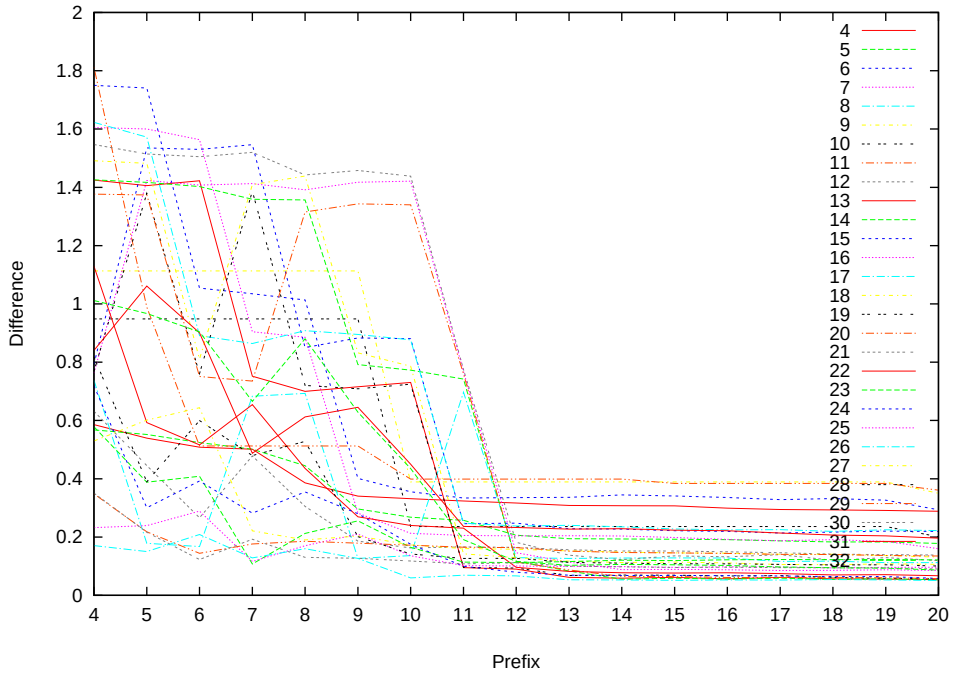


(a)

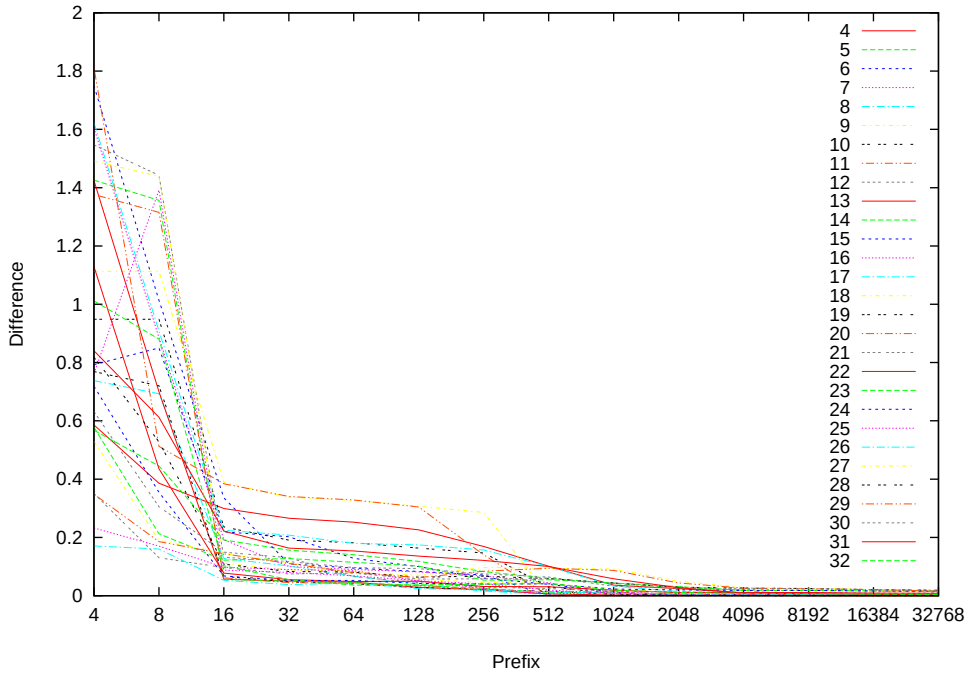


(b)

Figure D.2: Difference/Prefix plots of all features - RlxConv. The number of each plot indicates which feature is associated to the plot. It corresponds to the numbers of the features listed in Table 5.1.

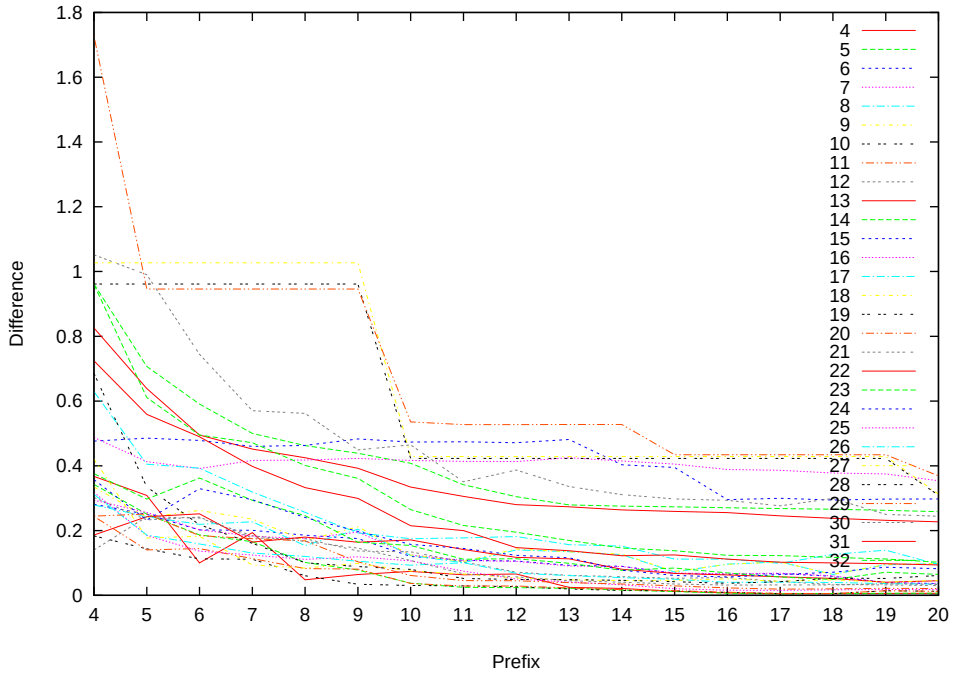


(a)

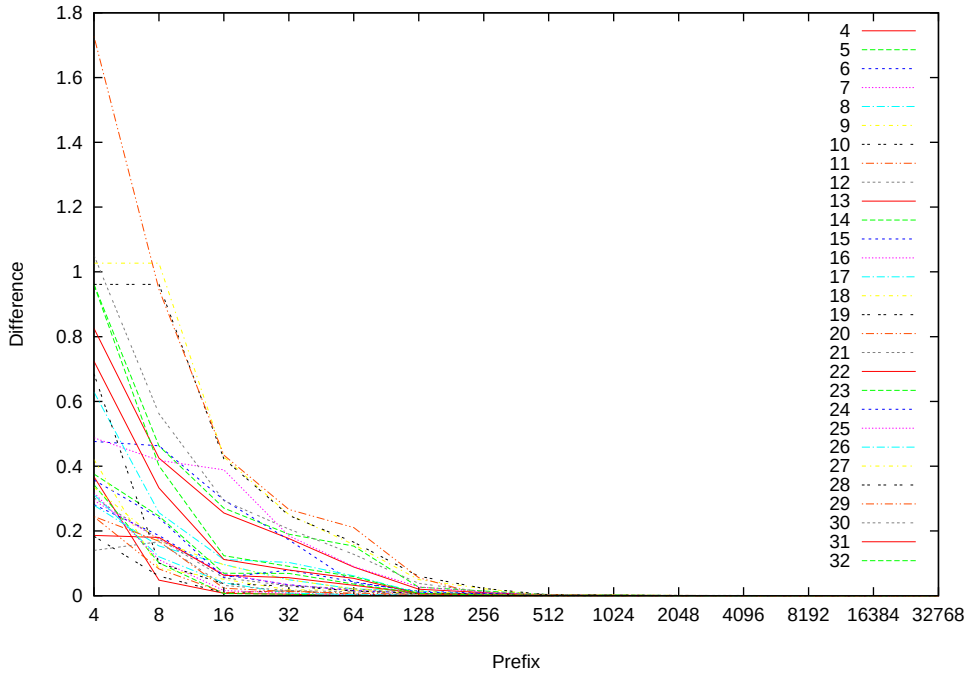


(b)

Figure D.3: Difference/Prefix plots of all features - Streaming. The number of each plot indicates which feature is associated to the plot. It corresponds to the numbers of the features listed in Table 5.1.

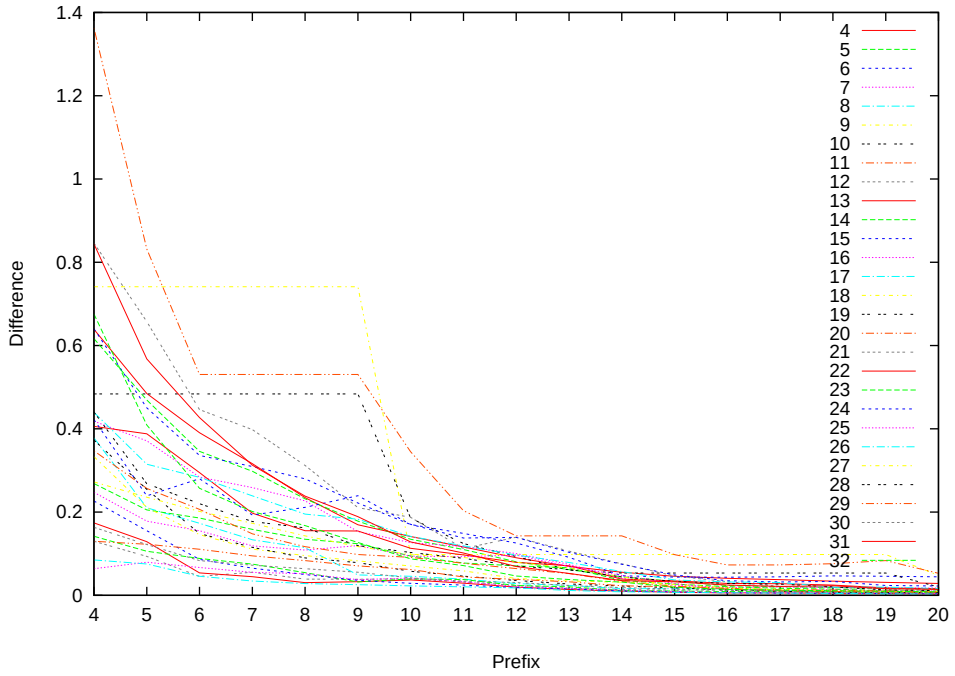


(a)

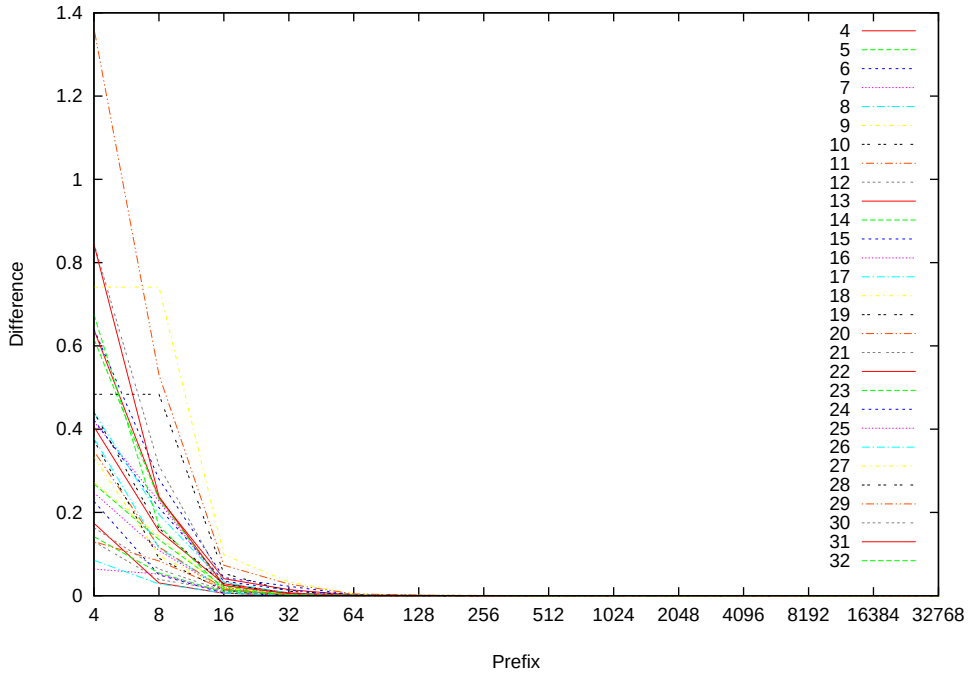


(b)

Figure D.4: Difference/Prefix plots of all features - Interactive. The number of each plot indicates which feature is associated to the plot. It corresponds to the numbers of the features listed in Table 5.1.

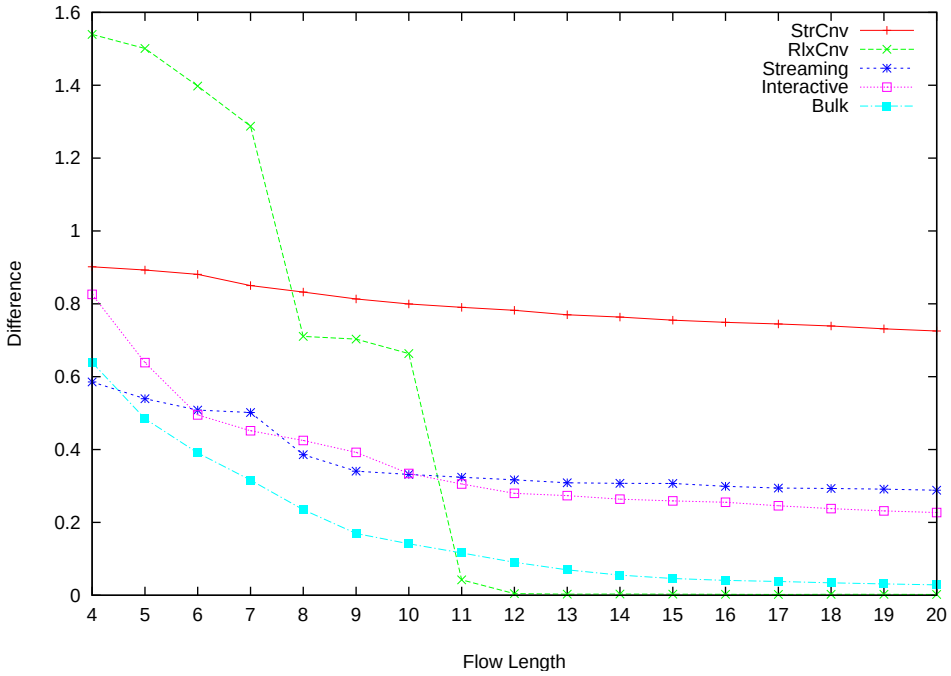


(a)

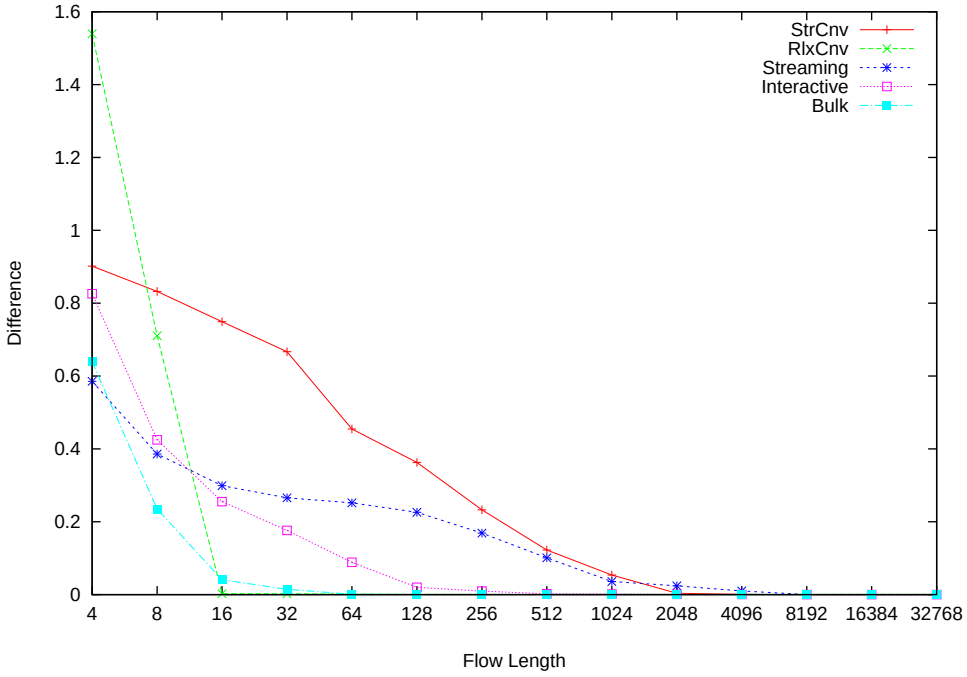


(b)

Figure D.5: Difference/Prefix plots of all features - Bulk. The number of each plot indicates which feature is associated to the plot. It corresponds to the numbers of the features listed in Table 5.1.

