UNIVERSITY OF CALIFORNIA
RIVERSIDE

Challenges in Security and Traffic Management in Enterprise Networks

A Dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy

in

Computer Science

by

Dhiman Barman

December 2008

The Dissertation of Dhiman Barman is approved:

____

____

____

Committee Chairperson

University of California, Riverside

Acknowledgements

I extend my gratefulness to my advisor Michalis Faloutsos without whose guidance I might not have made it this far. His academic, moral and financial support helped me work at my own pace and overcome many challenges. The most important thing that I learned from him was to focus on one problem at a time and think through deeply and inquisitively. His ability to question any idea and pay attention to details helped me improve my thought processing. His teaching skills helped me improve my presentation skills. His approach to treat students like peers makes him very approachable. He taught me to address a problem top down, ask important questions, come up with systematic solutions, and face both technical and non-technical challenges strategically. I will continue to benefit from his teachings and guidance throughput my life.

I owe my gratitute to Mart Molle, my thesis committee member. Mart offered guidance at different steps during my graduate studies. He offered me both technical and non-technical suggestions at times. His door was always open to whenever I faced problems. His insightful comments during presentation, thesis writing or otherwise helped me broaden by view and improve my work. I owe gratitude to Eamonn Keogh for agreeing to serve on my thesis committee. I learned a lot on presentation skills from him and also from his data-mining course. I owe gratitute to Nina Taft under whose mentorship, I learned to think clearly and big. Her emphasis on writing and expressiveness helped my writing skills and be consistent. I also express my thankfulless to Dimitrios Gunopulos and Neal Young with whom I collaborated on Hierarchical Change Detection problem. Their expertise on algorithms and insightful comments helped me improve my skills. My gratitude extends to all my mentor and co-authors namely Divesh Srivastava, Eitan Altman, Deepak Agarwal, Flip Korn, Kc Claffy etc who taught me in every step how to solve problems. I am also thankful to Hyunchul Kim with who I have been collaborating on the application traffic classification problem and without his contributons I could not have completed the work on traffic classification.

My graduate studies would not have been funfilled had I not been in the company of my friends, Anirban Banerjee, Nicholas Valler, Smruti Pariccha, Ece Gelal, Marios Illiofotou, Satya Mohanti, Sandeep Gupta, Yordanos Beyene and many more. Anirban always inspired me through his focus and efficiency in work. I learned from the Marios's presentation and thinking skills. Nicholas helped me in improving my writing skills. Smruti helped me in reviewing many papers and discussing and bouncing many ideas.

Lastly, my gratitute goes to my parents, siblings and wife. My parents and siblings were patient enough for me to finish the studies inspite of numerous challenges in our family which I should have attended to. Thanks to my lovely wife, Sankalpita for bearing with me when I used to steal time from her to finish work and supporting me with continued love and motivation at every step.

ABSTRACT OF THE DISSERTATION

Challenges in Security and Traffic Management in Enterprise Networks

by

Dhiman Barman

Doctor of Philosophy, Graduate Program in Computer Science
University of California, Riverside, December 2008
Dr. Michalis Faloutsos, Chairperson

Management of enterprise networks is a challenging problem because of their continued growth in size and functionality. In this thesis, we propose and evaluate a framework, *Godai*, which addresses the challenges in (i) setthing thresholds in end host anomaly detectors, (ii) hierarchical summarization in data and (iii) application traffic classification. *Godai* enables IT operators to identify the end hosts that have been enslaved by an attacker to launch attacks and *Godai* achieves it by diversifying anomaly detector configuration. The general policies in *Godai* framework are holistic and achieve two goals: (a) balance the trade-offs between false alarm and mis-detection rates and (b) show that the benefits of full diversity can be attained at reduced complexity, by clustering the end hosts and treating a cluster homogeneously.

The underlying principle of attack detection is to identify the traffic samples that change significantly from normal traffic. *Godai* generalizes the concept for data with hierarchical identifiers, e.g., IP prefixes, URLs. The main motivation of using a parsimonious hierarchical summarization of the measure attributes (e.g., total bytes or website hits) is that it eases the burden on IT operators to interprete analysis reports. *Godai* proposes efficient and provable algorithms to produce parsimonious explanations from the output of any statistical model that provides predictions and confidence intervals, making it widely applicable.

Finally, *Godai* takes a step towards associating applications to traffic flows and enable the operators to understand the profile of the end hosts. *Godai* critically re-visits the existing ad hoc techniques of traffic classification approaches based on *transport layer ports* [83], *host behavior* [68], and *flow features* [105] and analyzes the effectiveness of different approaches. The results allow us to answer questions about the best available traffic classification approach, the conditions under which it performs well, and the strengths and limitations of each approach. The multifarious functionalities allow *Godai* to be a viable solution in enterprise network management.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Management of enterprise networks is a complex task. Network management entails configuration, provisioning, monitoring, testing of different network entities and post-processing of logs. This thesis is concerned with three themes: *configuration*, *change detection in traffic* and *traffic classification*. Regarding *configuration*, our focus will be on configuring the end host anomaly detectors. The underlying principle of attack detection is to detect deviation in traffic profile. The component on *change detection in traffic* extends the idea of change detection for hierarchical data. Finally, we emphasize on identifying applications (e.g. HTTP, P2P) to know the end host traffic profile.

One of the important goals of the IT operators is to keep the enterprise networks secure through proper configuration of the network entities such as end hosts, servers, routers, links etc. In order to do the network configuration, the IT operators use a variety of tools both from the commercial and public domain [7, 6, 9, 1]. Such tools help the IT operators to certain extent but despite such efforts, the IT operators find it increasingly difficult to manage the enterprise networks. The online report [70] stated that 80% of the IT budget in enterprise is devoted to maintain just the status quo, but configuration error still accounts for 62% of network downtime. The IT operators manage a large number of entities and often carry out unplanned configuration tasks. With rising security threats to network resources, the IT operators need to manage all the end hosts in an enterprise while meeting enterprise wide policies.

1.1 Configuration Management

In this thesis, our focus is on configuring the intrusion detection systems deployed at the end hosts, so that the IT operators can effectively and efficiently detect if the end hosts have been enslaved for a botnet by external attackers. Botnets are a menacing threat in the Internet today [8, 12, 17, 55]. A botnet is a collection of compromised hosts under a commmon *command and control* botmaster. The aim of the botmaster is to recruit unsuspecting end hosts to form an array of hosts and then use their collective power to launch attacks and commit other illegal activities [86]. The majority of botnets today engage in spam generation and Distributed Denial of Service(DDoS) activities. Detection and mitigation of botnet-based attacks is a challenging task [50, 53, 52, 48, 65, 74, 49, 46]. According to recent estimates, the number of compromised hosts in a botnet range from 6 million [18] to 150 million [21]. In addition, the number of vulnerabilities in the operating systems keeps increasing which enables the botmaster to find new recruits. The collective power of the numerous recruited hosts in an enterprise could be tremendous. In May 2006, DDoS against Blue Security, an anti-spam company was forced to shut down its services. In April 2007, a sustained attack on the goverment and business websites in Estonia also brought the country on its knees.

This thesis proposes a design of a framework that enables the enterprise IT operators detect when its end-hosts participate in malicious activities such as DDoS. In modern enterprise networks today, each employee is typically given a laptop that is configured by the enterprise IT department. The IT departments employ a standard build for everyone in order to simplify the management of a large number of machines. One of the software applications usually installed on such laptops is a Host Intrusion Detection System (HIDS) and/or a personal firewall [95, 90]. HIDS are used to control a variety of security attacks. In addition to signature detection (that protects against known worms and viruses), many HIDS systems also have a set of features (system parameters) they track on ongoing monitoring network traffic. The effectiveness of these HIDS depends on how these HIDS are individually configured (e.g., how their threshold values are set) as these HIDS raise alerts and send to the IT operators. This thesis takes a first look at the issues on the implications of different ways of configuring the HIDS.

When the configuration of the end-hosts is a concern, usually the IT operators go for simplicity and set the same value for the triggering threshold for any particular feature for each user ("homogeneous" policy). In this thesis, we focus on the problem of setting threshold values in the end-host anomaly detectors across a *group* of employees when the choice is guided by a global enterprise “policy”. The HIDS features we examine in this thesis are those that could reveal botnet behaviors such as DDoS, spam-related behaviors and scanning activities. Such features (e.g., as used on Bro [90]) include the number of TCP connections, UDP connections, distinct connections, HTTP connections, SMTP connections etc.