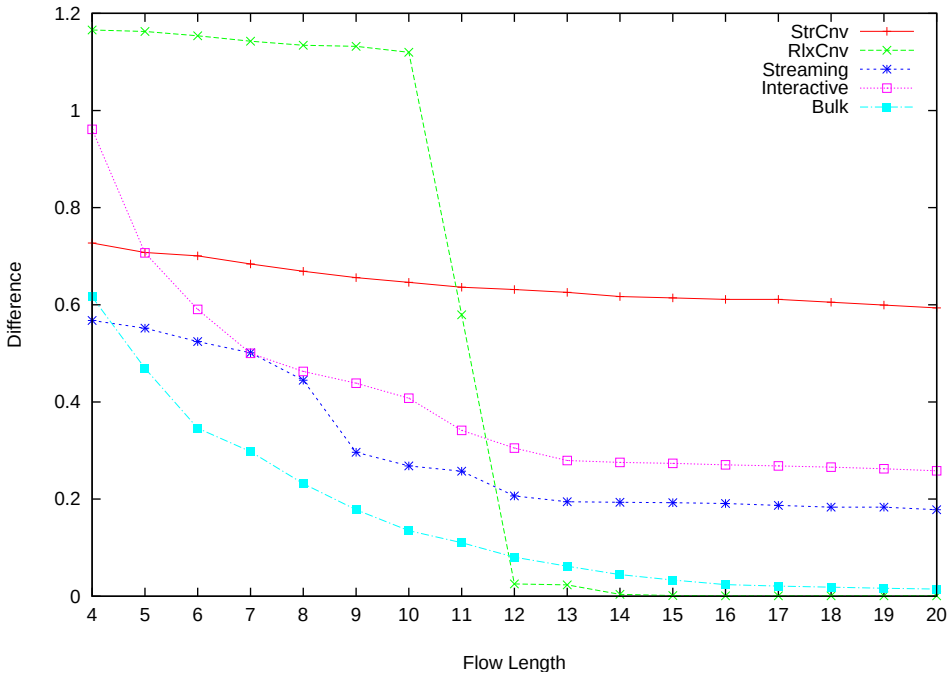


(a) Linear Scale

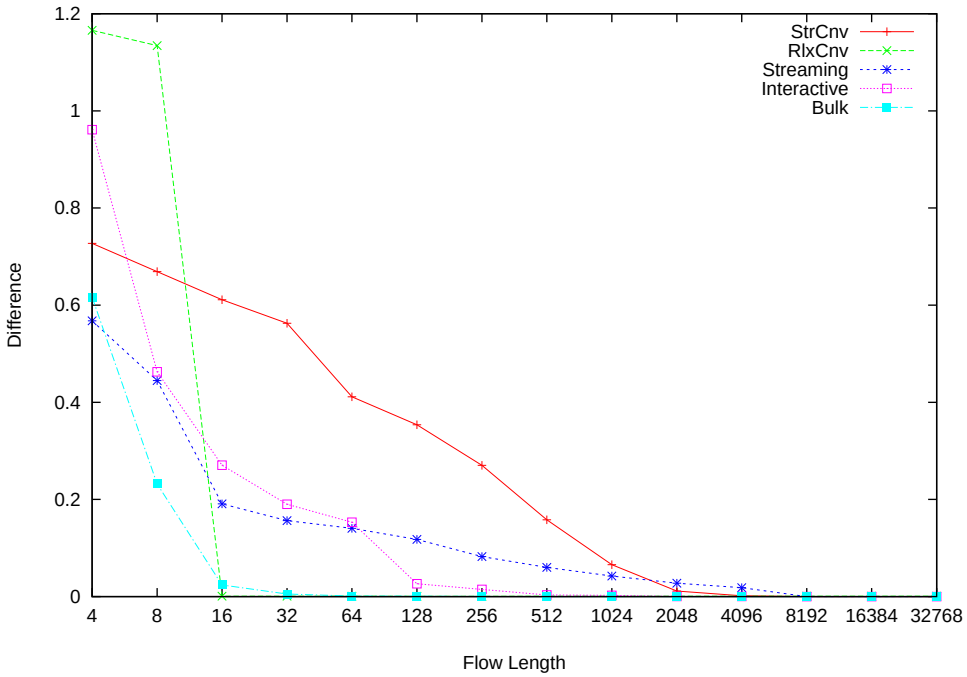


(b) Exponential Scale

Figure D.6: Difference/Prefix plots - connTime

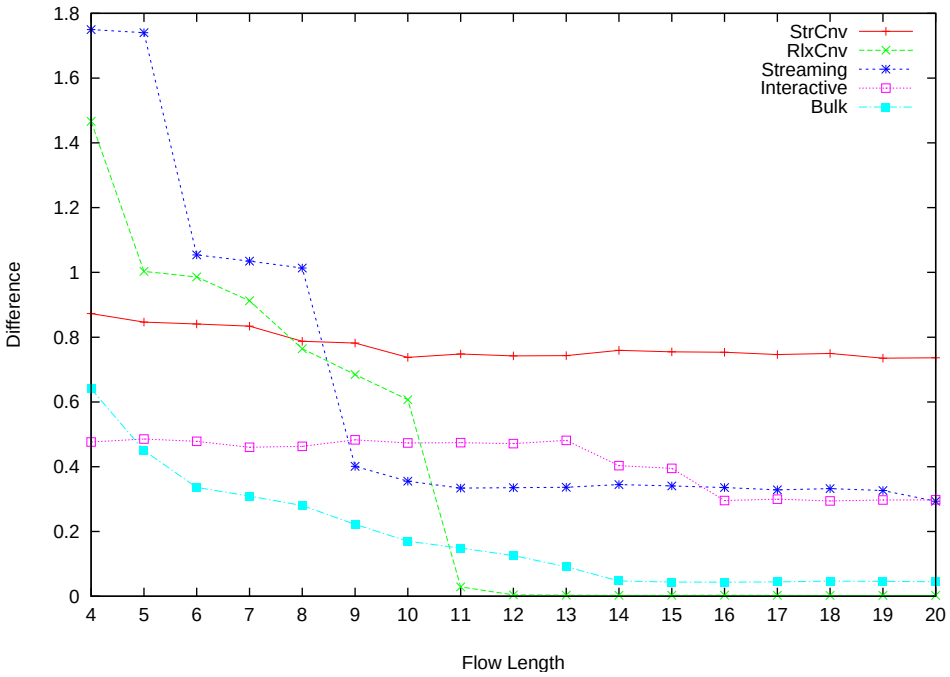


(a) Linear Scale

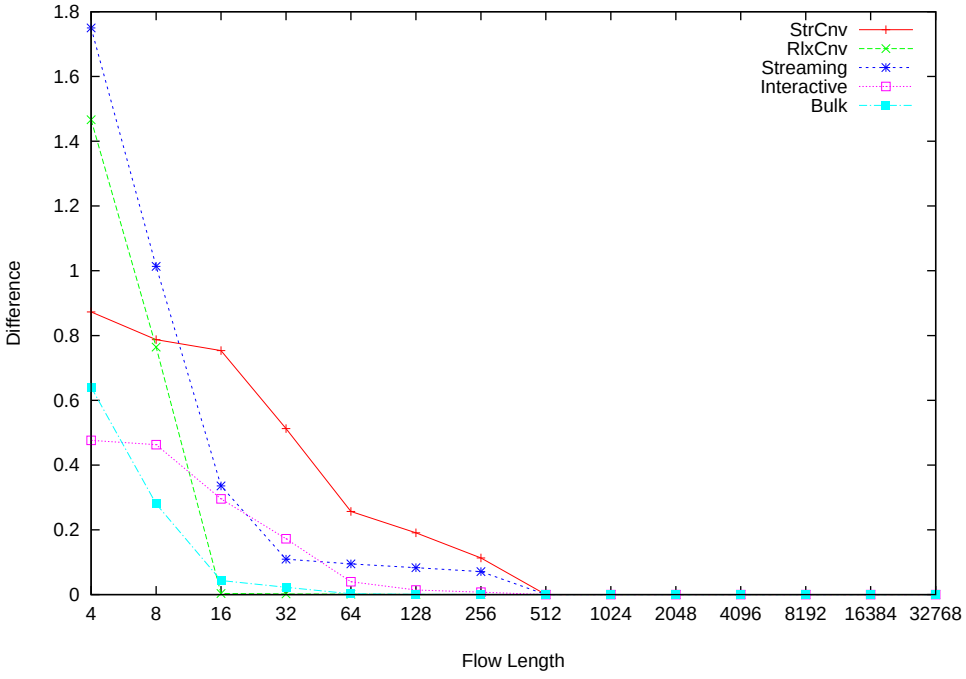


(b) Exponential Scale

Figure D.7: Difference/Prefix plots - connTimeCF

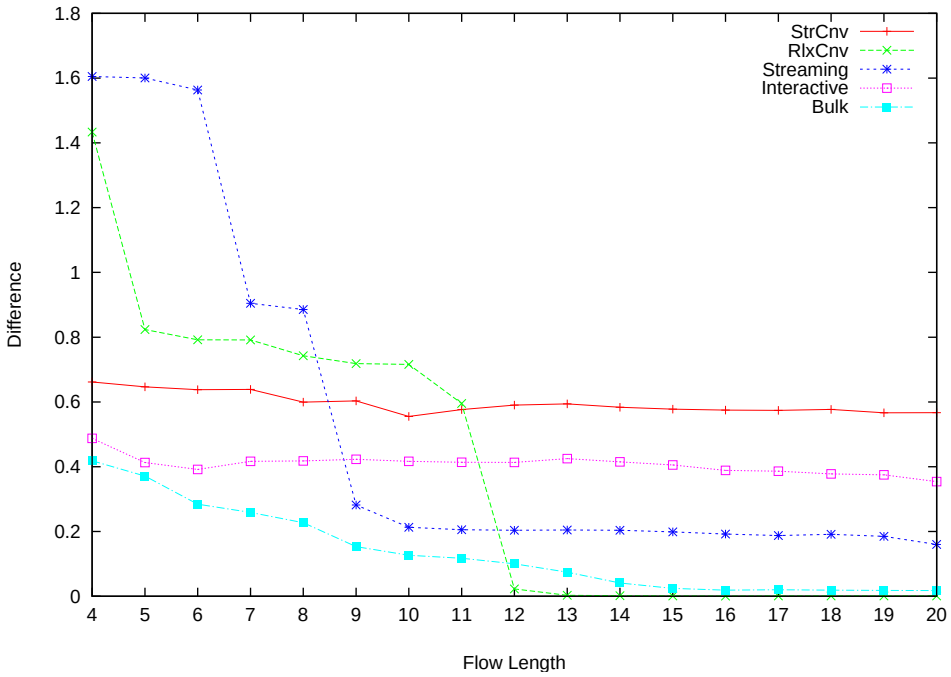


(a) Linear Scale

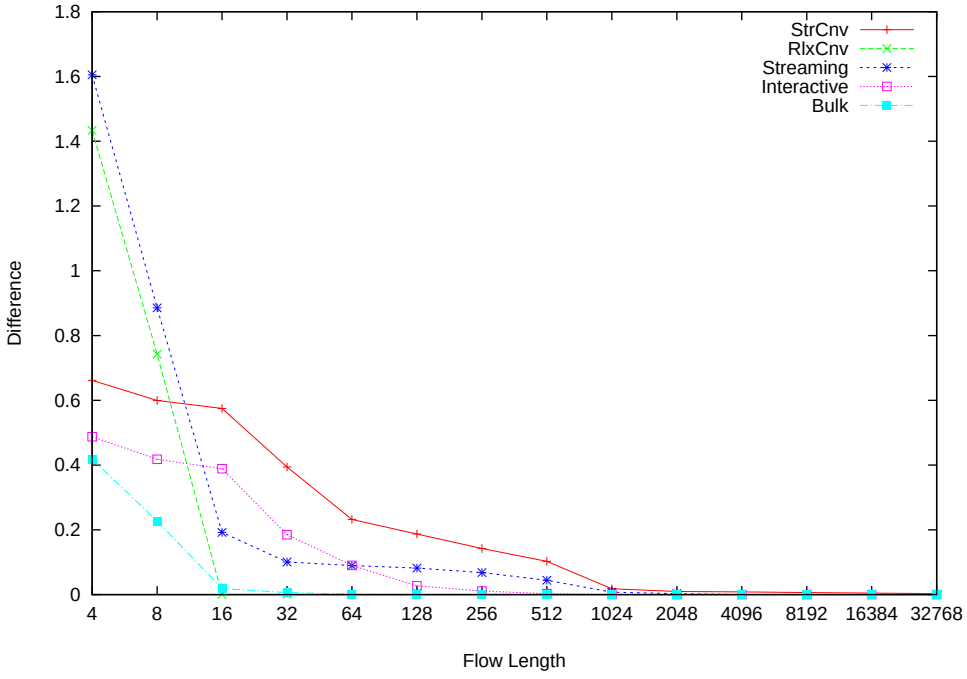


(b) Exponential Scale

Figure D.8: Difference/Prefix plots - dataVolume

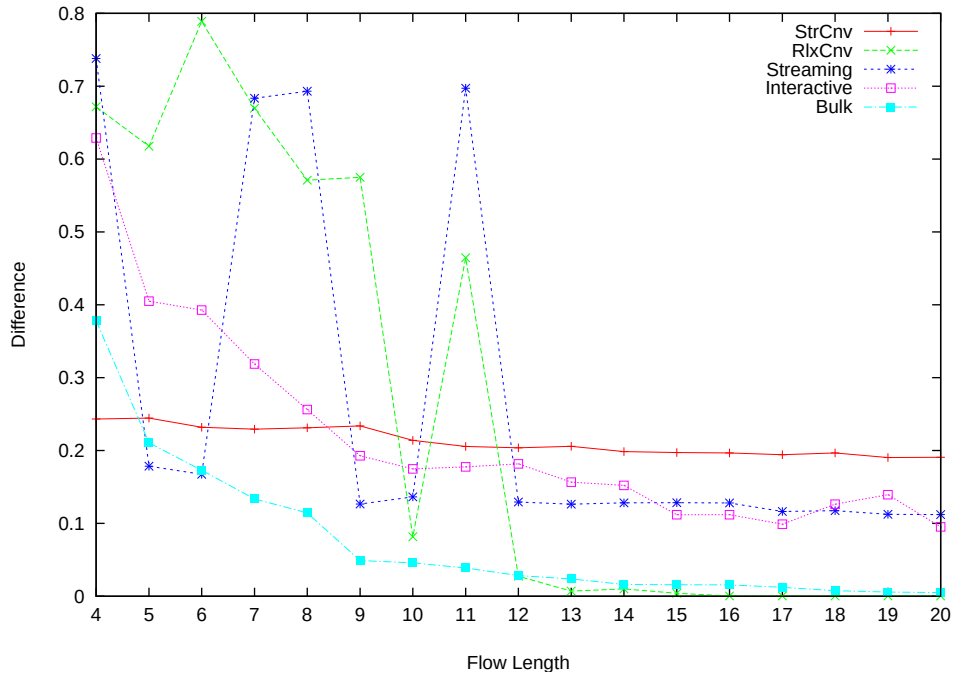


(a) Linear Scale

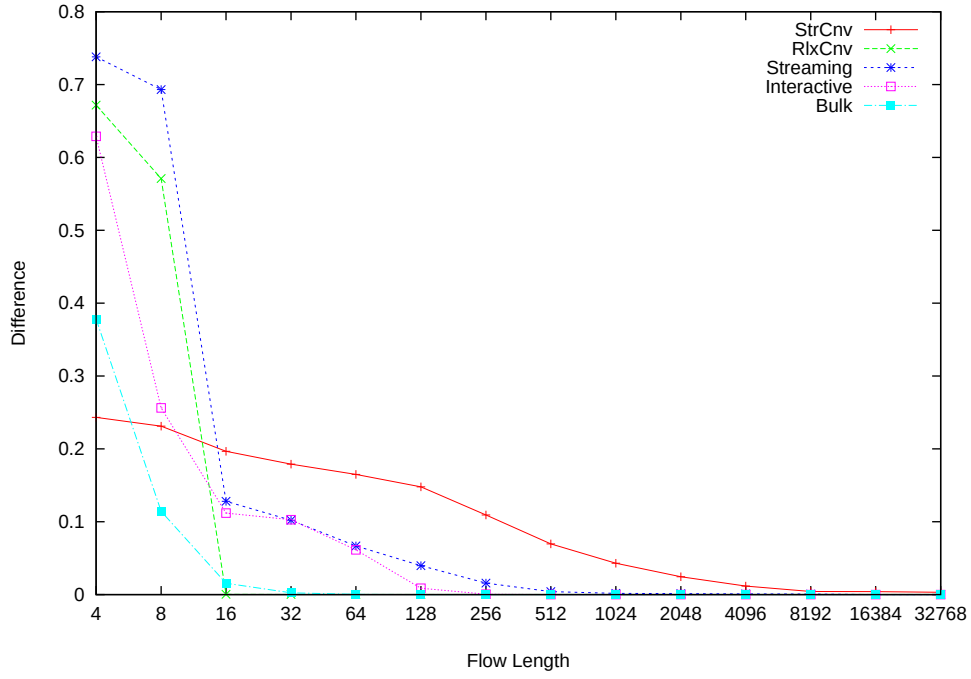


(b) Exponential Scale

Figure D.9: Difference/Prefix plots - dataVolumeCF

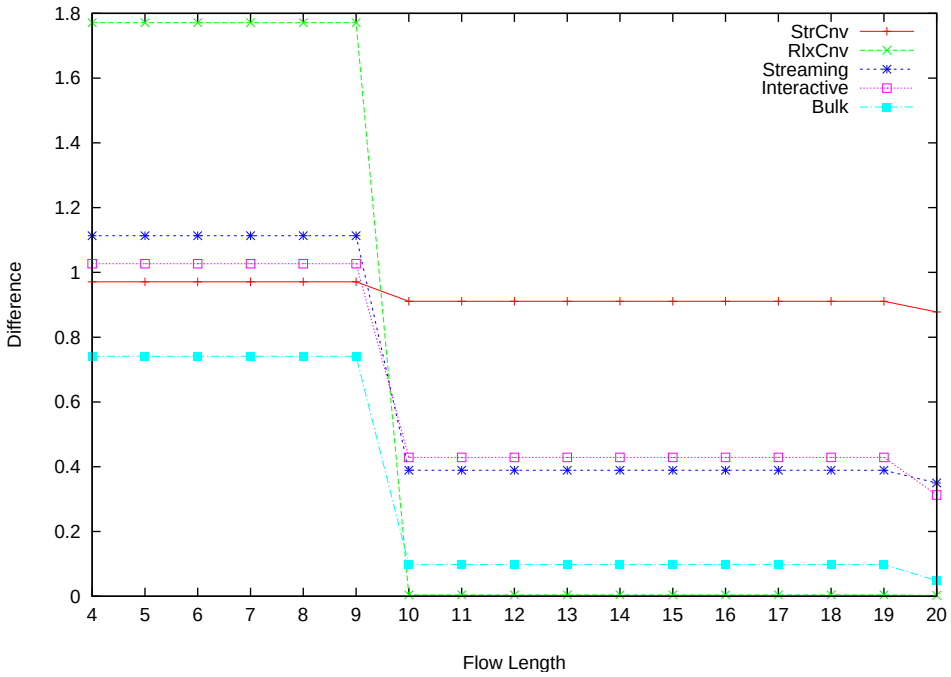


(a) Linear Scale

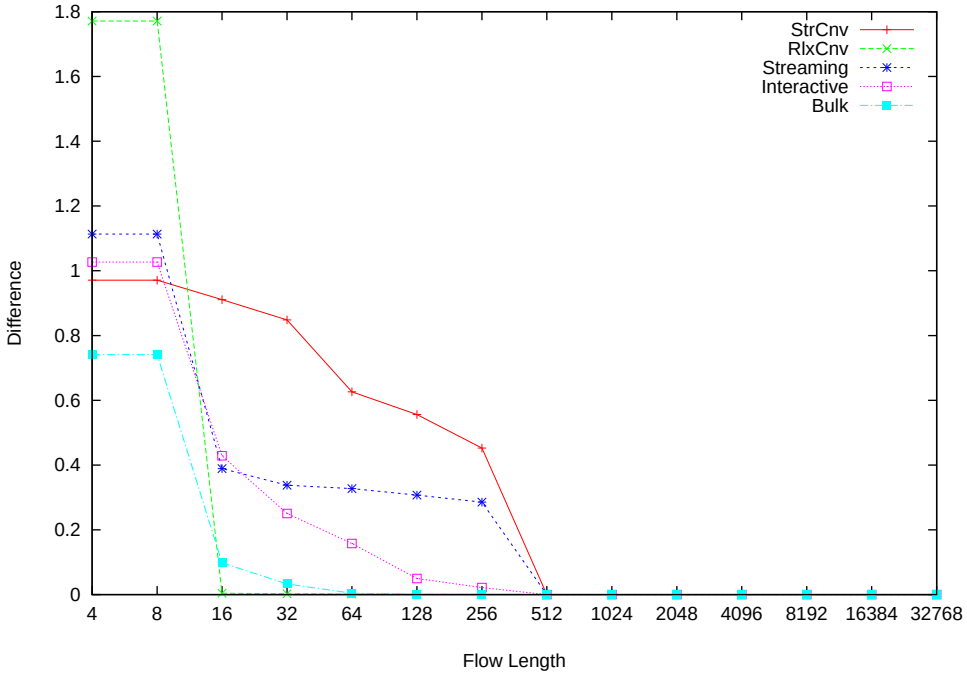


(b) Exponential Scale

Figure D.10: Difference/Prefix plots - dataVolumeRatio

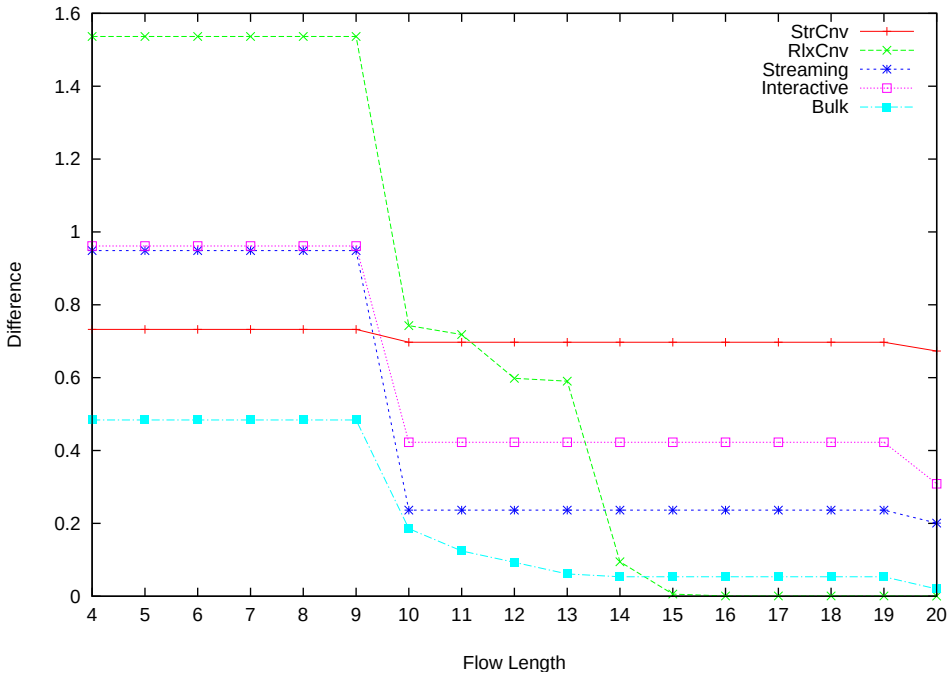


(a) Linear Scale

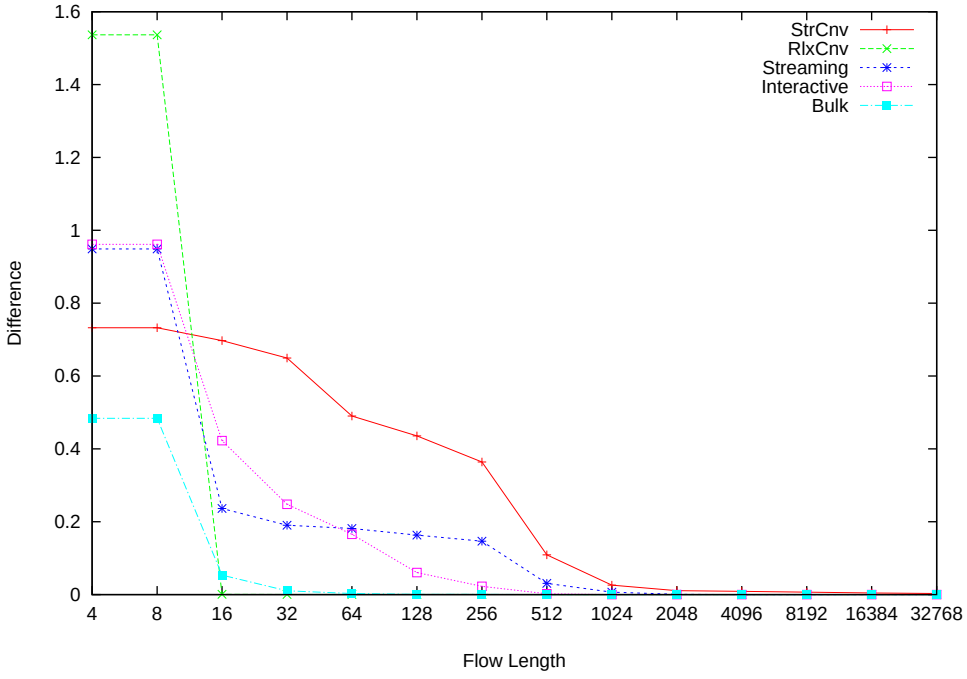


(b) Exponential Scale

Figure D.11: Difference/Prefix plots - pktCount

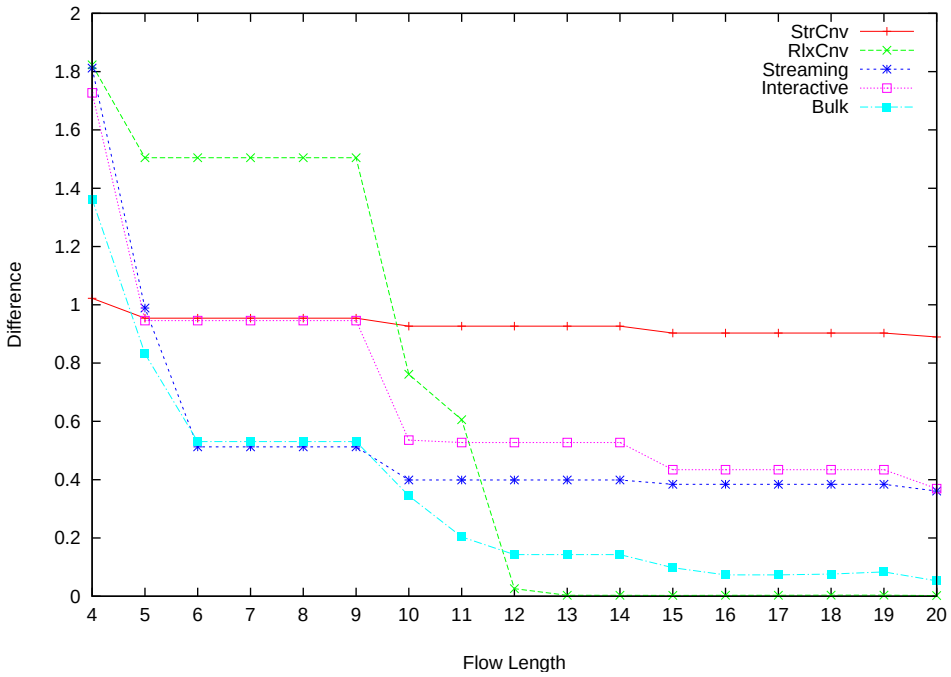


(a) Linear Scale

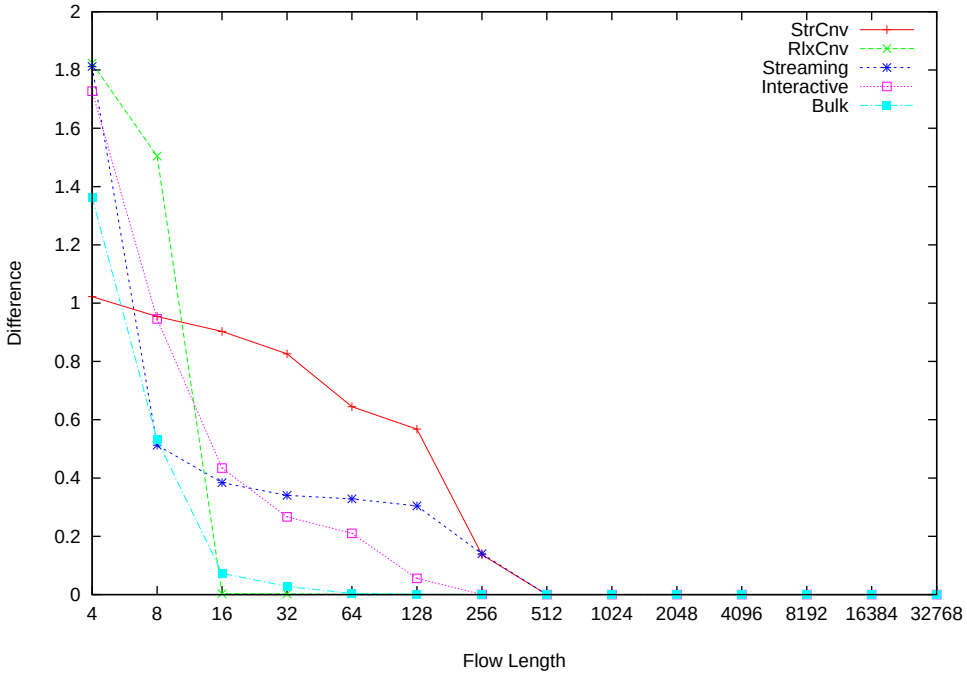


(b) Exponential Scale

Figure D.12: Difference/Prefix plots - pktCountCF

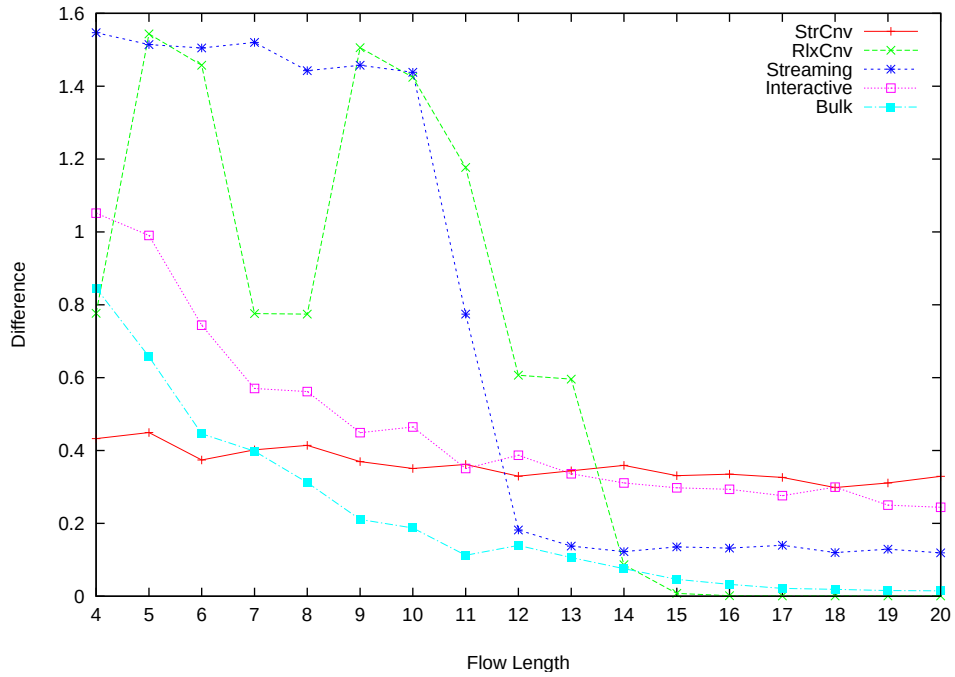


(a) Linear Scale

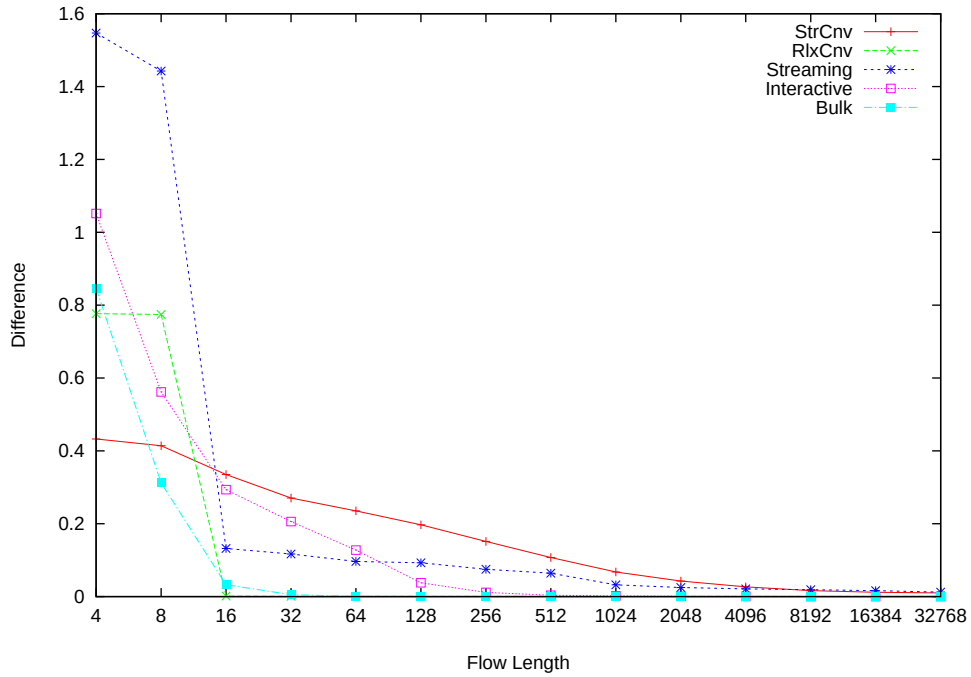


(b) Exponential Scale

Figure D.13: Difference/Prefix plots - pktCountTotal

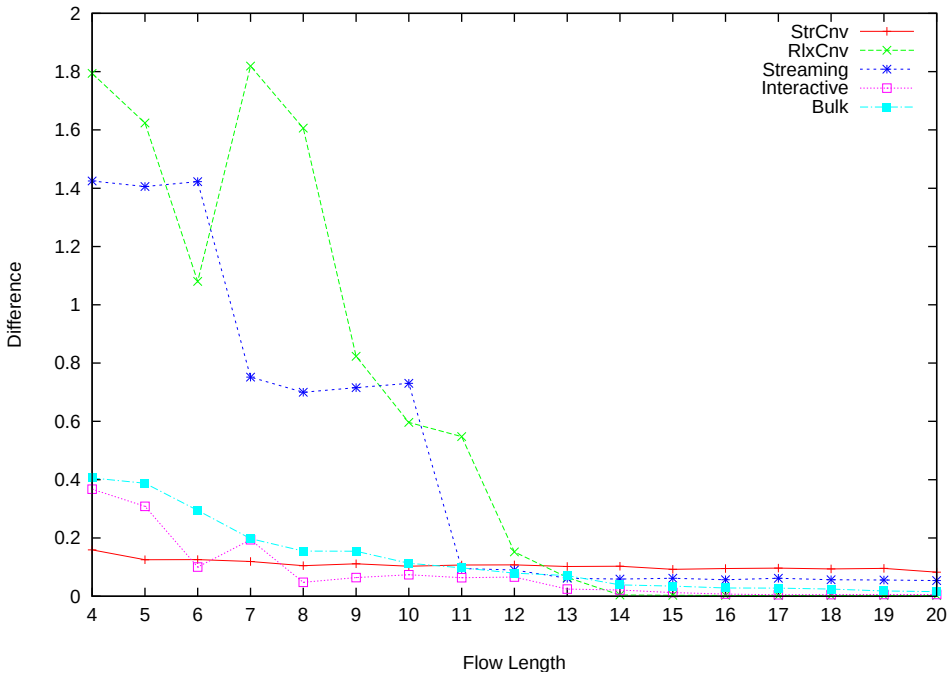


(a) Linear Scale

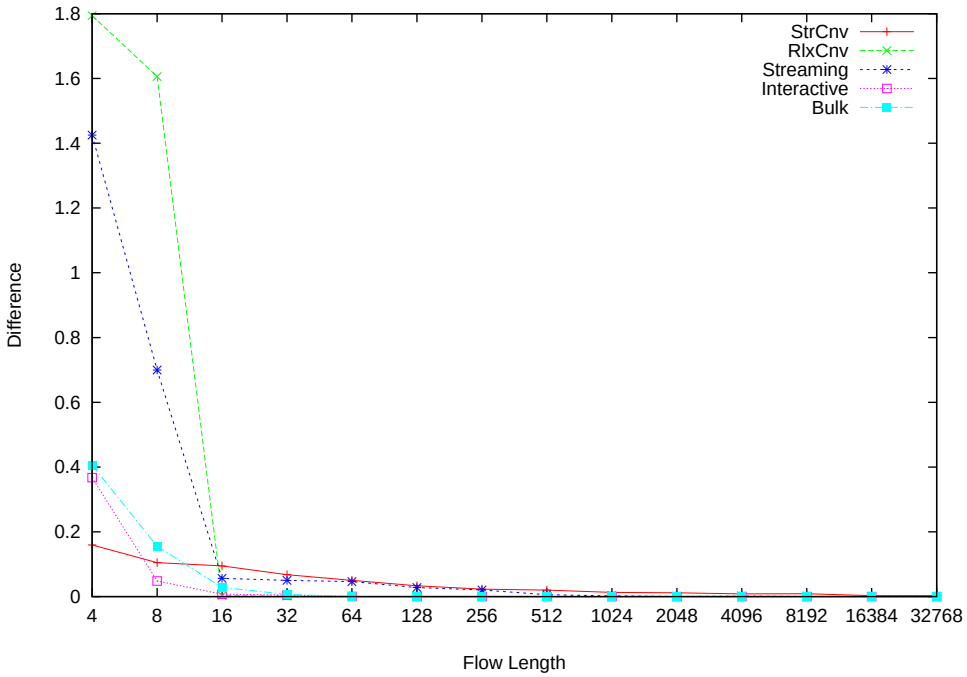


(b) Exponential Scale

Figure D.14: Difference/Prefix plots - pktCountRatio

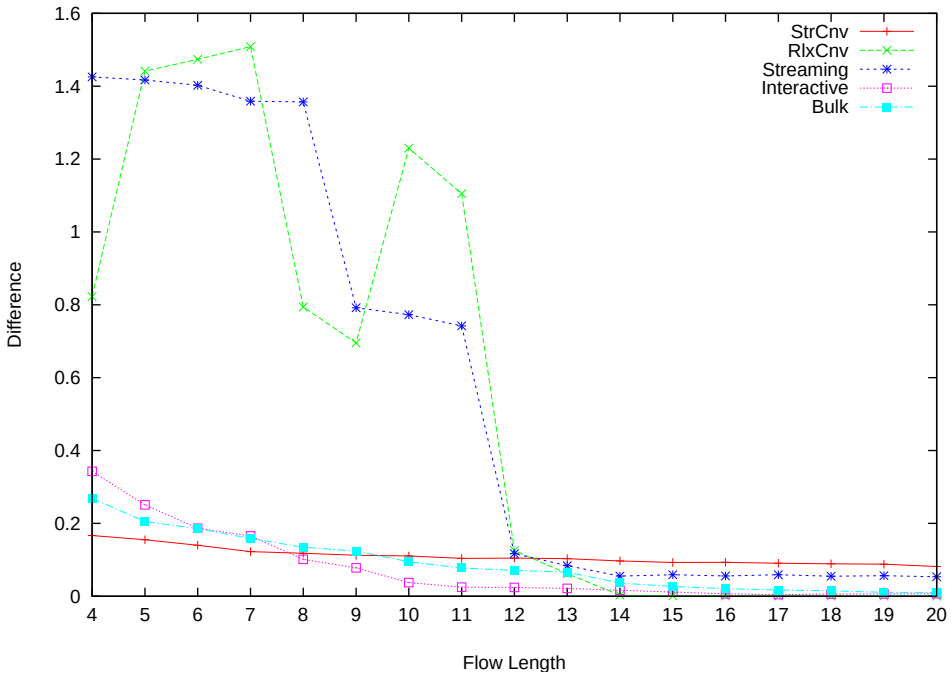


(a) Linear Scale

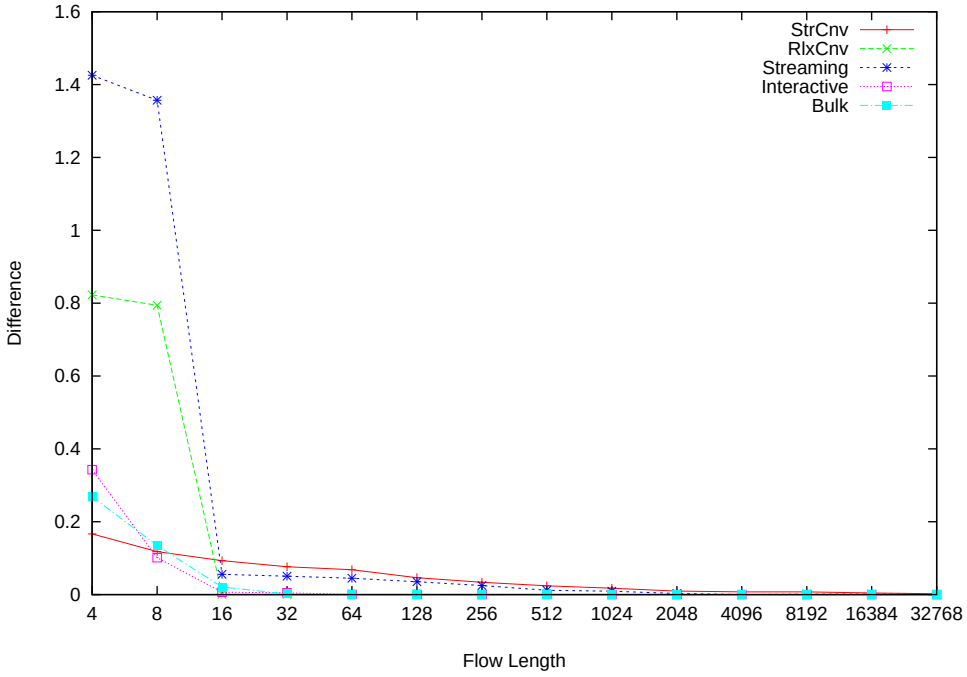


(b) Exponential Scale

Figure D.15: Difference/Prefix plots - pktSizeAvg

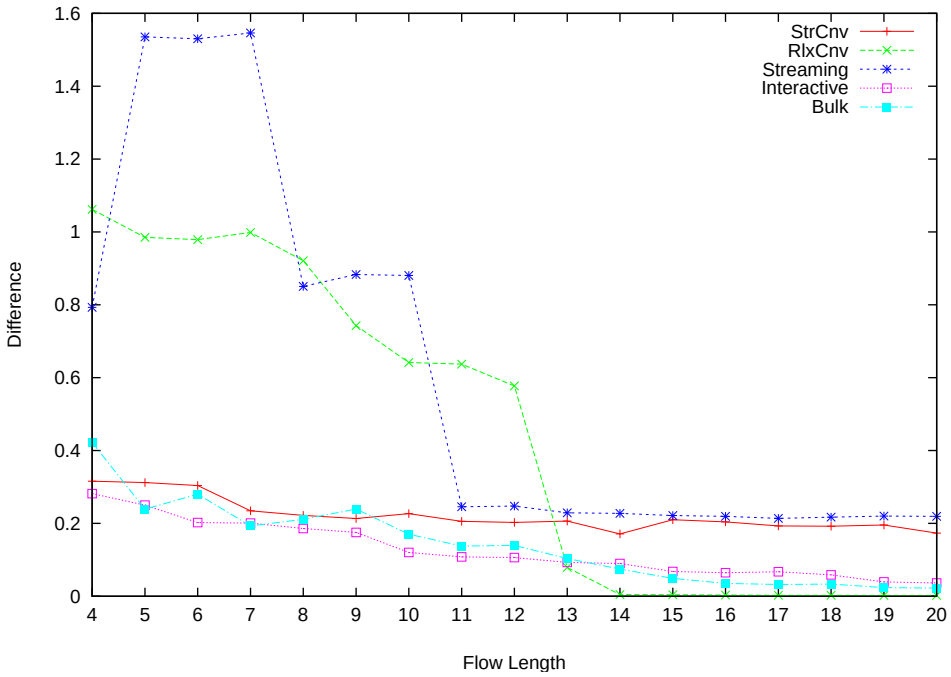


(a) Linear Scale

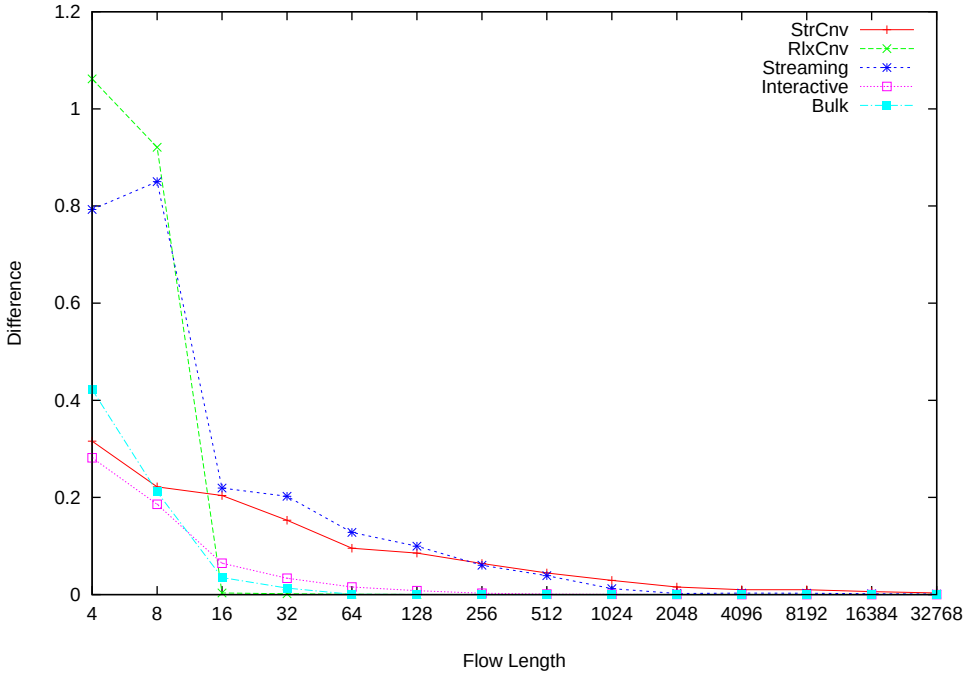


(b) Exponential Scale

Figure D.16: Difference/Prefix plots - pktSizeAvgCF

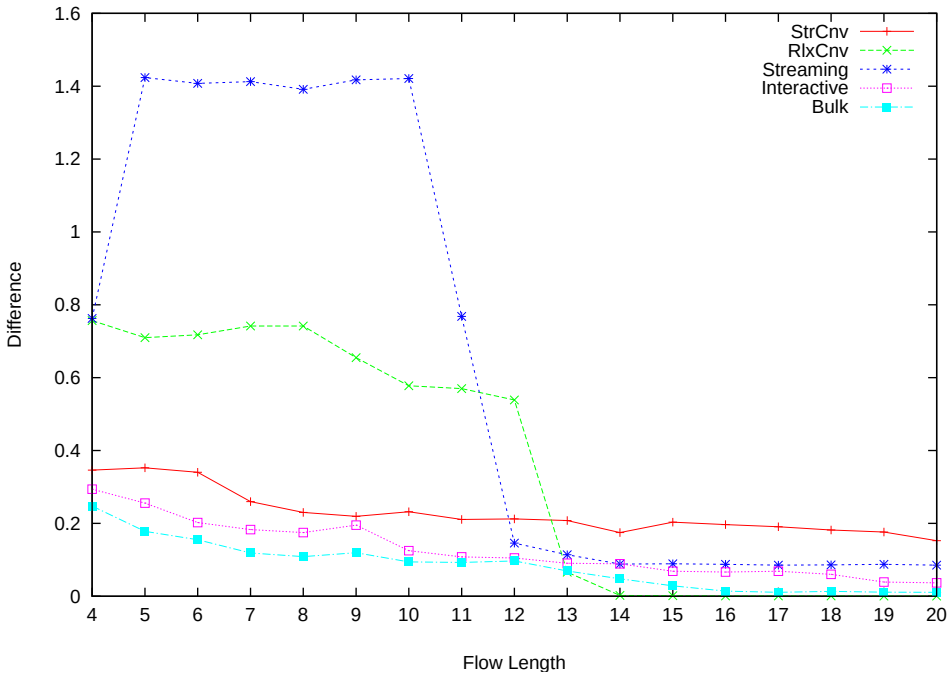


(a) Linear Scale

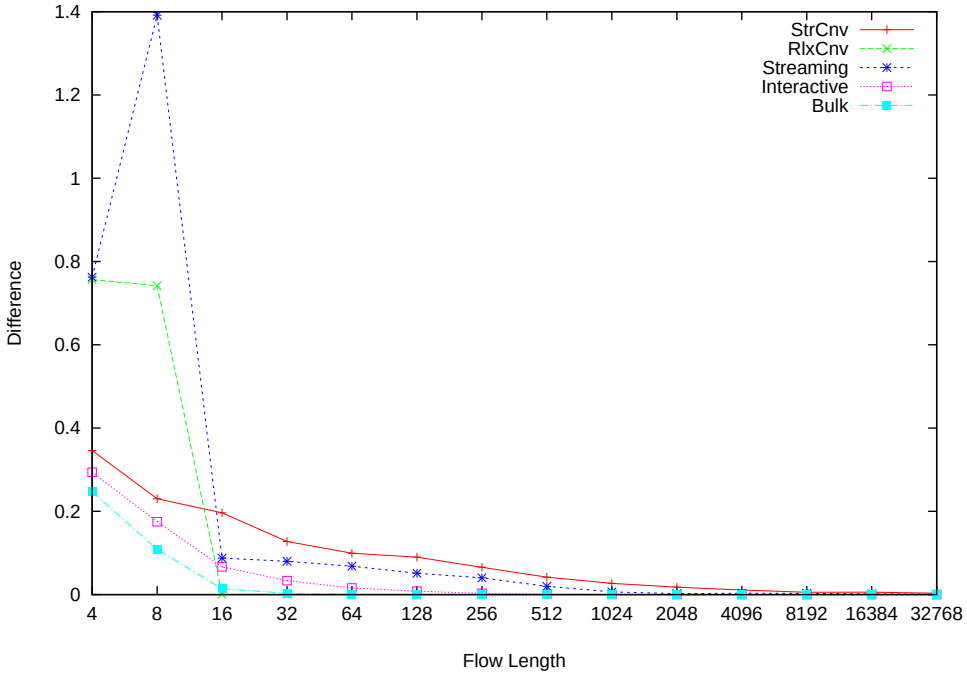


(b) Exponential Scale

Figure D.17: Difference/Prefix plots - pktSizeDiff

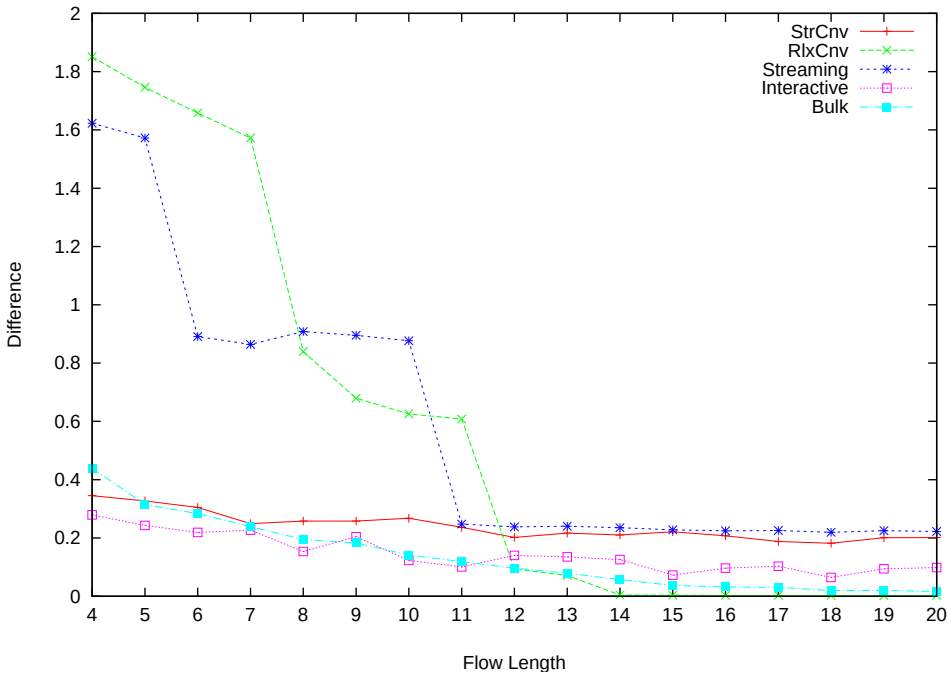


(a) Linear Scale

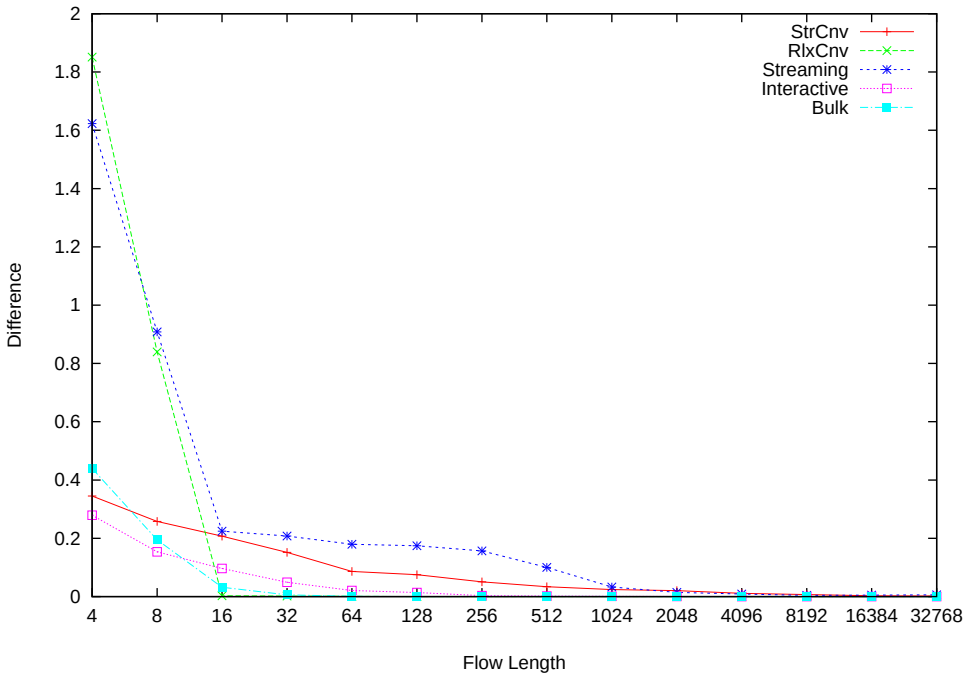


(b) Exponential Scale

Figure D.18: Difference/Prefix plots - pktSizeDiffCF

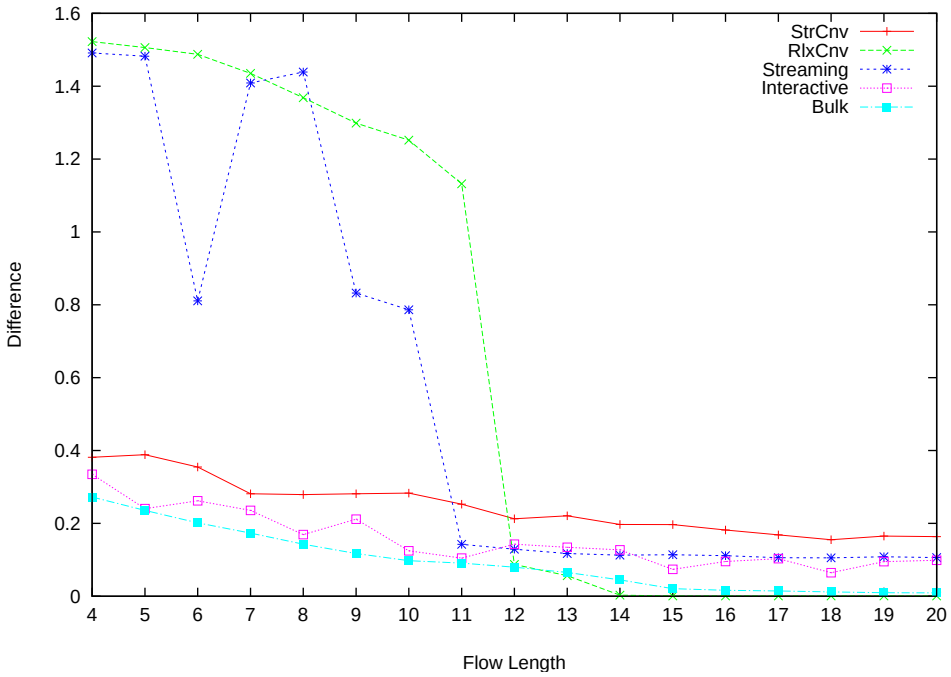


(a) Linear Scale

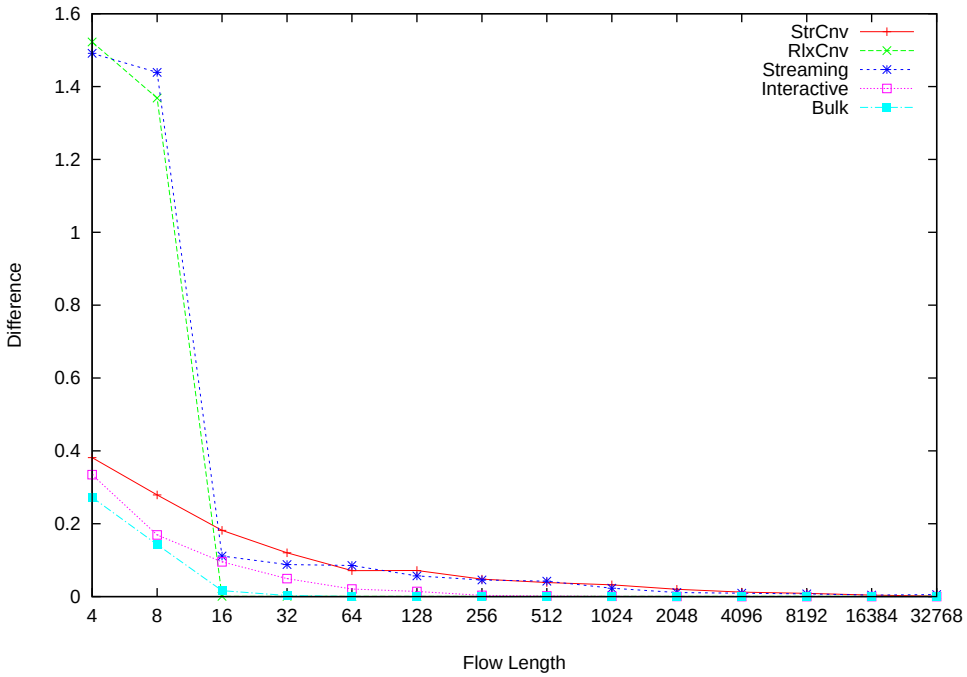


(b) Exponential Scale

Figure D.19: Difference/Prefix plots - pktSizeSD

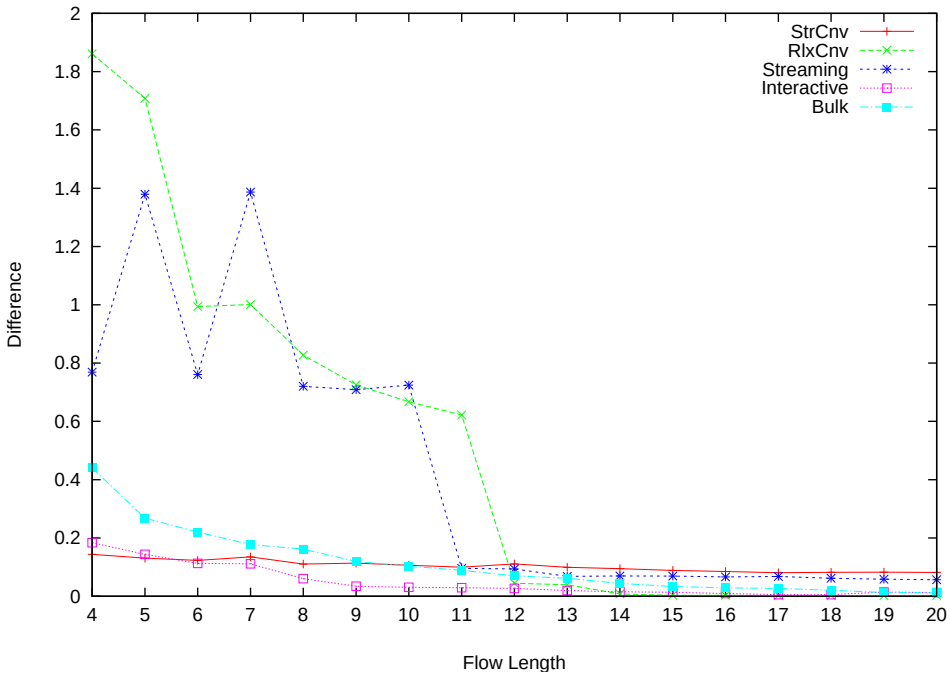


(a) Linear Scale

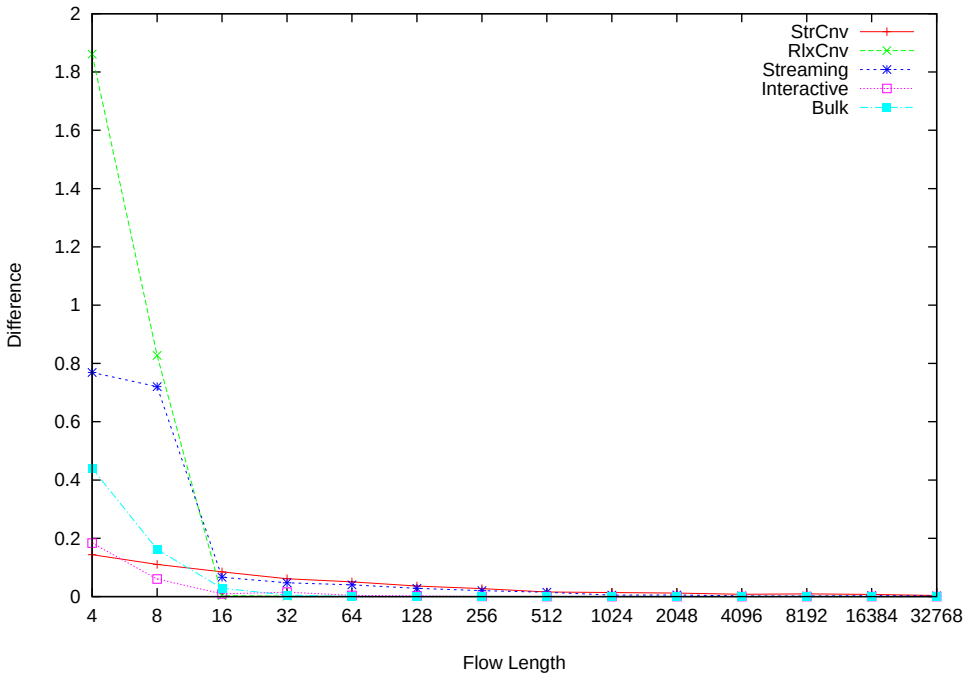


(b) Exponential Scale

Figure D.20: Difference/Prefix plots - pktSizeSDCF

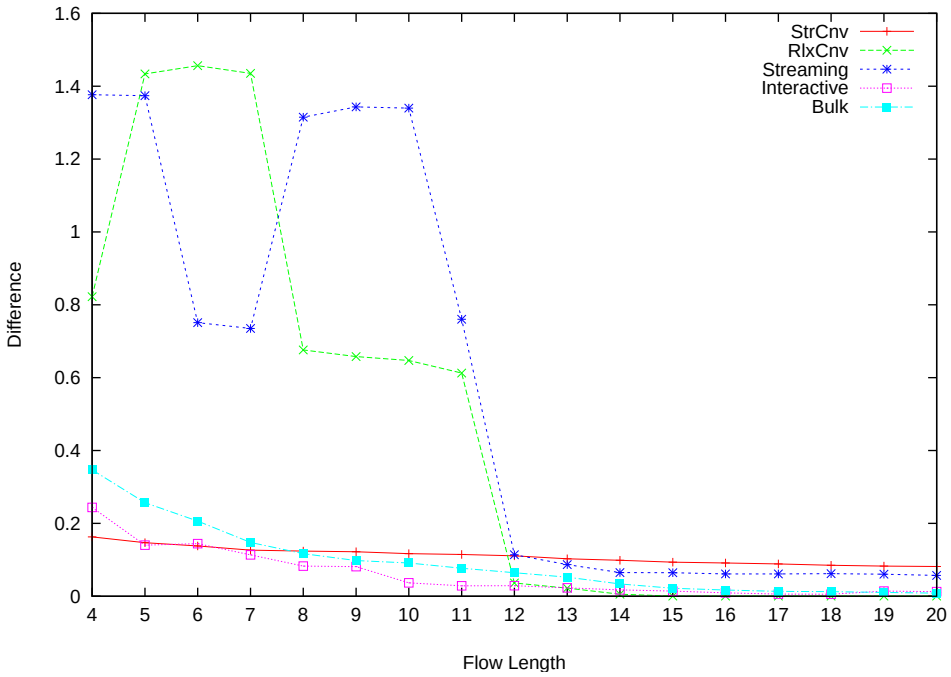


(a) Linear Scale

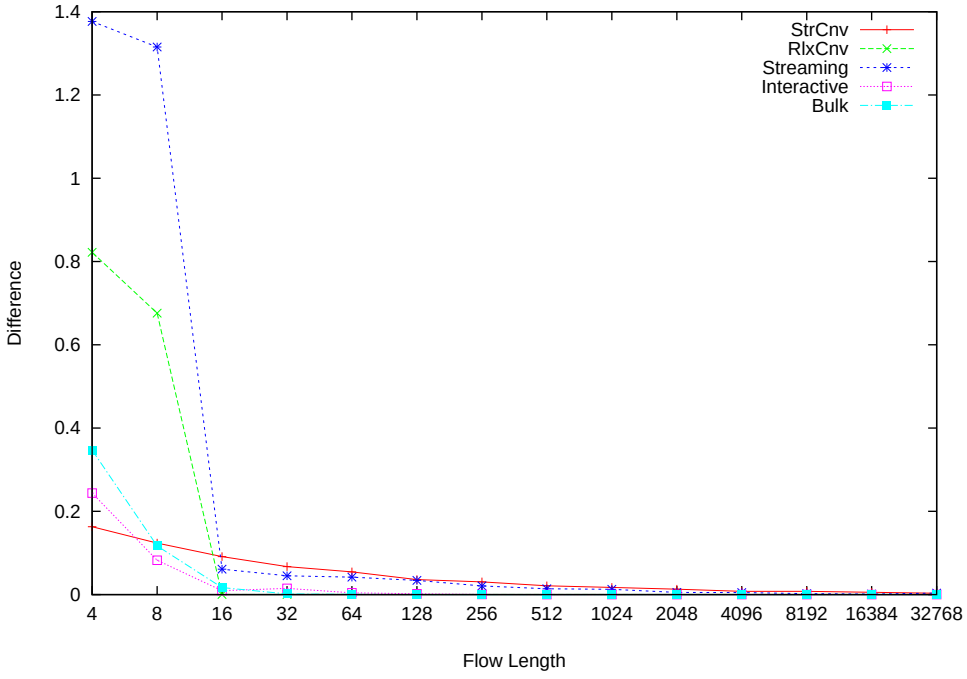


(b) Exponential Scale

Figure D.21: Difference/Prefix plots - pktSizeRMS

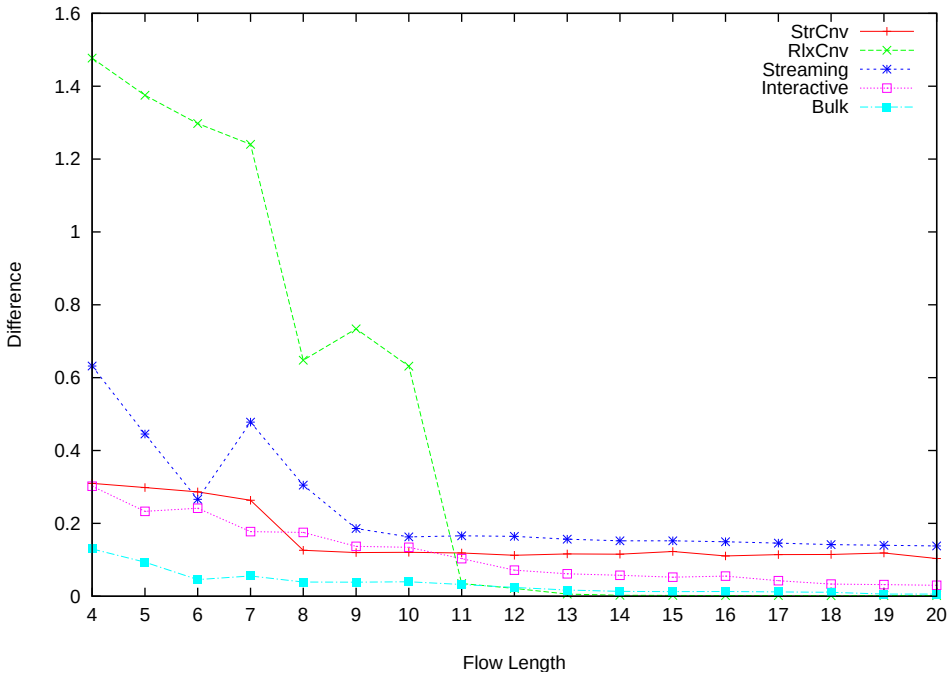


(a) Linear Scale

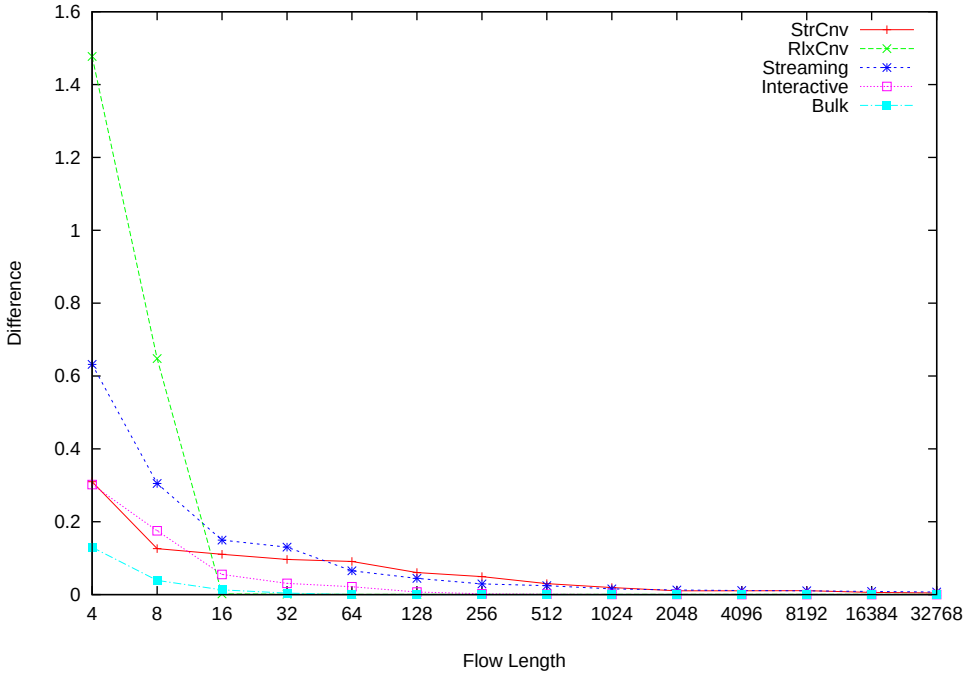


(b) Exponential Scale

Figure D.22: Difference/Prefix plots - pktSizeRMSCF

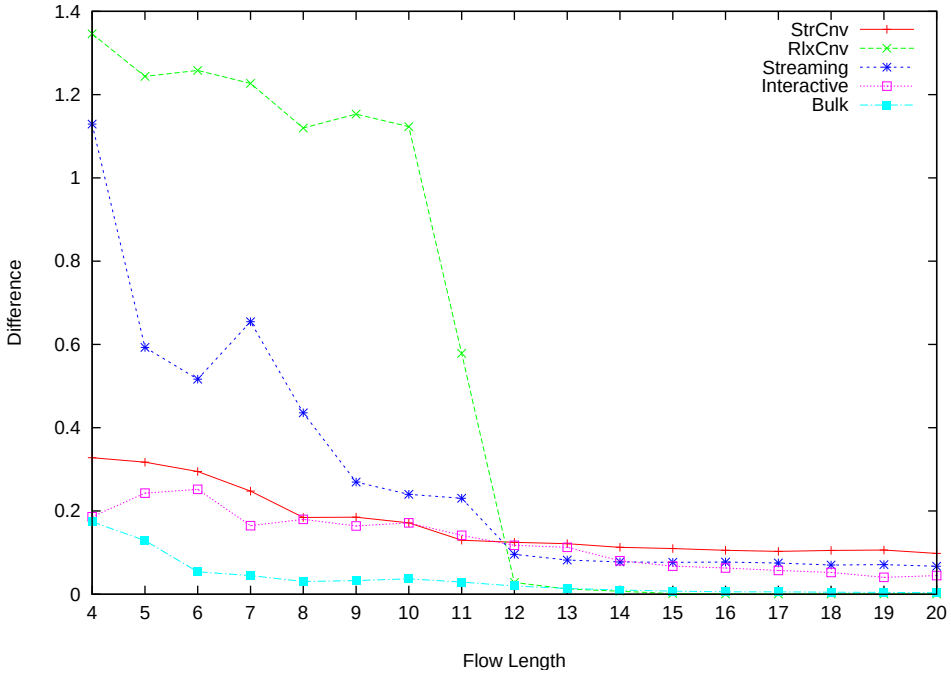


(a) Linear Scale

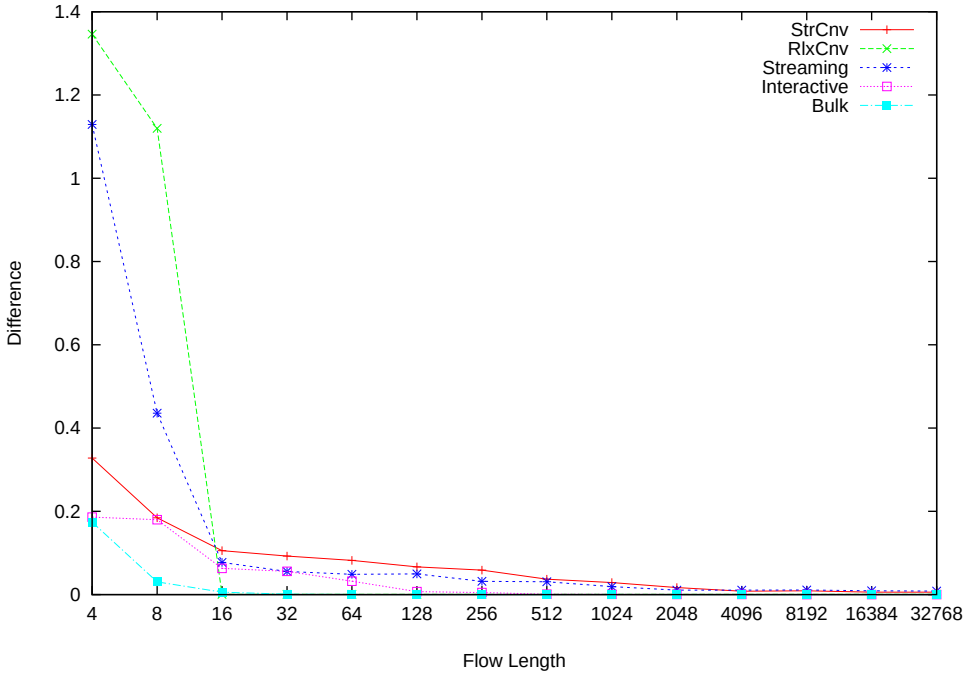


(b) Exponential Scale

Figure D.23: Difference/Prefix plots - dataTPUTAvg

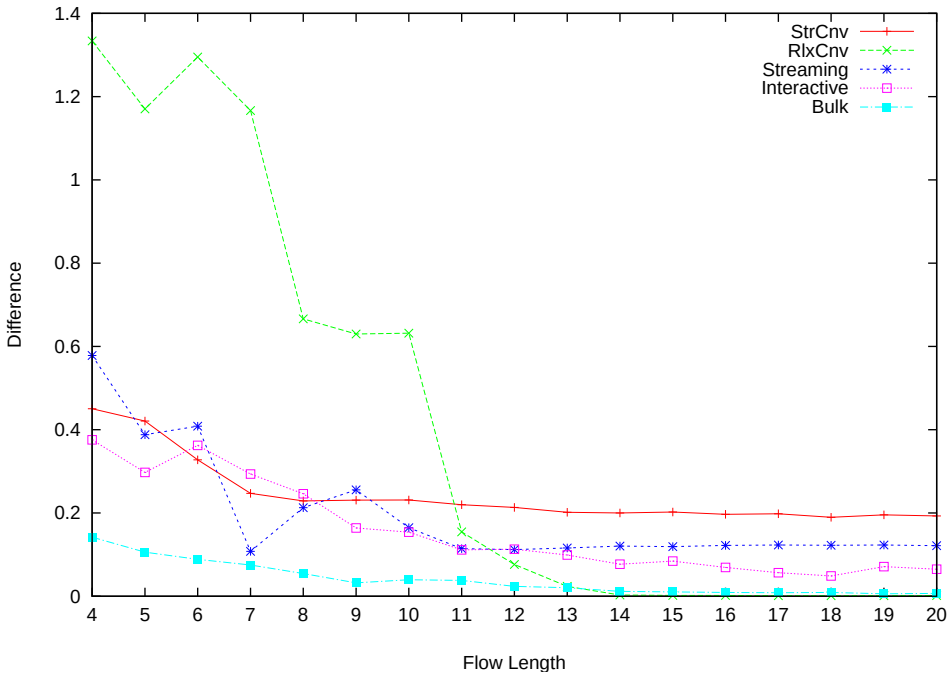


(a) Linear Scale

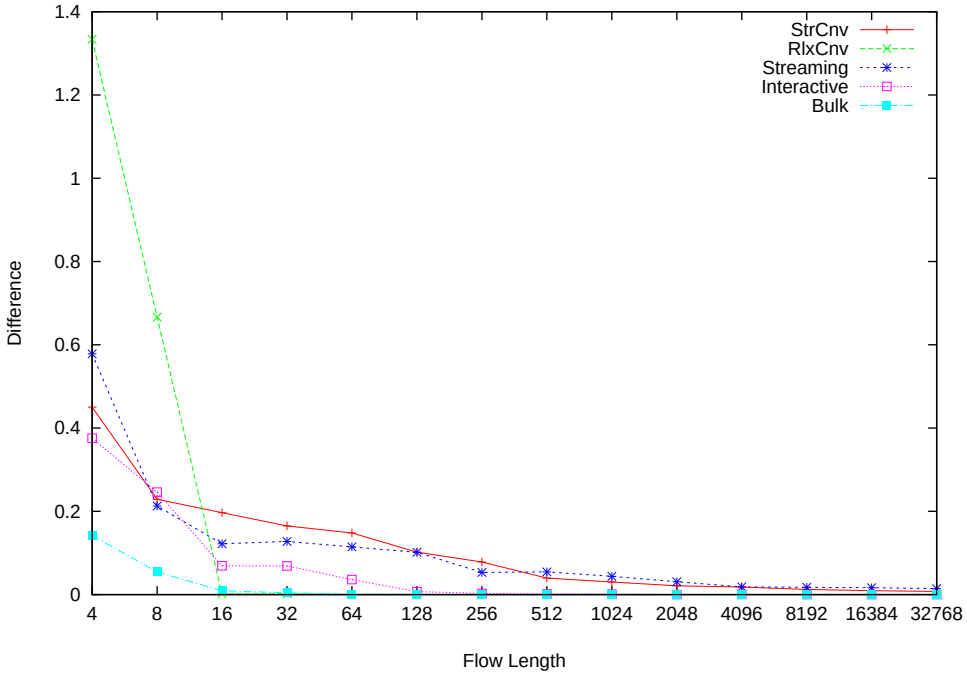