In order to understand the impact of the homogeneous policy across a population of employees, we study logs (i.e., tcpdump) from 350 employee laptops from a large enterprise. The dataset is unique because the data is collected directly on the laptops themselves, thus allowing us to capture all of the traffic generated by a user, regardless of which network he or she may be attached to. Using Bro [90], we process the logs and obtain the flow records and their associated features (e.g., protocol, start time, end time etc). We uncover that there is a great deal of user diversity in the “tail” (loosely speaking) of the distribution of HIDS’s features. By this we mean that the boundary between “normal” and “atypical” values of various features can differ by 3 to 4 orders of magnitude across our users. (We call this “fringe diversity”.) The *first* consequence of enforcing a homogeneous policy upon a set of users that are inherently diverse (in terms of what matters for anomaly detection), is that the individual false positive and false negative rates differ dramatically across users. This is an unintended ramification of a policy that focusses on ease of management. The *second* consequence is that the employees’ behaviors affect each others’ performance. For example, when a single threshold value is computed based upon looking at data from all employees simultaneously, then clearly a small number of heavy users will bias the computation, leading to thresholds that are detrimental to light users. There is a need to design a general approach which can address the above two consequences.

1.1.1 Diversity in Configuration

Our goal is to address the challenges arising due to homogeneity in configuration, control management complexity as well as balance between false alarm and mis-detection rates. Our proposed policies contain two components: First, a parameter that explicitly states the relative importance of false alarms and mis-detection rates so as to gain better control over this balance. Second, a grouping component defines whether the users are treated as a one single homogeneous group, a fully-diverse group (each user belongs to this own group), or whether users are clustered into a small set of behavior groups. The grouping component essentially states the level of granularity of diversity with which an IT department views its employees; none, full or partial diversity. We evaluate these policies, and in doing so, we raise an awareness of a variety of issues and trade-offs that arise to affect the outcome of such policies.

In a full diversity policy, each user gets a personalized configuration of its trigger threshold. We quantify the benefits of a diversity policy which includes a better balance between false alarms and mis-detection rates for a vast majority of the users. In addition, we illustrate that even in the presense of attackers with full knowledge of the users' behaviors, a diversity policy is more effective in weakening DDoS attacks than a homogeneous configuration policy. Next we ask two questions:

1. How broad are the conditions under which the benefits of the diversity arise ?

2. Is it enough for there to be a great deal of diversity in user fringe behavior ?

We use simple utility functions to capture the trade-offs between false alarm and mis-detection rates, and uncover that the choice of utility function has a strong impact on whether or not a diversity policy actually generates substantial benefits. Most enterprise IT departments have a hard time articulating what they consider more important, the reduction of false alarms or mis-detection rates. This classic trade-off remains elusive to quantify in practice. However we show that this choice is critical for the selection of a policy when setting thresholds for a group of employees. A diversity policy is most beneficial when reducing false alarm rate is considerably more important than reducing mis-detection rate.

As mentioned above, another pitfall of applying a homogeneous policy to a set of diverse users, is that the behavior of some employees can end up influencing the configuration of other employees. Trading off the performance of some employees at the expense of others is an issue that should be brought to bear much more explicitly. The choice between a homogeneous policy and a full diversity one thus reveals a key trade-off: the homogeneous policy achieves simple management but the employees end up influencing each others performance, whereas a diversity policy breaks the dependency across employees but may lead to more complex management.

Having understood the benefits of diversity, we then conducted a survey of a dozen IT personnel across 5 organizations (4 corporations and 1 university). All but one of them were resistant to the idea of configuring the end-hosts with different threshold values at which alarms should trigger. The personnel in our survey explained their resistance originates from a lack of: (a) human resouces to individually configure the end hosts and (b) a lack of understanding as to how to interpret alarms coming from different machines if they fire at different values. We believe that these two concerns can be easily overcome. First, HIDS can clearly be configured by automated algorithms. Moreover, when systems today are evaluated in greater depth after alarms fire, other data is collected to give information about activities on the machines at the time the alarm fired. This practice should facilitate alarm interpretation. However, despite these seemingly simple fixes to the root causes of their resistance, the adamant sentiment about simplicity of management means that this industry practice is unlikely to deviate in any large way from current practice.

Can we reconcile the benefits of diversity with the complexity of enterprise network security management? We propose an intermediate solution between homogeneity and full diversity, namely *partial diversity* in which employees are classified into a small set of groups and within each group, all laptops are configured the same way. We design an algorithm for computing thresholds under this policy. We demonstrate that with a small number of groups, the partial diversity policy can achieve most of the benefits of a full diversity policy.

In our conversations with our IT staff, they admitted that they were much more amenable to the idea of working with a small set of configurations that could be distributed amongst the employees. Therefore, this thesis addresses the methodology for threshold setting for HIDS features in a new light, namely, in the context of enterprises that must develop configuration policies that lead to an appropriate set of group-dependent thresholds. The thesis reveals the issues and trade-offs that surface due to user diversity.

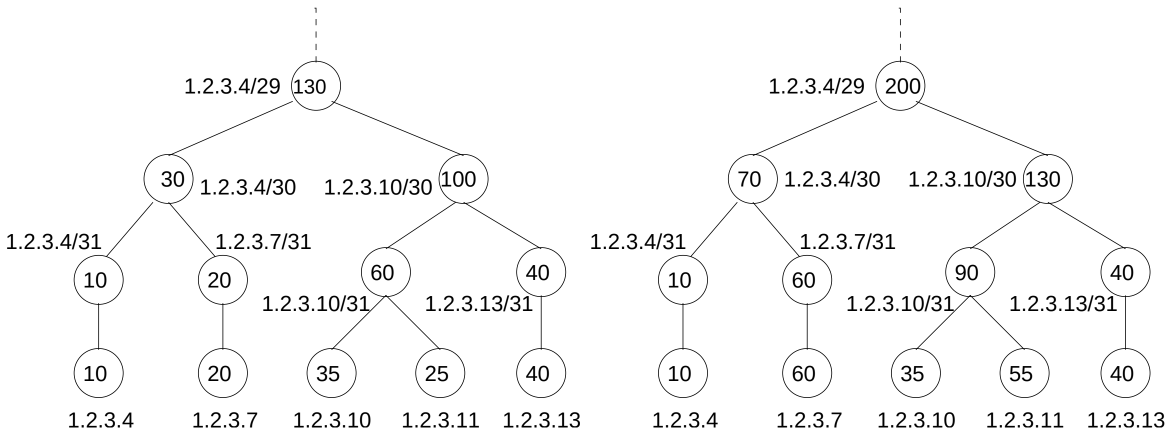
1.2 Hierarchical Change Detection

Currently, the enterprise IT operators monitor the traffic going through a network and analyze it to understand the underlying communication trends and usage [83, 104, 103, 36]. The underlying principle of attack detection is to detect changes in traffic profile which is significantly different from normal traffic. With this motivation, we generalize the concept of change detection in the context of data where the identifiers are hierarchical in nature. Data with hierarchical identifiers occurs in different contexts such as URL, IP prefix, Geographical locations, time etc. Our goal is to produce traffic explanations (i.e., reports) that match the underlying trends and changes. For example, instead of reporting heavy hitters, or hundreds of smalls flows or the amount of traffic to TCP port 80, our method reports that most change in traffic count has happened for certain group of clients or web servers. Dimension attributes in IP traffic are typically hierarchical, and a variety of applications call for summarizing the measure attributes along the hierarchies of these attributes. For example, the total volume of traffic (in bytes or packets) can be summarized hierarchically by source IP or destination IP prefixes (say of length 8 to 32), by time (e.g., year/month/day/hour), or by port number (e.g., port number 80) or the set of all possible values(*). Well-known services are allocated port numbers below 1024 and ephemeral port number above 1023 are allocated to new sessions on-demand, we can define hierarchy on the set of high ports (> 1023) port numbers and the set of low ports (< 1024) numbers.