(b) Exponential Scale

Figure D.24: Difference/Prefix plots - dataTPUTAvgCF

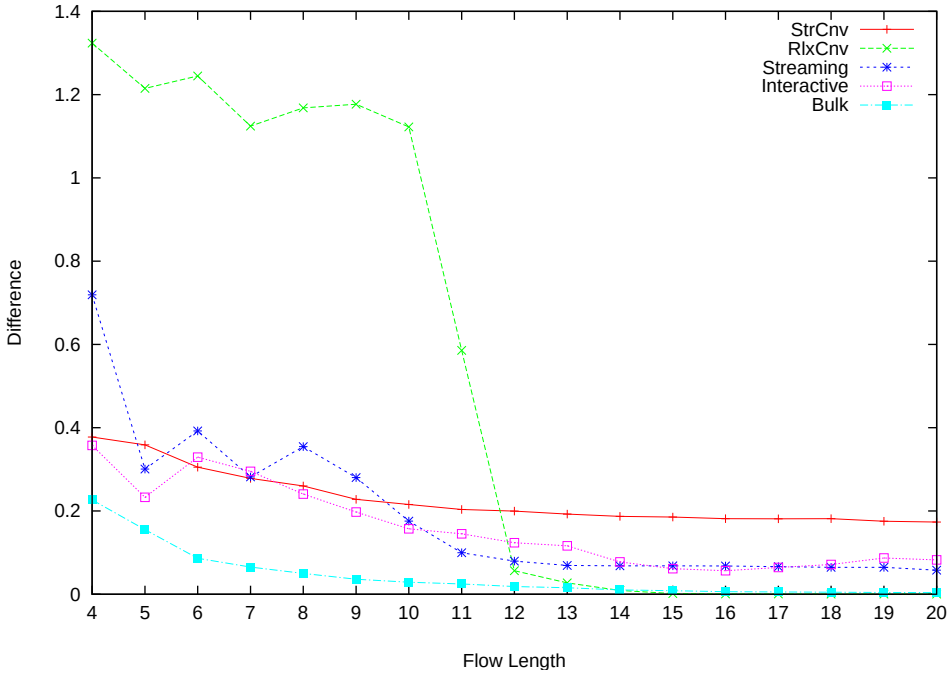


(a) Linear Scale

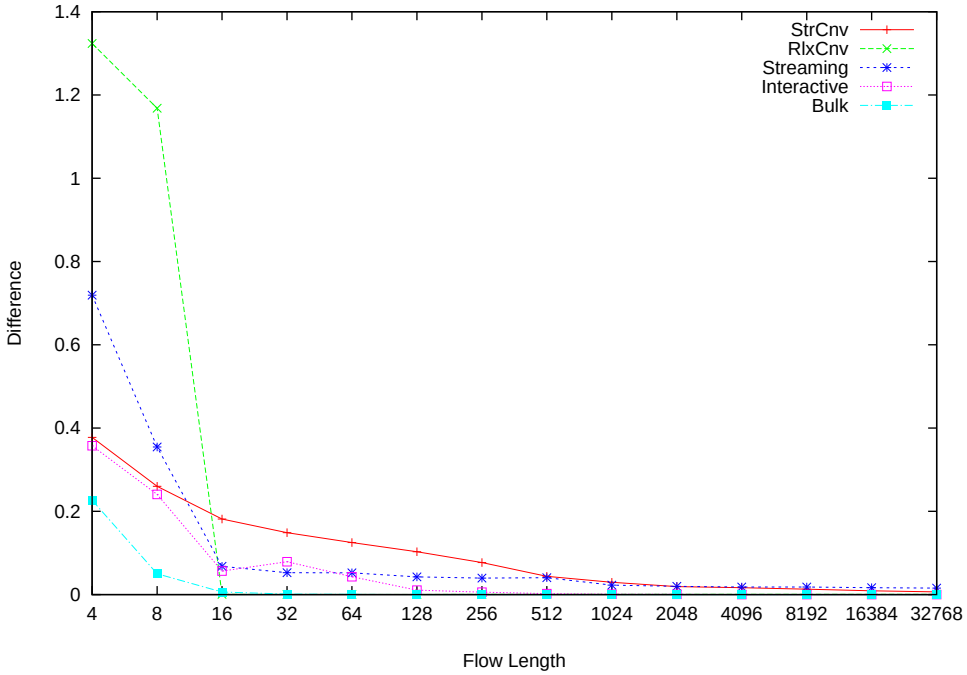


(b) Exponential Scale

Figure D.25: Difference/Prefix plots - pktTPUTAvg

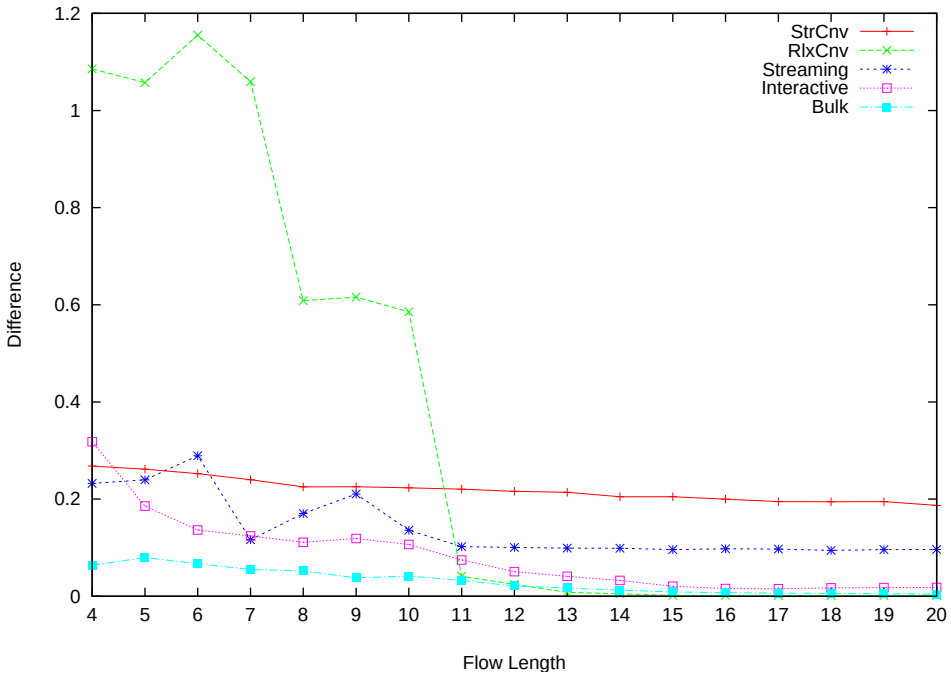


(a) Linear Scale

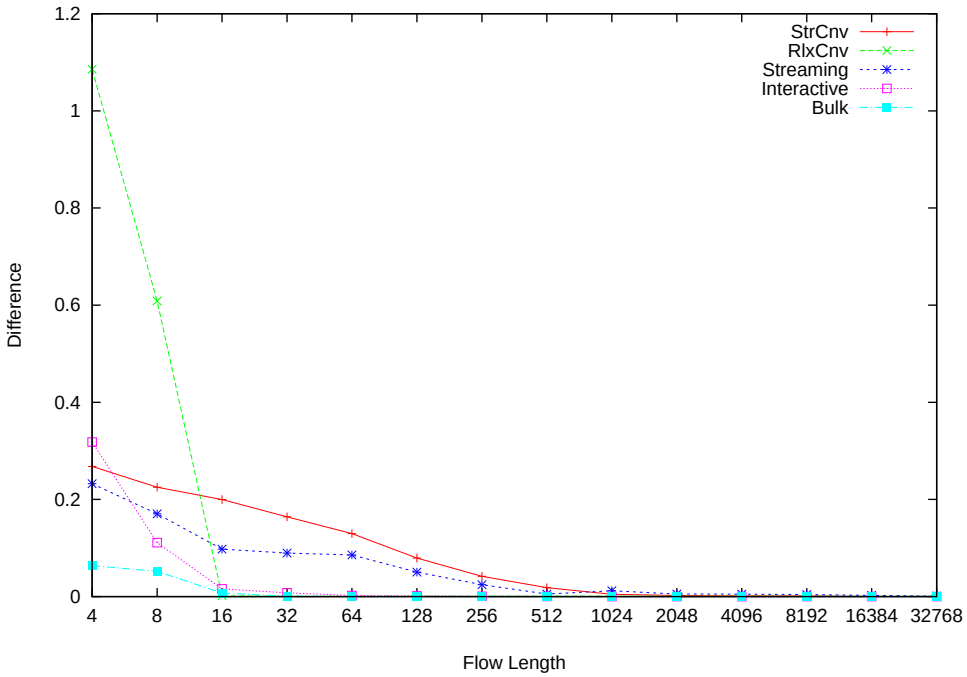


(b) Exponential Scale

Figure D.26: Difference/Prefix plots - pktTPUTAvgCF

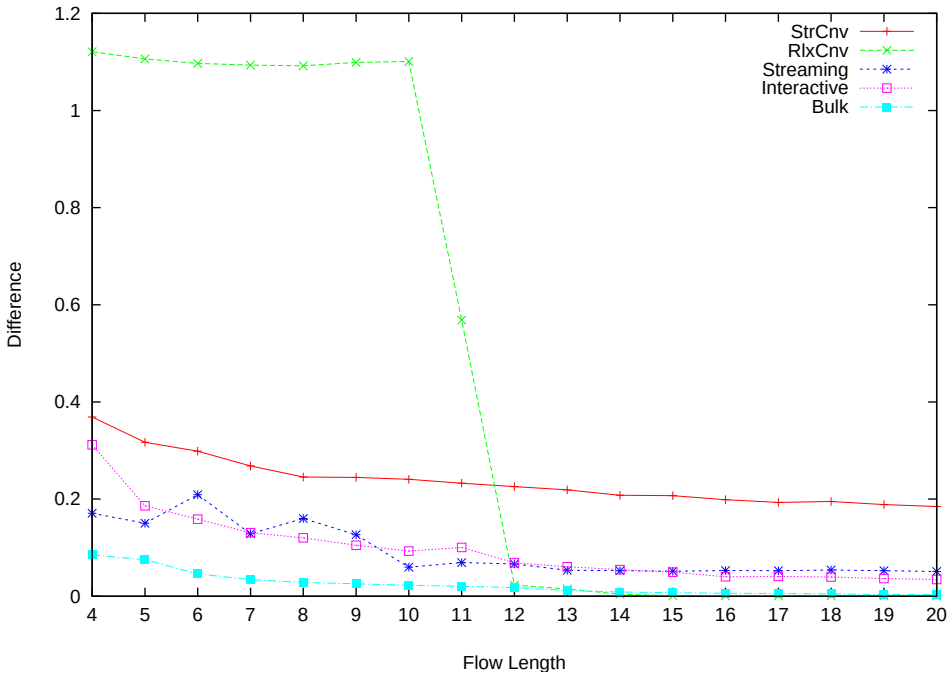


(a) Linear Scale

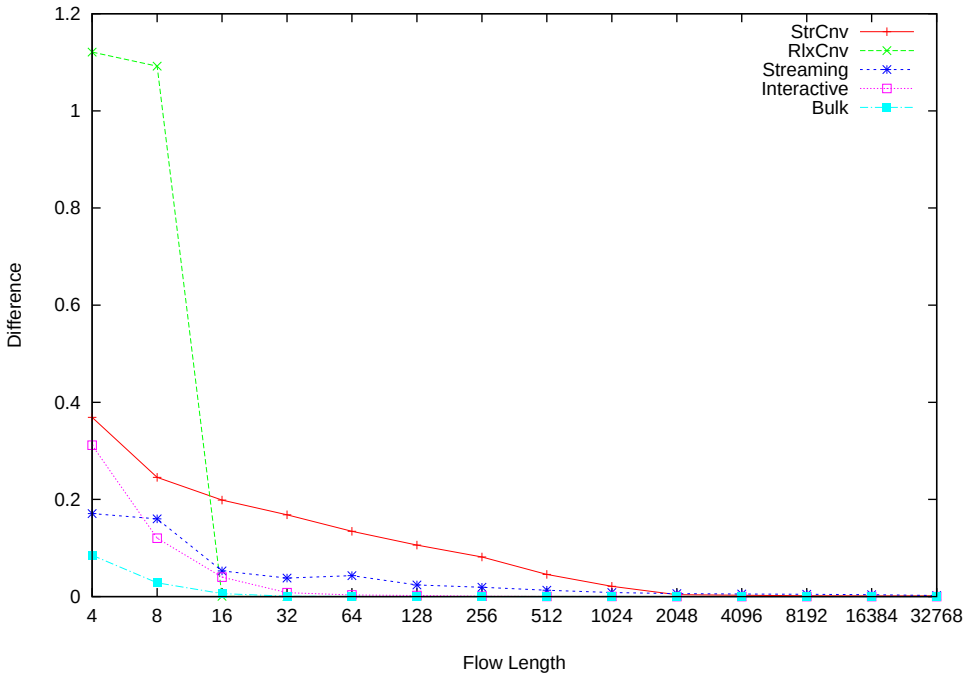


(b) Exponential Scale

Figure D.27: Difference/Prefix plots - iatAvg

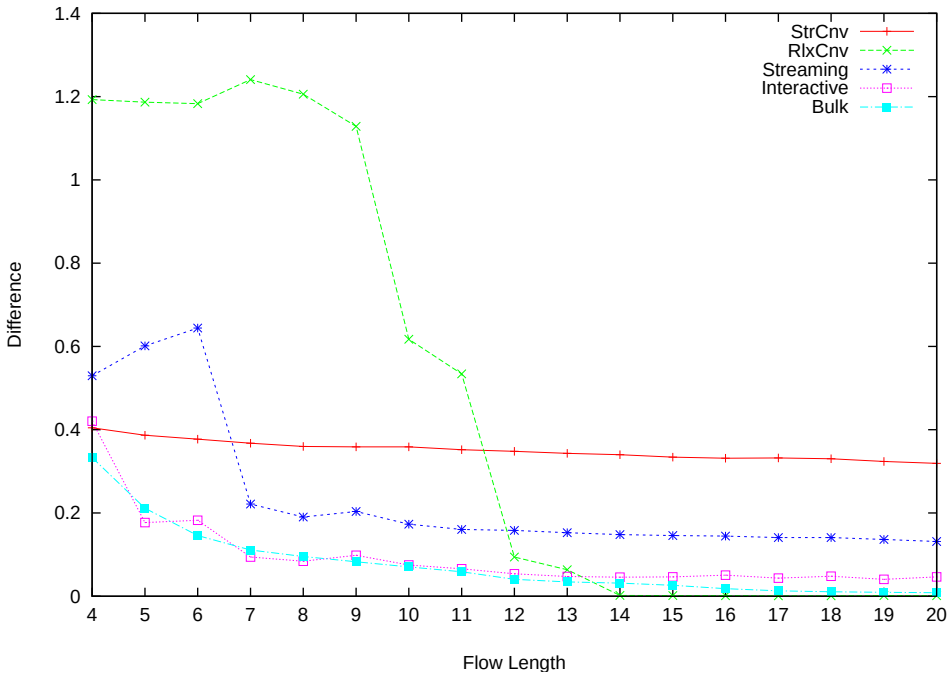


(a) Linear Scale

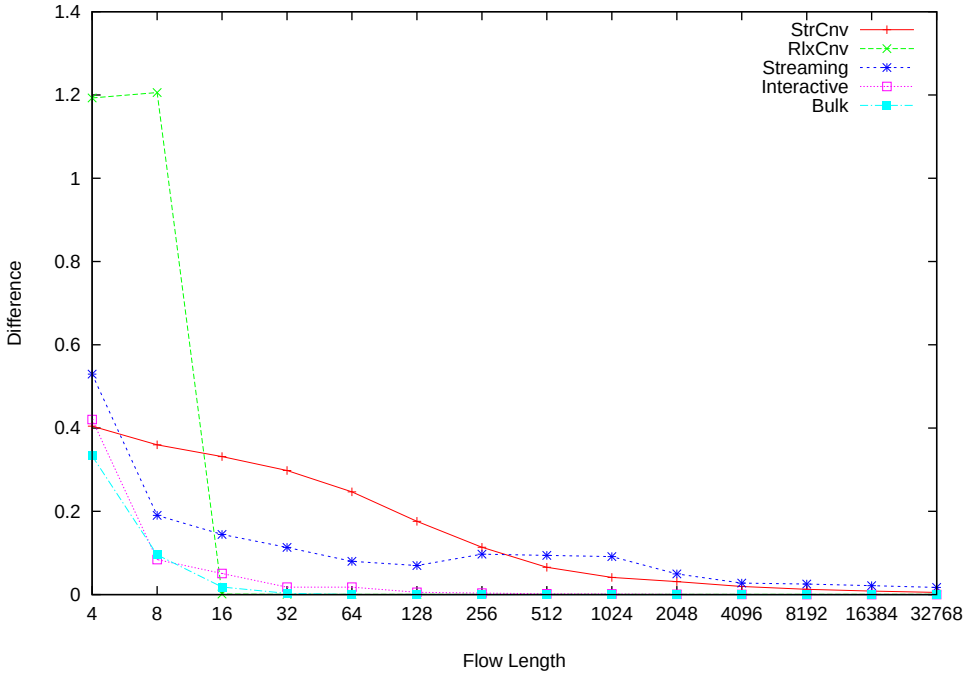


(b) Exponential Scale

Figure D.28: Difference/Prefix plots - iatAvgCF

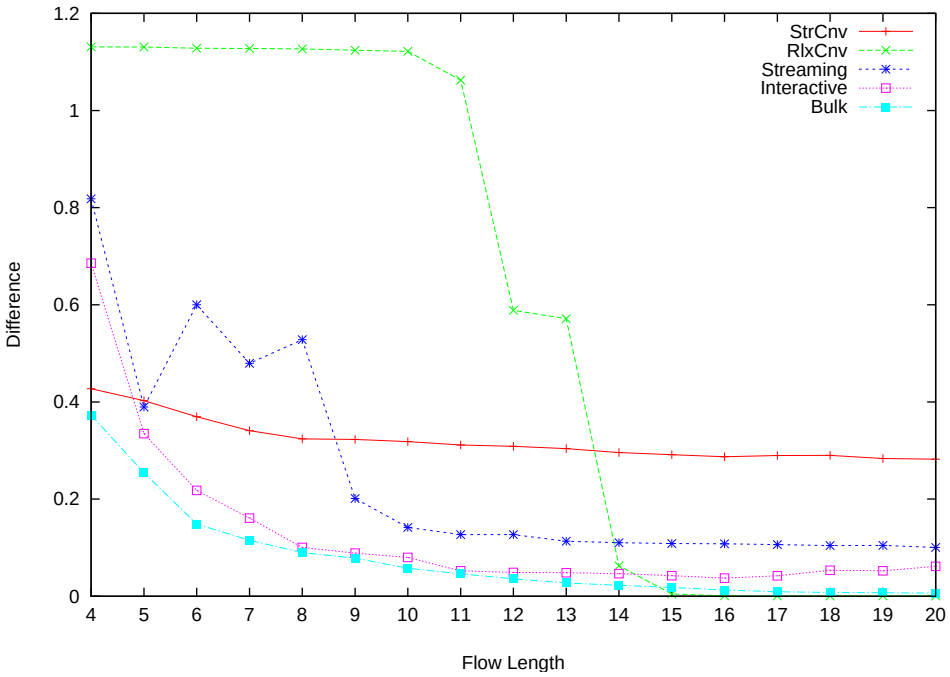


(a) Linear Scale

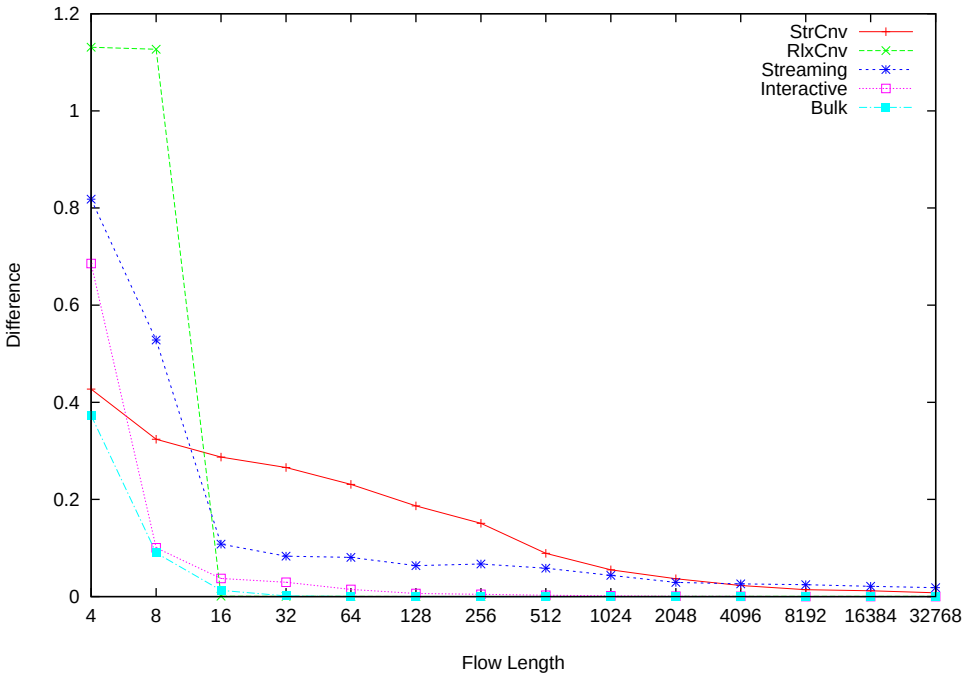


(b) Exponential Scale

Figure D.29: Difference/Prefix plots - iatSD

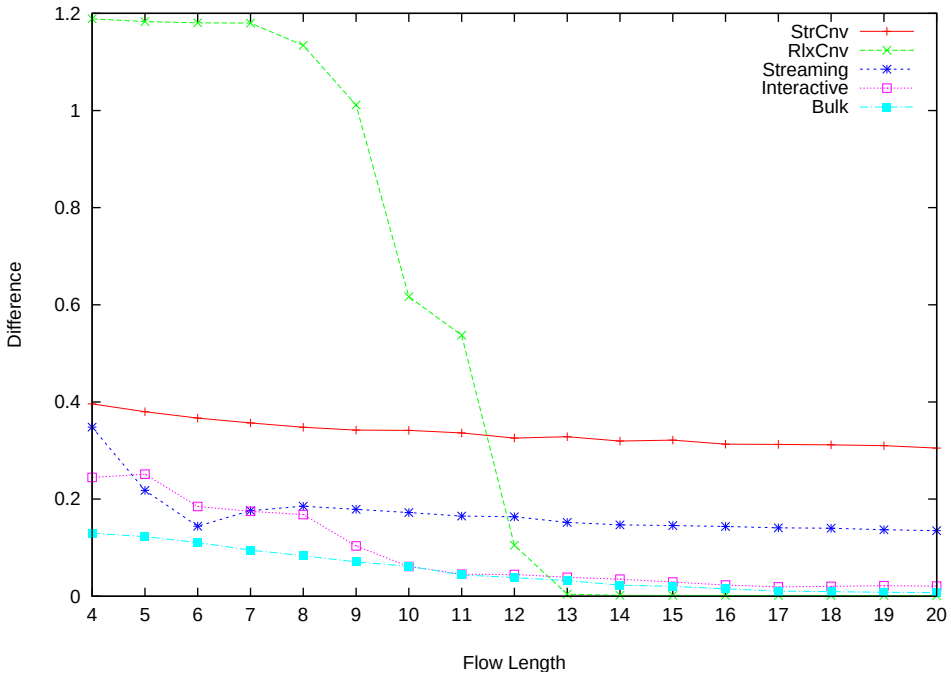


(a) Linear Scale

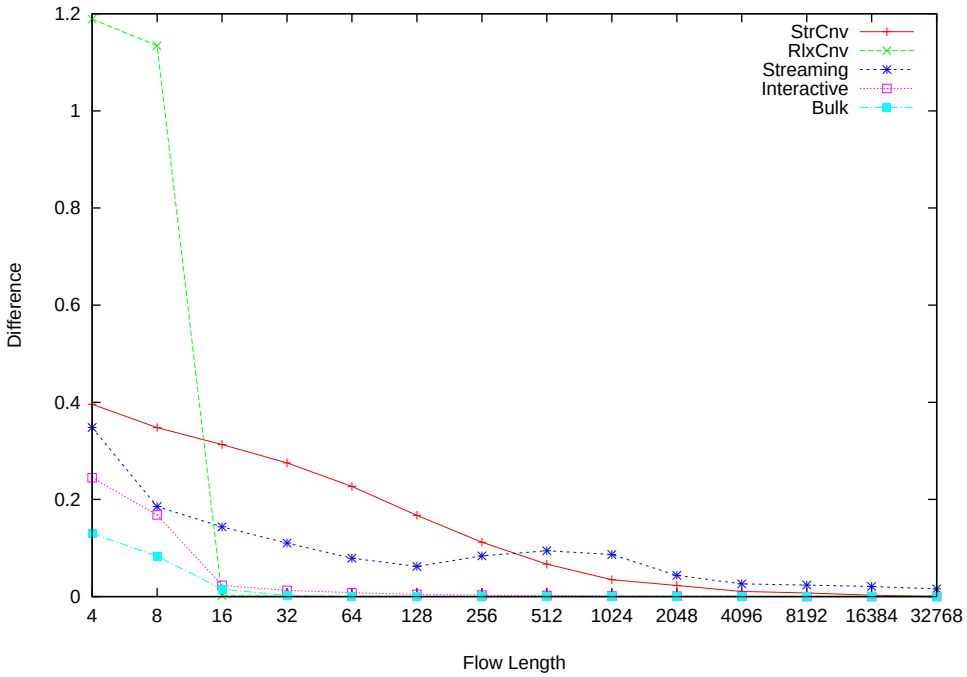


(b) Exponential Scale

Figure D.30: Difference/Prefix plots - iatSDCF

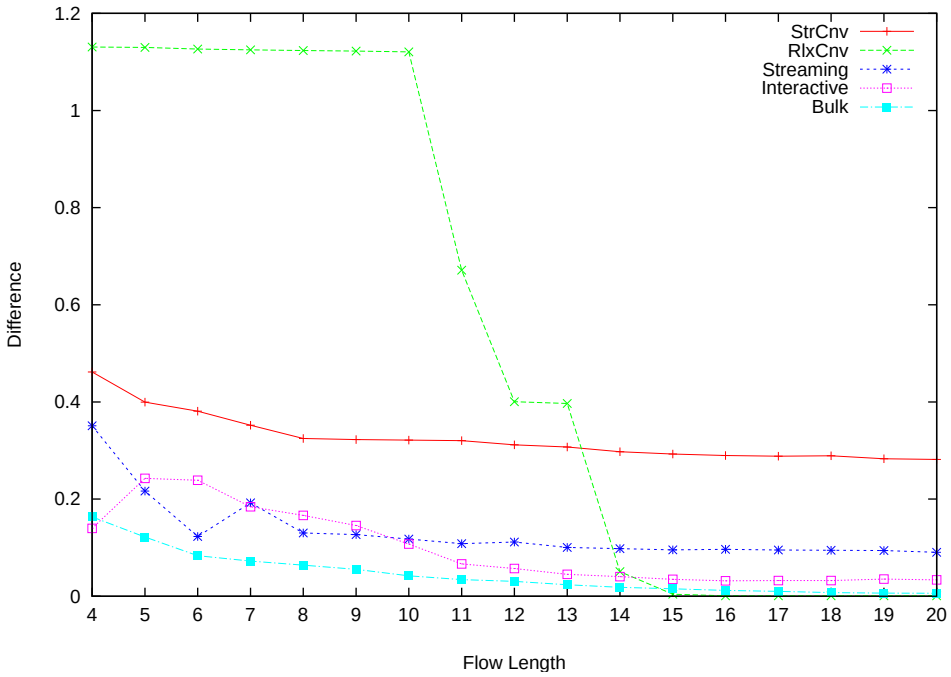


(a) Linear Scale

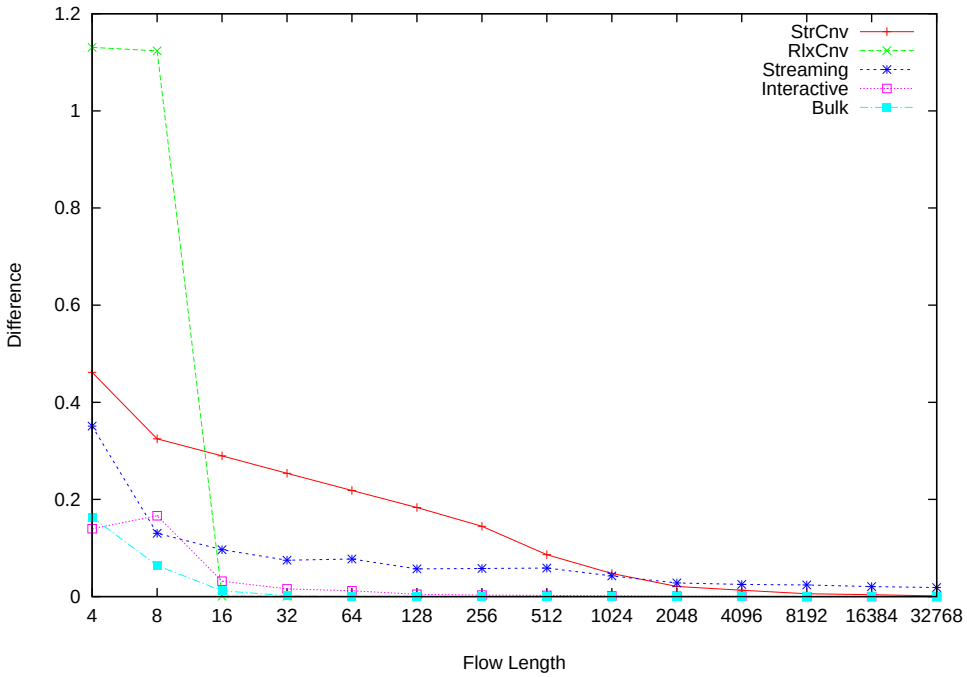


(b) Exponential Scale

Figure D.31: Difference/Prefix plots - iatRMS

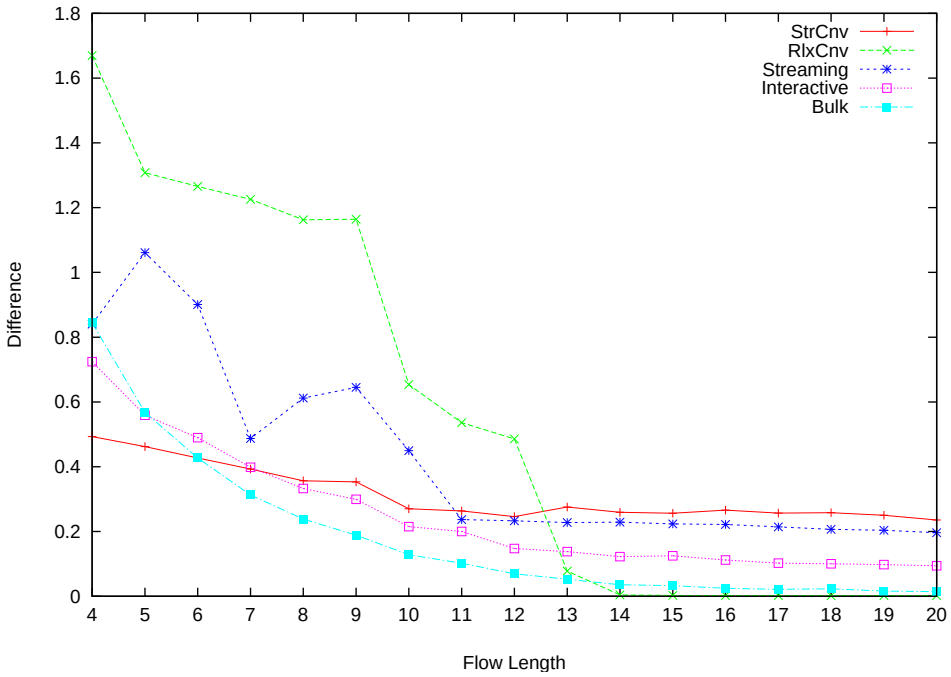


(a) Linear Scale

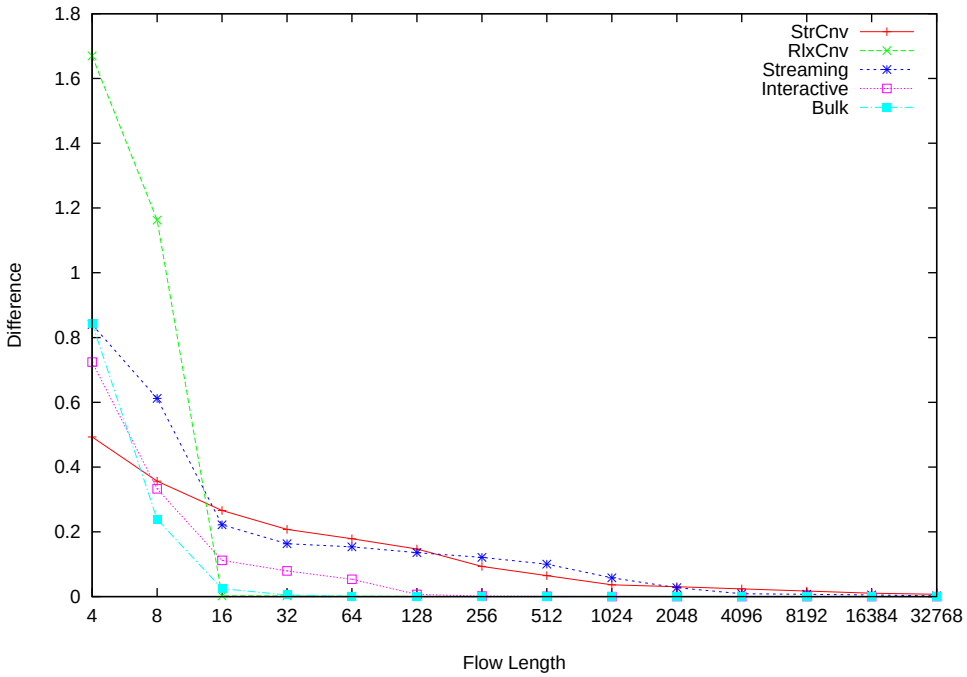


(b) Exponential Scale

Figure D.32: Difference/Prefix plots - iatRMSCF

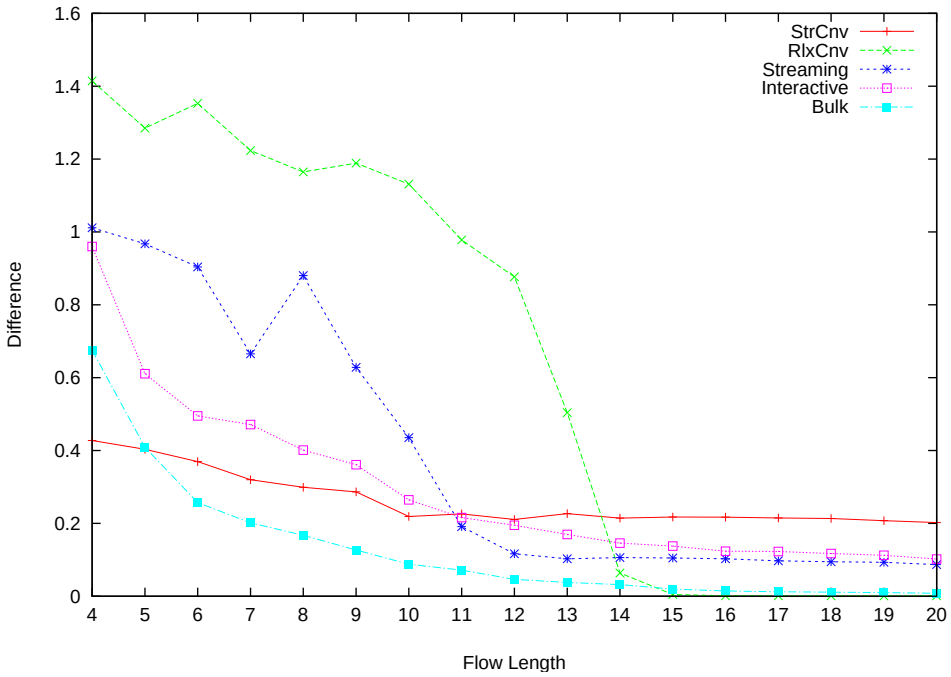


(a) Linear Scale

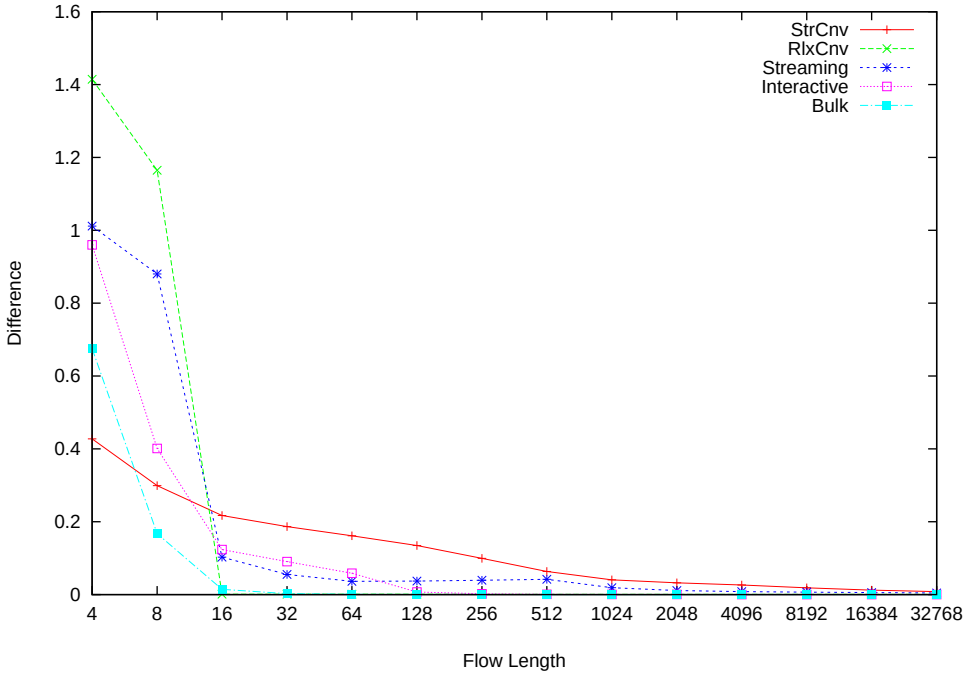


(b) Exponential Scale

Figure D.33: Difference/Prefix plots - iatVar



(a) Linear Scale



(b) Exponential Scale

Figure D.34: Difference/Prefix plots - iatVarCF

Bibliography

[3GP04] 3GPP; Technical Specification Group Services and System Aspects;. *TS 23.107: Quality of Service (QoS) Concept and Architecture (Release 6)*. 3rd Generation Partnership Project, December 2004. pages 28, 29, 75

[80204] IEEE. *802.16: Air Interface for Fixed Broadband Wireless Access Systems*, October 2004. pages 10

[80205] IEEE. *802.3: Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications*, December 2005. pages 15

[Ad03] S. Acid and L. M. de Campos. Searching for bayesian network structures in the space of restricted acylic partially directed graphs. *Journal of Artificial Intelligence Research*, 18:445–490, 2003. pages 68

[Ado] Adobe Systems Incorporated. *The Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8*. pages 144

[Aha97] David W. Aha. *Lazy learning*. Kluwer Academic Publishers, 1997. pages 68, 70

[AKA91] David W. Aha, Dennis Kibler, and Marc K. Albert. Instance-based learning algorithms. *Machine Learning*, 6(1):37–66, January 1991. pages 68

[Ana10] Isara Anantavrasilp. Adaptive ip traffic / flow classification system. In *Eighth International Network Conference*, 2010. pages 64, 79

[APA+05] S. Antonatos, M. Polychronakis, P. Akritidis, K. G. Anagnostakis, and E. P. Markatos. Piranha: A fast lookup pattern matching algorithm for intrusion detection. In *IFIP International Information Security Conference*, 2005. pages 22

[AS07a] Isara Anantavrasilp and Thorsten Schöler. Automatic flow classification using machine learning. In *15th International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, pages 1–6, 2007. pages 20, 64, 79, 81, 84, 102