Existing tools like FlowScan [4], Cisco's FlowAnalyzer are used by the IT managers to construct a model of application usage by classifying traffic according to the IP header fields. Analysis using such tools could reveal that 90% traffic is due to TCP and 75% of TCP traffic is due to HTTP. In the context of datamining, existing tools [98, 100] help to summarize and navigate the data at different levels of aggregation (e.g., total traffic in each prefix during July 2008) via *drill-down* and *roll-up* operators. These tools are also used to characterize changes in the hierarchical summaries over time (e.g., the traffic in July 2008 compared to their expectations over different prefixes), to detect anomalies and characterize trends. When it comes to the summarizing the changes, there are conflicting issues of readability, compactness of the reports, robustness and theoretical optimality. The current summarization tools [83] have limitations that they report excessive details in explaining traffic trends and usage – such as TCP traffic contributes 90% flows or 153.3.0.0/16 results in 30% of traffic etc. However, when the data items or identifiers is large or the number of identifiers at which changes happen is large, IT operators will need compact explanations. By "change" we mean when the total measure values differ significantly from the expected values. Thus, the traffic analysis involves two conflicting issues: the verbosity of the reports and human readability. More explanations can provide better understanding but can make the reports harder to read and interprete. One can make the explanations verbose by providing a separete ad hoc explanation for the observed change at each prefix. Explanations can also be *parsimonious* (e.g., a single explanation for multiple observed changes such as attributing a drop in traffic at a large number of prefixes to network outage). Parsimonious explanations are obviously more desirable and more effective than (ad hoc) verbose explanations. Thus, the main challenge is how to summarize the significant trends and report the significant changes.

1.2.1 Effective Explanations of Change

In this thesis, we are interested in *parsimonious explanations* of changes in measure attributes (e.g., total bytes or website hits) aggregated along an associated dimension attribute hierarchy (e.g., IP prefix or URL). Existing work has addressed the issue of explaining change between OLAP (Online Analytical Processing) aggregates in terms of subaggregates [99] but these changes are expressed as outliers of point-to-point subaggregate comparisons. We seek a more holistic explanation. We propose a natural model that makes effective use of the dimension hierarchy and describe changes at the leaf nodes of the hierarchy (e.g., individual IP addresses) as a composition of "node weights" along each node's root-to-leaf path in the dimension hierarchy – each node weight constitutes an explanatory term. For example, *overall traffic volume in IP prefix /24 increased by a factor of three*. Figure 1.1 shows an ex-



(a) Long-term summary (b) Short-term summary

Figure 1.1: Measure counts in IP prefix hierarchy at two different snapshots

ample of IP prefix hierarchy where the number inside the circles represent the traffic counts.

The leaves of the tree represent prefix /32 and the root is represented by (*). The child node counts are aggregated and assigned to the parent. Formally, we assume that the dimension hierarchy remains fixed over time, and each data item (e.g., a flow identified with an IP) has a timestamp and is associated with a leaf node (e.g., an individual IP address) of the hierarchy. A hierarchical summary or snapshot (over some time interval) then associates with each node in the dimension hierarchy the aggregated value of the measure attribute (e.g., total traffic in bytes or packets) of all the data items (with a timestamp in that time interval) in its subtree. Figure 1.1(a)-(b) correspond to different snapshots over different intervals of time.

If we consider two snapshots, it is clear that the changes between the trees can be expressed over the different levels of the dimension hierarchy in numerous possible ways. For example, the traffic volume at the prefix 1.2.3.10/31 increases $\frac{3}{2}$ fold, we can model this change (among other possibilities) as (i) a weight of $\frac{3}{2}$ for IP addresses, 1.2.3.10 and 1.2.3.11, or (ii) a weight of 3 at IP prefix 1.2.3.10/31 and a weight of $\frac{1}{2}$ at the IP addresses, 1.2.3.10 and 1.2.3.11. The important question is, what are the nodes in the hierarchy that explain the (most significant) changes parsimoniously.

Solution Approaches

Here we discuss different approaches to assign weights to the nodes in a hierarchy. A straightforward and intuitive attempt at identification of parsimonious explanations is a *top-down approach*. Starting from the roots of the two snapshots, compare aggregate values of the measure attributes at corresponding nodes. If the difference between the aggregates is completely "explained" by the composition of node weights along the path from the root to the parent of that node, no additional node weight (or explanatory term) is needed at that node. Otherwise, the node weight is set appropriately to the differential value with respect to the composition of weights along nodes for ancestor path from the root to that node. While straightforward and intuitive, such an explanation can be easily shown to not be optimally parsimonious.

For example, the IP prefixes 1.2.3.10/31 and 1.2.3.13/31 used to have the same traffic count but now the IP prefix 1.2.3.13/31 exhibits double traffic count while the other IP prefix exhibits no change. A top-down explanation would attribute a 1.5-fold (3/2) increase at the IP prefix 1.2.3.10/30, and then would have to have to additional explanations at each IP address 1.2.3.10/31 and 1.2.3.13/31 to explain the difference with 1.2.3.10/30 level explanation thus needing 3 explanatory terms. An optimally parsimonious explanation, on the other hand, needs only 1 explanatory term - a 3/2-fold increase at the anomalous prefix 1.2.3.13/31. This explanation is parsimonious in the sense that changes are aggregated with maximal generalization along the dimension hierarchy.

We envision that in many practical cases the IT operators want to compare a hierarchical snapshot with another snapshot whose values are output of a statistical model. Such an operation would be particularly useful, for example, when validating a forecasting model, or to identify conditions that are not properly modeled or to provide parsimonious explanation of anomalies that are expected to be related through the hierarchical structure. In such scenarios, the use of statistical modeling would provide an expected value for each leaf of the hierarchy, with associated confidence intervals. Our proposed method can provide parsimonious explanation after incorporating uncertainty in the forecasts, quantified through confidence intervals. Our framework, *Godai* makes the following contributions towards change detection:

- We formalize the notion of parsimonious explanation of change when comparing two hierarchical summaries, or when we compare a snapshot with the results of a forecasting model. To account for confidence intervals provided by a forecasting model, and to deal with noise, our model allows for a maximum tolerance between the observed change and the root-to-leaf explanation.

- We prove that optimally parsimonious explanations of our problem can be computed efficiently in polynomial time, proportional to the product of the number of leaves and the depth of the dimension hierarchy.

- To complement our conceptual and algorithmic contributions, we conduct a statistically sound experimental evaluation to understand the effectiveness and efficiency of our approach on real hierarchical datasets. We use a predictive model based on an exponentially weighted moving average (EWMA), which is widely used in time series applications. Our experiments demonstrate the effectiveness and robustness of our proposed approach for explaining significant changes, and show that it is more efficient than the worst-case bounds in practice.

1.3 Application Traffic Classification

In order to understand the application footprint that the end hosts generate, *Godai* proposes to critically re-visit application traffic approaches. Traffic classification can help the operators to study the breakdown of traffic volume (e.g., per protocol, port, subnet) as well as to identify new traffic patterns such as network worms [84] or peer-to-peer applications (e.g., BitTorrent, Skype). The goal of *Godai* is to analyze the raw traces which contain IP headers or TCP headers and discover the underlying applications which generate the traffic. The knowledge of what application traffic is flowing through enterprise network can empower the IT operators to provision the network entities and services. For example, the operators can limit the bandwidth allocated to the peer-to-peer applications or access to YouTube site during peak hours. Political, economic, and legal struggles over appropriate use and pricing of the Internet have brought the issue of traffic classification to mainstream media. Three of the most important and acrimonious tussles are: (a) the file sharing tussle, between the file sharing community and intellectual property representatives RIAA (Recording Industry Association of America) and MPAA (Motion Picture Association of America); (b) the battle between malicious hackers, e.g. worm creators, and security management companies; and (c) the network neutrality debate, between ISPs and content/service providers.

In all cases the algorithmic playing field is traffic classification: stopping or deprioritizing traffic of a certain type, versus obfuscating a traffic profile to avoid being thus classified. Traffic classification is also relevant to the more mundane but no less important task of optimizing current network operations and planning improvements in future network architectures. Earlier the IT operators could rely on the use of transport layer port numbers, typically registered with IANA [63] to represent a well-known application. Figure 1.2 shows the TCP header [13] and UDP header [14] and the bit fields associated with the source and destination ports. More recently, increasingly popular applications such as those that support peer-to-peer (P2P) file sharing, hide their identity by assigning ports dynamically and/or using well-known ports of other applications, rendering port-based classification less reliable [67, 81, 102]. A more reliable approach adopted by commercial tools [3, 11, 10] inspects the packet payloads for specific string patterns of known applications [31, 62, 68, 81, 102]. While this approach is more accurate, it is resource-intensive, expensive, scales poorly to high bandwidths, does not work on encrypted traffic, and causes tremendous privacy and legal concerns. Two proposed traffic classification approaches that avoid payload inspection are: (1) host-behavior-based, which takes advantage of information regarding "social interaction" of hosts [68, 64], and (2) flow features-based, which classifies the flows based on flow duration, number and size of packets per flow, and inter-packet arrival time [80, 82, 96, 26, 40, 41, 37, 113, 44, 109, 77]. Despite many proposed algorithms for traffic classification, the IT operators do not have definitive answers to some pragmatic questions:

- What are the best available traffic classification approaches ?
- Under what link characteristics and traffic conditions do they perform well ? Specifically, which approaches are well-suited for classifying enterprise traffic ?