[AS07b] Isara Anantavrasilp and Thorsten Schöler. Providing quality-of-service support to legacy applications using machine learning. In *IADIS International Conference on Telecommunications, Networks and Systems*, pages 75–83, 2007. pages 20, 64, 84

[ATS04] Stephanos Androutsellis-Theotokis and Diomidis Spinellis. A survey of peer-to-peer content distribution technologies. *ACM Computing Surveys (CSUR)*, 36(4):335 – 371, 2004. pages 146

[AV98] J. Altmann and P. Varaiya. Index project: user support for buying qos with regard to user's preferences. In *6th International Workshop on Quality of Service*, pages 101–104, 1998. pages 19

[BBC+98] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. *RFC 2475: An Architecture for Differentiated Services.* IETF, 1998. pages 17, 18, 25

[BCB06] J. Babiarz, K. Chan, and F. Baker. *RFC 4594: Configuration Guidelines for DiffServ Service Classes.* IETF, August 2006. pages 28, 29, 75

[BCS94] R. Braden, D. Clark, and S. Shenker. *RFC 1633: Integrated Services in the Internet Architecture: an Overview.* IETF, 1994. pages 17, 28

[Ber03] Marcus Bergner. Improving performance of modern peer-to-peer services. Master's thesis, Umea University, 2003. pages 25

[Ber07] Laurent Bernaille. *Classification temps réel d'applications sur l'Internet.* PhD thesis, Pierre and Marie Curie University, 2007. pages 124

[BFSO84] Leo Breiman, Jerome Friedman, Charles J. Stone, and R.A. Olshen. *Classification and Regression Trees.* Chapman & Hall/CRC, 1984. pages 56

[Bha43] A. Bhattacharyya. On a measure of divergence between two statistical populations defined by probability distributions. *Bulletin of the Calcutta Mathematical Society*, 35:99–109, 1943. pages 56, 107

[BHH08] Sebastian Bader, Pascal Hitzler, and Steffen Hölldobler. Connectionist model generation: A first-order approach. *Neurocomputing*, 71:2420–2432, 2008. pages 71

[Bis08] Christopher M. Bishop. *Pattern Recognition and Machine Learning.* Springer Verlag, 2008. pages 72

[BLFF] T. Berners-Lee, R. Fielding, and H. Frystyk. *RFC 1945: Hypertext Transfer Protocol - HTTP/1.0.* IETF. pages 11

[Bli89] Wayne D. Blizard. Multiset theory. *Notre Dame Journal of Formal Logic*, 30(1):36–66, 1989. pages 42

[BN92] W. Buntine and T. Niblett. A further comparison of splitting rules for decision-tree induction. *Machine Learning*, 8(1):75–85, 1992. pages 56, 57

[Bra89] R. Braden. *RFC 1122: Requirements for Internet Hosts – Communication Layers*. IETF, 1989. pages 11

[Bre01] Leo Breiman. Random forest. *Machine Learning*, 45(1):5–32, 2001. pages 71

[BSD00] A. Bouch, M. A. Sasse, and H. DeMeer. Of packets and people: A user-centered approach to quality of service. In *International Workshop on Quality of Service*, 2000. pages 17

[BSTW95] J. Beran, R. Sherman, M.S. Taqqu, and W. Willinger. Long-range dependence in variable-bit-rate video traffic. *IEEE Transactions on Communications*, 43(234):1566–1579, Feb/Mar/Apr 1995. pages 116

[BTA+06] Laurent Bernaille, Renata Teixeira, Ismael Akodkenou, Augustin Soule, and Kave Salamatian. Traffic classification on the fly. *ACM SIGCOMM Computer and Communication Review*, 36(2):23–26, April 2006. pages 23, 24, 25, 26, 79, 116, 129

[BTS06] Laurent Bernaille, Renata Teixeira, and Kav Salamatian. Early application identification. In *2nd Conference on Future Networking Technologies (CoNext 2006)*, 2006. pages 20, 23, 24, 25, 71, 79, 119, 124

[Cam96] Andrew T. Campbell. *A Quality of Service Architecture*. PhD thesis, Computing Department, Lancaster University, January 1996. pages 19

[Ces90] B. Cestnik. Estimating probabilities: A crucial task in machine learning. In *Ninth European Conference on Artificial Intelligence*, 1990. pages 63, 66

[CH67] T. M. Cover and P. E. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13:21–27, 1967. pages 68, 69

[Che08] Checkpoint software technologies. `http://www.checkpoint.com`, January 2008. pages 22, 25

[Chi02] D. M. Chickering. Optimal structure identification with greedy search. *Journal of Machine Learning Research*, 3:507–554, 2002. pages 68

[Cho08] Kenjiro Cho. Wide-transit 150 megabit ethernet trace 2008-03-18 (collection), 2008. `http://imdc.datcat.org/collection/1-05L9-X=WIDE-TRANSIT+150+Megabit+Ethernet+Trace+2008-03-18` (accessed on 2008-07-08). pages 94

[Cis08] Cisco ips 4200 series sensors. `http://www.cisco.com/en/US/products/hw/vpndevc/ps4077/index.html`, January 2008. pages 22, 25

[CKB87] B. Cestnik, I. Kononenko, and I. Bratko. Assistant 86: A knowledge elicitation tool for sophisticated users. In *the Second European Working Session on Learning*, 1987. pages 64

[Cla01] A. D. Clark. Modeling the effects of burst packet loss on recency of subjective voicequality. In *IP Telephony Workshop*, 2001. pages 17

[CLR67] I. M. Chakravarti, R. G. Laha, and J. Roy. *Handbook of Methods of Applied Statistics*, volume I. John Wiley and Sons, USE, 1967. pages 56

[CN89] P. Clark and T. Niblett. The cn2 induction algorithm. *Machine Learning*, 3:261–283, 1989. pages 63

[Coh95] W. W. Cohen. Fast effective rule induction. In *Proceedings of the Twelfth International Conference on Machine Learning*, 1995. pages 61, 63, 126

[Coh03] Bram Cohen. Incentives build robustness in bittorrent. In *First Workshop on the Economics of Peer-to-Peer Systems*, June 2003. pages 25

[CT91] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory.* Wiley-Interscience, 1991. pages 106, 107

[dA98] R. Lopez de Mantaras and E. Armengol. Machine learning from examples: Inductive and lazy methods. *Data and Knowledge Engineering Journal*, 25:99–123, 1998. pages 68

[Dav09] Jörn David. Noise robust classification based on spread spectrum. In *SIAM International Conference on Data Mining*, 2009. pages 71

[DC02] K. Dharmalingam and M. Collier. Transparent qos support of network applications using netlets. *Lecture Notes in Computer Science*, 2521:206–215, 2002. pages 19

[DCB+02] B. Davie, A. Charny, J.C.R. Bennett, K. Benson, J.Y. Le Boudec, W. Courtney, S. Davari, V. Firoiu, and D. Stiliadis. *RFC 3246: An Expedited Forwarding PHB (Per-Hop Behavior)*, 2002. pages 27

[DH98] S. Deering and R. Hinden. *RFC 2460: Internet Protocol, Version 6 (IPv6) Specification.* IETF, December 1998. pages 31

[Dir07] Direct connect. `http://dcplusplus.sourceforge.net/`, November 2007. pages 25

[DKSL03] Sarang Dharmapurikar, Praveen Krishnamurthy, Todd Sproull, and John Lockwood. Deep packet inspection using parallel bloom filters. In *The 11 th Symposium on High Performance Interconnects*, 2003. pages 22