- What are the fundamental contributions and limitations of the existing approaches ?

Rigorous comparison of algorithms remains a challenge for three reasons [43] and has been missing in the community. First, there is no publicly available trace data to use as a benchmark, so every approach is evaluated using different traces, typically locally collected, often without payload (ground truth). Second, different techniques track different features, tune different parameters and even define flows and applications differently. Third, authors usually do not make their tools or data available with their results, so reproducing results is essentially impossible.

1.3.1 Traffic Classification Demystified

In order to calibrate the existing approaches, we have conducted a comprehensive evaluation of three traffic classification approaches: *port-based*, *host-behavior-based*, and *flow-features-based*. We evaluate each technique on a broad range of data sets: seven payload traces collected at two backbone and two edge links located in Japan, Korea, and the US. Diverse geographic locations, link characteristics, and application traffic mix in these data allow us to test the approaches under a wide variety of conditions. In order to avoid "tool-bias", we evaluate the performance of CoralReel [83](port-based), BLINC [68] (host-behavior-based) and seven commonly used machine learning algorithms (host-behavior-based). Our analysis reveals the advantages and limitations of each approach and we propose solutions to overcome the limitations. Our study leads to insights and recommendations for both research and practical application of traffic classification.

- **Support Vector Machine (SVM) is accurate**

  One of the interesting results is that we find Support Vector Machines achieving highest accuracy on every trace and for every application. On average, the accuracy is $>$ 98.0 when SVM is trained with more than 5,000 flow (2.5% of the size of the test datasets).

- **Choice of features**

  Choice of the right features is important. We find a set of single-directional dominant key flow features that appear consistently within an application across our traces; ports, protocol, TCP flags, and packet size. A limitation of the previous attempts based on flow features [80, 82, 96, 26, 40, 37, 113, 44, 109, 77] is that they use bi-directional TCP connection statistics which are not applicable to UDP traffic (or traces collected in backbone links). Backbone links (although not directly relevant to enterprise network) see both directions of traffic under (atypical) symmetric routing conditions. However, the links in an enterprise networks see bi-directional flows. We observe that *port number* information is one of the most important discriminators, particularly when used in combination with other flow features such as packet size information, TCP header flags and protocol.

- **Host-behavior approach is effective in enterprise networks**

  We find that the accuracy of host-behavior-based methods such as BLINC strongly depends on the location from which the trace is collected. If location is the border link of a singled-homed edge network, BLINC performs well as the trace will contain bi-directional flow information of the enterprise end-hosts. However, BLINC is not recommended, if a trace (i) contains a small portion of behavioral information for each host and (ii) misses one direction of traffic. To mitigate the limitation of BLINC on backbone traffic classification, we extend BLINC to identify some application traffic (e.g., Web, P2P) even when both directions of flows are not observed. This process significantly improves the accuracy on backbone traces by as much as 45%.

1.4 An Overview of This Thesis

This thesis is logically divided into several chapters. Here is the chapter-by-chapter breakdown of the text.

Chapter 2 Related Work

This chapter discusses the related work, the limitations and strengths of different approaches.

Chapter 3 Diversity in Configuration

This chapter discusses in details the concept of diversity. We also describe the network setup and data collection process. Initially the chapter focusses on the percentile detector commonly used by the operators. Then, we propose new policies and finally validates the proposals through evaluation on real traces.