[DL97] M. Dash and H. Liu. Feature selection for classification. *Intelligent Data Analysis*, 1(3):131Ãś–156, 1997. pages 122, 123

[DP97] Pedro Domingos and Michael J. Pazzani. On the optimality of the simple bayesian classifier under zero-one loss. *Machine Learning*, 29(2-3):103–130, 1997. pages 66

[DWF03] C. Dewes, A. Wichmann, and A. Feldmann. An analysis of internet chat systems. In *3rd ACM/SIGCOMM Internet Measurement Conference*, 2003. pages 25, 78, 82

[E.894] ITU-Recommendation. *E.800: Quality of Service and Dependability Vocabulary*, 1994. pages 17

[EBR03] James P. Early, Carla E. Brodley, and Catherine Rosenberg. Behavioral authentication of server flows. In *19th Annual Computer Security Applications Conference*, 2003. pages 23, 25, 26, 45, 60

[eDo07] edonkey2000. `http://en.wikipedia.org/wiki/EDonkey2000`, November 2007. pages 25

[Elm90] J. L. Elman. Finding structure in time. *Cognitive Science*, 14:179–211, 1990. pages 71

[EMA+07] Jeffrey Erman, Anirban Mahanti, Martin Arlitt, Ira Cohen, and Carey Williamson. Offline/realtime traffic classification using semi-supervised learning. *Performance Evaluation*, 64(9-12):1194–1213, October 2007. pages 24, 25, 26, 27, 71, 116, 119, 124, 129, 141

[EN01] M. Eder and S. Nag. *RFC 3052: Service Management Architectures Issues and Review*. IETF, 2001. pages 16

[End04] Endace Measurement Systems. *DAG 4.3GE Installation Guide*, March 2004. pages 92

[Epi07] Epic Games. Unreal. `http://www.unreal.com`, June 2007. pages 75

[EST02] ESTI. *TR 101 329-5: "Telecommunications and Internet Protocol Harmonization OverNetworks (TIPHON) Release 3; End-to-end Quality of Service in TIPHON Systems,Part 5: Quality-of-Service (QoS) Measurement Methodologies"*, 2002. pages 17

[FB96a] N. Freed and N. Borenstein. *RFC 2045: Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies*. Network Working Group, November 1996. pages 144

[FB96b] N. Freed and N. Borenstein. *RFC 2046: Multipurpose Internet Mail Extensions (MIME) Part One: Media Types*. Network Working Group, November 1996. pages 144

[FBF77] J. H. Friedman, J. L. Bentley, and R. A. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software*, 3:209–226, 1977. pages 70

[FF03] Johannes Fürnkranz and Peter Flach. An analysis of rule evaluation metrics. In *Twentieth International Conference on Machine Learning*, 2003. pages 63

[FGM+] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. *RFC 2616: Hypertext Transfer Protocol – HTTP/1.1.* IETF. pages 11, 16

[Flo96] P. Florissi. *QoSME: QoS Management Environment.* PhD thesis, Department of Computer Science, Columbia University, 1996. pages 19

[Fu94] L. M. Fu. Rule generation from neural networks. *IEEE Transactions on Systems, Man and Cybernetics*, 28(8):1114–1124, 1994. pages 71

[FW94] J. Fürnkranz and G. Widmer. Incremental reduced-error pruning. In *the Eleventh International Conference on Machine Learning*, 1994. pages 61, 63

[FW98] Eibe Frank and Ian H. Witten. Generating accurate rule sets without global optimization. In J. Shavlik, editor, *Machine Learning: Proceedings of the Fifteenth International Conference*, 1998. pages 43, 64, 126

[G.101a] ITU-Recommendation. *G.1000: Communications Quality of Service: A framework and definitions*, November 2001. pages 17

[G.101b] ITU-Recommendation. *G.1010: Quality of Service and Performance*, November 2001. pages 28, 29, 75

[Gal93] S.I. Gallant. *Neural Network Learning and Expert Systems.* MIT Press, 1993. pages 71

[GE03] Isabelle Guyon and André Elisseeff. An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3:1157–1182, 2003. pages 123

[GH99] D. Grossman and J. Heinanen. *RFC 2684: Multiprotocol Encapsulation over ATM Adaptation Layer 5.* IETF, September 1999. pages 15

[Ghi64] E. E. Ghiselli. *Theory of Psychological Measurement.* McGraw-Hill, 1964. pages 123

[GJ09] K.P. Girish and Sunil Jacob John. Relations and functions in multiset context. *Information Sciences*, 179(6):758 – 768, 2009. pages 42

[Goo08] Google video. `http://video.google.com/`, June 2008. pages 144

[Gro02] D. Grossman. *RFC 3260: New Terminology and Clarifications for Diffserv.* IETF, 2002. pages 18

[GW94] Mark W. Garrett and Walter Willinger. Analysis, modeling, and generation of self-similar vbr video traffic. In *ACM SIGCOMM*, 1994. pages 116

[H.306] ITU-Recommendation. *H.323: Packet-Based Multimedia Communications Systems*, June 2006. pages 10

[Hal98] M. Hall. *Correlation-based Feature Selection for Machine Learning*. PhD thesis, Department of Computer Science, Waikato University, 1998. pages 123, 124, 126

[Hal00] M. Hall. Correlation-based feature selection for discrete and numeric class machine learning. In *17th International Conference on Machine Learning*, 2000. pages 123

[HBWW99] J. Heinanen, F. Baker, W. Weiss, and J. Wroclawski. *RFC 2597: Assured Forwarding PHB Group*. IETF, 1999. pages 27

[HFH+09] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The weka data mining software: An update. *SIGKDD Explorations*, 11(1):10–18, 2009. pages 88

[HMC99] D. Heckerman, C. Meek, and G. Coope. *A Bayesian Approach to Causal Discovery*, pages 141–165. MIT Press, 1999. pages 68

[HMS66] E. Hunt, J. Martin, and P. Stone. *Experiments in Induction*. Academic Press, 1966. pages 56

[Hop82] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 79:2554–2558, 1982. pages 71

[HSSW05] Patrick Haffner, Subhabrata Sen, Oliver Spatscheck, and Dongmei Wang. Acas: Automated construction of application signatures. In *ACM SIGCOMM Workshop on Mining Network Data*, 2005. pages 22

[HX02] Samuel H. Huang and Hao Xing. Extract intelligible and concise fuzzy rules from neural networks. *Fuzzy Sets and Systems*, 132(2):233–243, 2002. pages 71

[I.199] ITU-Recommendation. *I.150: B-ISDN Asynchronous Transfer Mode Functional Characteristics*, February 1999. pages 15

[Int07] Internet Assigned Numbers Authority (IANA). Port numbers. `http://www.iana.org/assignments/port-numbers`, October 2007. pages 21, 22

[JGT01] S. P. Jena, S. K. Ghosh, and B. K. Tripathy. On the theory of bags and lists. *Information Sciences: an International Journal*, 132(1-4):241–254, 2001. pages 42

[JL95] George H. John and Pat Langley. Estimating continuous distributions in bayesian classifiers. In *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, pages 338–345, 1995. pages 66, 67

[JN02] Jingwen Jin and Klara Nahrstedt. Classification and comparison of qos specification languages for distributed multimedia applications. Technical Report UIUCDCS-R-2002-2302/UILU-ENG-2002-1745, Department of Computer Science, University of Illinois at Urbana-Champaign, 2002. pages 19

[JN04] Jingwen Jin and Klara Nahrstedt. Qos specification languages for distributed multimedia applications: A survey and taxonomy. *IEEE Multimedia*, 11(3):74–87, 2004. pages 19

[JNP99] V. Jacobson, K. Nichols, and K. Poduri. *RFC 2598: An Expedited Forwarding PHB*. IETF, 1999. pages 27

[KAD+04] Pradnya Karbhari, Mostafa Ammar, Amogh Dhamdhere, Himanshu Raj, George Riley, and El len Zegura. Bootstrapping in gnutella: A measurement study. In *The Fifth Anuual Passive & Active Measurement Workshop*, 2004. pages 25

[Kan91] B. Kantor. *RFC 1258: BSD Rlogin*. IETF, 1991. pages 25

[KBFC04] Thomas Karagiannis, Andre Broido, Michalis Faloutsos, and Kc Claffy. Transport layer identification of p2p traffic. In *Internet Measurement Conference*, 2004. pages 21

[Kil99] Kalevi Kilkki. *Differentiated Services for the Internet*. Sams, July 1999. pages 17

[KJ97] Ron Kohavi and George H. John. Wrappers for feature subset selection. *Artificial Intelligence*, 97(1-2):273–324, 1997. pages 122

[KMK+01] Ken Keys, David Moore, Ryan Koga, Edouard Lagache, Michael Tesch, and k Claffy. The architecture of coralreef: An internet traffic monitoring software suite. In *A Workshop on Passive and Active Measurements (PAM 2001)*, 2001. pages 92

[Koh95] Ron Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *IJCAI*, pages 1137–1145, 1995. pages 42, 73

[Kot07] S. B. Kotsiantis. Supervised machine learning: A review of classification techniques. *Informatica*, 31:249–268, 2007. pages 56, 57, 66, 69, 72

[KPF05] Thomas Karagiannis, Konstantina Papagiannaki, and Michalis Faloutsos. Blinc: multilevel traffic classification in the dark. In *SIGCOMM '05: Proceedings of the 2005 Conference on Applications, Technologies,*

Architectures, and Protocols for Computer Communications, pages 229–240, New York, NY, USA, 2005. ACM Press. pages 21, 24, 25, 27, 45

[Kra87] Eugene F. Krause. *Taxicab Geometry: An Adventure in Non-Euclidean Geometry*. Dover Publications, 1987. pages 68

[KW02] Tianbo Kuang and Carey Williamson. A measurement study of realmedia audio/video streaming traffic. In *ITCOM*, pages 68–79, 2002. pages 116

[L7-08] Application layer packet classifier for linux. `http://l7-filter.sourceforge.net/`, June 2008. pages 20, 22

[Lan94] P. Langley. Selection of relevant features in machine learning. In *Proceedings of the AAAI Fall Symposium on Relevance*, 1994. pages 122

[LBS+98] Joseph P. Loyall, David E. Bakken, Richard E. Schantz, John A. Zinky, David A. Karr, Rodrigo Vanegas, and Kenneth R. Anderson. Qos aspect languages and their runtime integration. *Lecture Notes in Computer Science*, 1511:303, 1998. pages 19

[LC03] Connie Logg and Les Cottrell. Characterization of the traffic between slac and the internet. `http://www.slac.standford.edu/comp/net/slac-netflow/html/SLAC-netflow.html`, 2003. pages 21, 25, 48, 77

[LGW04] Alberto Leon-Garcia and Indra Widjaja. *Communication Networks: Fundamental Concepts and Key Architectures*. McGraw-Hill, second edition edition, 2004. pages 10, 17

[Liu05] Huan Liu. Toward integrating feature selection algorithms for classification and clustering. *IEEE Transaction on Knowledge and Data Engineering*, 17(4):491–502, 2005. pages 123

[LK02] E. Leopold and J. Kindermann. Text categorization with support vector machine. how to represent texts in input space. *Machine Learning*, 46:423–444, 2002. pages 122

[LLS00] Tjen S. Lim, Wei Y. Loh, and Yu S. Shih. A comparison of prediction accuracy, complexity, and training time of thirty-three old and new classification algorithms. *Machine Learning*, 40(3):203–228, 2000. pages 56, 57

[LLW02] Huiqing Liu, Jinyan Li, and Limsoon Wong. A comparative study on feature selection and classification methods using gene expression profiles and proteomic patterns. *Genome Informatics*, 13:51–60, 2002. pages 123

[LM98] Huan Liu and Hiroshi Motoda. *Feature Selection for Knowledge Discovery and Data Mining*. Kluwer Academic Publishers, 1998. pages 122, 123

[LS94] P. Langley and S. Sage. Induction of selective bayesian classifiers. In *10th Conference on Uncertainty in Artificial Intelligence*, 1994. pages 126

[Man07] Yann Grossel Raphaël Manfredi. gtk-gnutella. http://gtk-gnutella.sourceforge.net, May 2007. pages 25

[MBB98] Ryszard Michalski, Ivan Bratko, and Avan Bratko. *Machine Learning and Data Mining: Methods and Applications*. Wiley & Sons, 1998. pages 71, 72

[Met08] Metacafe. http://www.metacafe.com/, June 2008. pages 144

[MHS02] Yu-Ben Miao, Wen-Shyang Hwang, and Ce-Kuen Shieh. A transparent deployment method of rsvp-aware applications on unix. *Comput. Networks*, 40(1):45–56, September 2002. pages 19

[Mic88] D. Michie. Machine learning in the next five years. In *3rd European Working Session on Learning*, 1988. pages 71

[Mic07] Microsoft Corp. Windows live messenger. http://messenger.msn.com/, 2007. pages 75, 142

[Min89] J. Mingers. An empirical comparison of selection measures for decision-tree induction. *Machine Learning*, 3(4):319–342, 1989. pages 56

[Mit97] Tom Mitchell. *Machine Learning*. McGraw-Hill, 1997. pages 69, 72

[MKK+01] D. Moore, K. Keys, R. Koga, E. Lagache, and kc Claffy. Coralreef software suite as a tool for system and network administrators. In *15th System Administration Conference*, 2001. pages 21, 25, 48, 77

[ML05] Annie De Montigny-Leboeuf. Flow attributes for use in traffic characterization. Technical Report CRC-TN-2005-03, Communications Research Centre Canada, December 2005. pages 20, 23, 25, 26, 78, 82, 100

[MLJ07] S. McCanne, C. Leres, and V. Jacobson. pcap - packet capture library. http://www.tcpdump.org, June 2007. pages 92

[MLK06] Justin Ma, Kirill Levchenko, and Christian Kreibich. Unexpected means of protocol inference. In *6th ACM SIGCOMM Conference on Internet Measurement*, 2006. pages 22, 124

[MM99] S. Maneewongvatana and D. Mount. Analysis of approximate nearest neighbor searching with clustered point sets. In *Workshop on Algorithm Engineering and Experiments*, 1999. pages 70

[MMS08] Microsoft windows media services. http://www.microsoft.com/windows/windowsmedia/forpros/server/server.aspx, June 2008. pages 144

[Moo01] A. v. Moorsel. Metrics for the internet age: Quality of experience and quality of business. In *Fifth International Workshop on Performability Modeling of Computer and Communication Systems*, 2001. pages 17

[MP69] Marvin Minsky and Seymour Papert. *Perceptrons*. MIT Press, 1969. pages 71

[MP05] Andrew W. Moore and Konstantina Papagiannaki. Toward the accurate identification of network applications. *Lecture Notes in Computer Science*, 3431:41–54, 2005. pages 21, 25, 26, 27

[Mur98] Sreerama K. Murthy. Automatic construction of decision trees from data: A multi-disciplinary survey. *Data Mining and Knowledge Discovery*, 2:345–389, 1998. pages 56

[MW06] A. Madhukar and C. Williamson. A longitudinal study of p2p traffic classification. In *Conference on Measurement and Simulation of Computer and Telecommunication Systems (MASCOTS)*, September 2006. pages 21

[MZ05a] Andrew W. Moore and Denis Zuev. Discriminators for use in flow-based classification. Technical report, Intel Research, Cambridge, 2005. pages 45

[MZ05b] Andrew W. Moore and Denis Zuev. Internet traffic classification using bayesian analysis techniques. In *Proceedings of the 2005 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, pages 50–60, New York, NY, USA, June 2005. ACM Press. pages 20, 23, 25, 26, 27, 45, 67, 100

[NA06] T. Nguyen and G. Armitage. Training on multiple sub-flows to optimise the use of machine learning classifiers in real-world ip networks. In *31st IEEE Conference on Local Computer Networks*, 2006. pages 25, 26, 116

[NBBB98] K. Nichols, S. Blake, F. Baker, and D. Black. *RFC 2474: Definition of the Differentiated Services Field (DS Field) inthe IPv4 and IPv6 Headers*. IETF, 1998. pages 18

[Nil65] N. J. Nilsson. *Learning Machines*. McGraw-Hill, 1965. pages 64

[NMTM00] K. Nigam, A. K. Mccallum, S. Thrun, and T. Mitchell. Text classification from labeled and unlabeled documents using em. *Machine Learning*, 39:103–134, 2000. pages 122

[NSS05] Nnamdi Nwanze, Douglas H. Summerville, and Victor A. Skormin. Real-time identification of anomalous packet payloads for network intrusion detection. In *IEEE Workshop on Information Assurance and Security United States Military Academy*, 2005. pages 22

[NWG02] Klara Nahrstedt, Duangdao Wichadakul, and Xiaohui Gu. A programming framework for quality-aware ubiquitous multimedia applications. In *ACM Multimedia*, 2002. pages 19

[Oik93] J. Oikarinen. *RFC 1459: Internet Relay Chat Protocol*. IETF, 1993. pages 76

[PF95] V. Paxson and S. Floyd. Wide-area traffic: The failure of poisson modeling. *IEEE/ACM Transactions on Networking*, 3(3):226–244, 1995. pages 23

[PFK98] Foster Provost, Tom Fawcett, and Ron Kohavi. The case against accuracy estimation for comparing induction algorithms. In *In Proceedings of the Fifteenth International Conference on Machine Learning*, pages 445–453. Morgan Kaufmann, 1998. pages 72

[Pha05] Phong H. Pham. Traffic classification with passive measurement. Master's thesis, Department of Informatics, University of Oslo, 2005. pages 98, 141

[PM00] Dan Pelleg and Andrew Moore. X-means: Extending k-means with efficient estimation of the number of clusters. In *17th International Conference on Machine Learning*, pages 727–734, 2000. pages 70

[PN97] P. A. Porras and P. G. Neumann. Emerald: Event monitoring enabling responses to anomalous live disturbances. In *20th National Information Systems Security Conference*, pages 353–365, 1997. pages 20, 21, 25, 48, 77

[Pol06] Ramiro Polla. Reverse-engineering msn messenger's video conversation formats. `http://ml20rc.msnfanatic.com/vc_1_1/`, September 2006. pages 98, 141, 142

[Pos80] Jon Postel. *RFC 768: User Datagram Protocol*. IETF, August 1980. pages 13, 31, 34

[Pos81a] Jon Postel. *RFC 791: Internet Protocol*. IETF, September 1981. pages 14, 31, 34

[Pos81b] Jon Postel. *RFC 793: Transmission Control Protocol*, 1981. pages 13, 31, 34

[PR83] J. Postel and J. Reynolds. *RFC 854: Telnet Protocol Specification*. IETF, 1983. pages 16, 25

[PR85] J. Postel and J. Reynolds. *RFC 959: File Transfer Protocol*. IETF, 1985. pages 11, 25

[Pro08] Protocolinfo. `http://protocolinfo.org`, June 2008. pages 98, 141, 144

[Qui86] Ross Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986. pages 56, 58

[Qui90] Ross Quinlan. Learning logical definitions from relations. *Machine Learning*, 5:239–266, 1990. pages 61

[Qui93] Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers, 1993. pages 43, 56, 57, 58, 59

[Qui94] J. R. Quinlan. Minimum description length principle and categorical theories. In *Proceedings of the Eleventh International Conference on Machine Learning*, pages 464–470. Morgan Kaufmann, 1994. pages 63

[Qui95] J. R. Quinlan. Mdl and categorical theories (continued). In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 464–470. Morgan Kaufmann, 1995. pages 63

[Qui08] Apple quicktime. `http://www.apple.com/quicktime/`, June 2008. pages 144

[Rad08] RadioToolbox. The shoutcast streaming standard. `http://forums.radiotoolbox.com/viewtopic.php?t=74`, June 2008. pages 144

[Räi03] Vilho Räisänen. *Service Quality in IP Networks*. John Wiley & Sons Ltd., 2003. pages 1, 2, 16, 17

[RB00] T. Roscoe and G. Bowen. Script-driven packet marking for quality of service support in legacy applications. In *SPIE Conference on Multimedia Computing and Networking*, 2000. pages 19

[Rea08] Realnetworks. `http://www.real.com/`, June 2008. pages 76, 116, 144

[RG07] M. Roesch and C. Green. *Snort - The Open Source Network Intrusion Detection System*, 2007. pages 20, 22, 25, 98, 141

[RHC99] Y. Rui, T. S. Huang, and S. Chang. Image retrieval: Current techniques, promising directions, and open issues. *Visual Communication and Image Representation*, 10:39–62, 1999. pages 122

[Ric09] Tristan Richardson. The rfb protocol. `http://www.realvnc.com/docs/rfbproto.pdf`, October 2009. pages 143

[RJM06] Shai Rubin, Somesh Jha, and Barton P. Miller. Protomatching network traffic for high throughput network intrusion detection. In *The 13th ACM Conference on Computer and Communications Security*, 2006. pages 22

[RR95] Kay A. Robbins and Steven Robbins. *Practical UNIX Programming*. Prentice Hall PTR, 1st edition, November 1995. pages 35

[RSFWH98] Tristan Richardson, Quentin Stafford-Fraser, Kenneth R. Wood, and Andy Hopper. Virtual network computing. *IEEE Internet Computing*, 2:33–38, 1998. pages 22, 76

[RSSD04] Matthew Roughan, Subhabrata Sen, Oliver Spatscheck, and Nick Duffield. Class-of-service mapping for qos: a statistical signature-based approach to ip traffic classification. In *IMC '04: Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement*, pages 135–148, New York, NY, USA, 2004. ACM Press. pages 20, 21, 23, 25, 26, 30, 70, 71, 78, 79, 116

[Rul07] RuleQuest Research. Data mining tools see5 and c5.0. `http://www.rulequest.com/see5-comparison.html`, 2007. pages 23, 60

[SA93] L. Shastri and V. Ajjanagadde. From associations to systematic reasoning: A connectionist representation of rules, variables and dynamic bindings using temporal synchrony. *Behavioural and Brain Sciences*, 16(3):417–494, 1993. pages 71

[SC03] H. Schulzrinne and S. Casner. *RFC 3551: RTP Profile for Audio and Video Conferences with Minimal Control*. IETF, 2003. pages 22

[SCFJ03] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. *RFC 3550: RTP: A Transport Protocol for Real-Time Applications*. IETF, July 2003. pages 10, 75, 142

[SD90] Jude W. Shavlik and Thomas G. Dietterich, editors. *Readings in Machine Learning (The Morgan Kaufmann Series in Machine Learning)*. Morgan Kaufmann, June 1990. pages 41

[SG05] Ramesh Subramanian and Brian D. Goodman, editors. *Peer to Peer Computing: The Evolution of a Disruptive Technology*. IGI Global, 2005. pages 146

[Sha48] C. E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379Ãś423 and 623Ãś656, 1948. pages 57

[SHO08] Shoutcast. `http://www.shoutcast.com/`, June 2008. pages 144

[SIYS07] D. Singh, A. M. Ibrahim, T. Yohanna, and J. N. Singh. An overview of the applications of multisets. *Novi Sad Journal of Mathematics*, 37(2):73–92, 2007. pages 42

[SLSH04] Chi-Huang Shih, Chung-Chih Liao, Ce-Kuen Shieh, and Wen-Shyang Huang. A transparent qos mechanism to support intserv/diffserv networks. In *First IEEE Consumer Communications and Networking Conference*, pages 233–238, 2004. pages 20

[Smo87] P. Smolensky. On variable binding and the representation of symbolic structures in connectionist systems. Technical Report CU-CS-355-87, Department of Computer Science & Institute of Cognitive Science, University of Colorado, 1987. pages 71

[SOMS08] Gèza Szabò, Dàniel Orincsay, Szabolcs Malomsoky, and Istvàn Szabò. On the validation of traffic classification algorithms. *Lecture Notes in Computer Science*, 4979:72–81, 2008. pages 3, 31

[SRL98] H. Schulzrinne, A. Rao, and R. Lanphier. *RFC 2326: Real Time Streaming Protocol (RTSP)*. IETF, April 1998. pages 10, 144

[SS98] W. Siedlecki and J. Sklansky. On automatic feature selection. *International Journal of Pattern Recognition and Artificial Intelligence*, 2:197–220, 1998. pages 122

[SSW04] S. Sen, O. Spatscheck, and D. Wang. Accurate, scalable in-network identification of p2p traffic using application signatures. In *13th International World Wide Web Conference (WWW)*, 2004. pages 20, 21, 22, 25, 26

[Sto77] M. Stone. Asymptotics for and against cross-validation. *Biometrika*, 64:29–35, 1977. pages 73

[SW95] D. L. Swets and J. J. Weng. Efficient content-based image retrieval using automatic feature selection. In *IEEE International Symposium on Computer Vision*, 1995. pages 122

[SW03] M. Siller and J. Woods. Improving quality of experience for multimedia services by qos arbitration on a qoe framework. In *IEEE Packet Video*, 2003. pages 17

[Tan03] Andrew Tanenbaum. *Computer Networks*. Pearson Education, Inc., 4th edition, 2003. pages 10, 14, 23

[TAO07] Masaki Tai, Shingo Ata, and Ikuo Oka. Fast, accurate, and lightweight real-time traffic identification method based on flow statistics. *Lecture Notes in Computer Science*, 4427:255–259, 2007. pages 20, 23, 25, 26, 30, 78, 82

[Tel04] TeleManagement Forum. *SLA Management Handbook*, 2004. Public Evaluation Version 2.0. pages 16

[Thi07] S. Thiemjarus. *A Framework for Contextual Data Fusion in Body Sensor Networks*. PhD thesis, Imperial College, London, 2007. pages 122

[TK03] Sergios Theodoridis and Konstantinos Koutroumbas. *Pattern Recognition*. Elsevier, 2003. pages 72

[TLY04] S. Thiemjarus, B. P. L. Lo, and G-Z Yang. Feature selection for wireless sensor networks. In *First International Workshop on Wearable and Implantable Body Sensor Networks*, 2004. pages 122

[TMV01] C. Tsetsekas, S. Maniatis, and I. S. Venieris. Supporting qos for legacy applications. In *ICN '01: Proceedings of the First International Conference on Networking-Part 2*, pages 108+, 2001. pages 19

[TS93] G. G. Towell and J. W. Shavlik. Extracting refined rules from knowledge-based neural networks. *Machine Learning*, 13:71–101, 1993. pages 71

[Tys07] Jeff Tyson. How the old napster worked. `http://computer.howstuffworks.com/napster.htm`, May 2007. pages 25

[Val07] Valve Corporation. Half-life. `http://www.half-life.com`, June 2007. pages 75

[Wer75] Paul Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. PhD thesis, Havard University, 1975. pages 71

[WF05] Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, second edition, 2005. pages 41, 43, 57, 64, 69, 71, 72, 91

[WNGX01] Duangdao Wichadakul, Klara Nahrstedt, Xiaohui Gu, and Dongyan Xu. 2kq+: An integrated approach of qos compilation and component-based, run-time middleware for the unified qos management framework. In *IFIP/ACM International Conference on Distributed Systems Platforms*, November 2001. pages 19

[WZA06] Nigel Williams, Sebastian Zander, and Grenville Armitage. A preliminary performance comparison of five machine learning algorithms for practical ip traffic flow classification. *ACM SIGCOMM Computer Communication Review*, 36(5), October 2006. pages 20, 23, 25, 26, 33, 60, 67, 79, 100, 123, 124

[X.995] ITU-Recommendation. *X.902: Information technology - Open Distributed Processing - Reference Model: Foundations*, November 1995. pages 17

[XJK01] E. Xing, M. Jordan, and R. Karp. Feature selection for high-dimensional genomic microarray data. In *15th International Conference on Machine Learning*, 2001. pages 122

[XZB05] Kuai Xu, Zhi-Li Zhang, and Supratik Bhattacharyya. Profiling internet backbone traffic: behavior models and applications. In *SIGCOMM '05: Proceedings of the 2005 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 169–180, New York, NY, USA, October 2005. ACM Press. pages 24, 25, 46

[Yah08] Yahoo coder's cookbook. `http://www.ycoderscookbook.com/`, June 2008. pages 76, 98, 141

[YL04] L. Yu and H. Liu. Redundancy based feature selection for microarray data. In *10th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2004. pages 122

[Ylo06] T. Ylonen. *RFC 4251: The Secure Shell (SSH) Protocol Architecture*. IETF, 2006. pages 25, 76

[You08] Youtube. `http://www.youtube.com`, June 2008. pages 76, 144

[YP97] Y. Yang and J. O. Pederson. A comparative study on feature selection in text categorization. In *14th International Conference on Machine Learning*, 1997. pages 122

[ZBHJ97] L. Zhang, S. Berson, S. Herzog, and S. Jamin. *RFC 2205: Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specification*. IETF, 1997. pages 17

[ZNA05a] Sebastian Zander, Thuy Nguyen, and Grenville Armitage. Automated traffic classification and application identification using machine learning. In *IEEE Conference on Local Computer Networks 30th Anniversary (LCN'05)*, 2005. pages 23, 25, 71, 79

[ZNA05b] Sebastian Zander, Thuy Nguyen, and Grenville J. Armitage. Self-learning ip traffic classification based on statistical flow characteristics. *Lecture Notes in Computer Science*, 3431:325–328, April 2005. pages 20, 23, 25, 26, 71, 79, 100

[ZP00] Yin Zhang and Vern Paxson. Detecting backdoors. In *Proc. 9th USENIX Security Symposium*, pages 157–170, aug 2000. pages 20, 21, 22, 25, 26, 32, 77, 78, 82, 98, 141