Chapter 4 Hierarchical Change Explanation

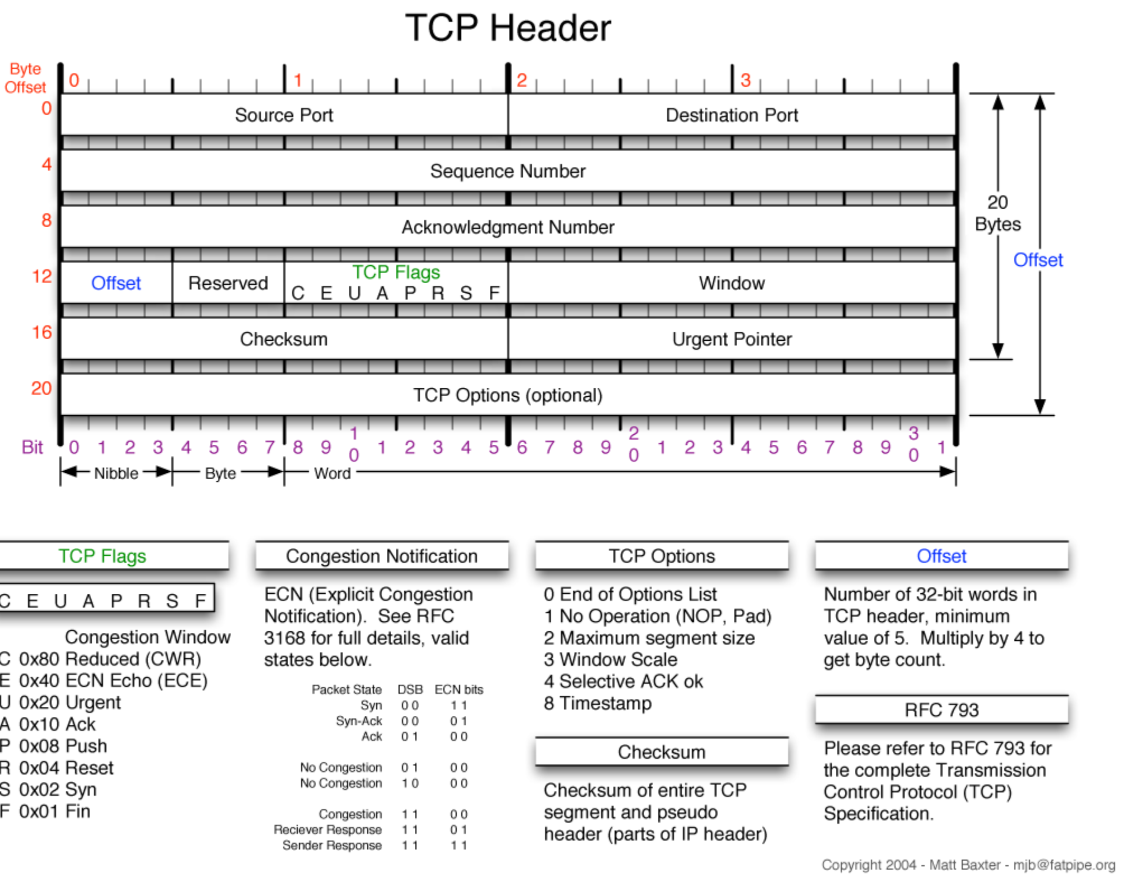
This chapter discusses in details the concept of hierarchical change and explanatory terms in hierarchical summary. The chapter describes several solution approaches and presents an optimal algorithm to compute changes in hierarchical summary. The chapter introduces the application of Exponential Weighted Moving Average (EWMA) smoothing filter to predict measure values at the leaves of a hierarchy along with along associated confidence levels.

Chapter 5 Application Traffic Classification

This chapter critically re-visits three application traffic classification approaches. Through a detailed evaluation on variety of traces, different insightful conclusions are made which can be useful for the operators. The evaluation can lead one to understand the limitations and strengths of different approaches.

Chapter 6 Conclusion and Future Work

This chapter summarizes the contributions made in the thesis. We discuss possible implications of different assumptions made in the thesis. We discuss various future avenues into which this thesis can lead to.



TCP Flags

C E U A P R S F

Congestion Window
C 0x80 Reduced (CWR)
E 0x40 ECN Echo (ECE)
U 0x20 Urgent
A 0x10 Ack
P 0x08 Push
R 0x04 Reset
S 0x02 Syn
F 0x01 Fin

Congestion Notification

ECN (Explicit Congestion Notification). See RFC 3168 for full details, valid states below.

Packet State	DSB	ECN bits
Syn	0 0	1 1
Syn-Ack	0 0	0 1
Ack	0 1	0 0
No Congestion	0 1	0 0
No Congestion	1 0	0 0
Congestion	1 1	0 0
Reciever Response	1 1	0 1
Sender Response	1 1	1 1

TCP Options

0 End of Options List
1 No Operation (NOP, Pad)
2 Maximum segment size
3 Window Scale
4 Selective ACK ok
8 Timestamp

Checksum

Checksum of entire TCP segment and pseudo header (parts of IP header)

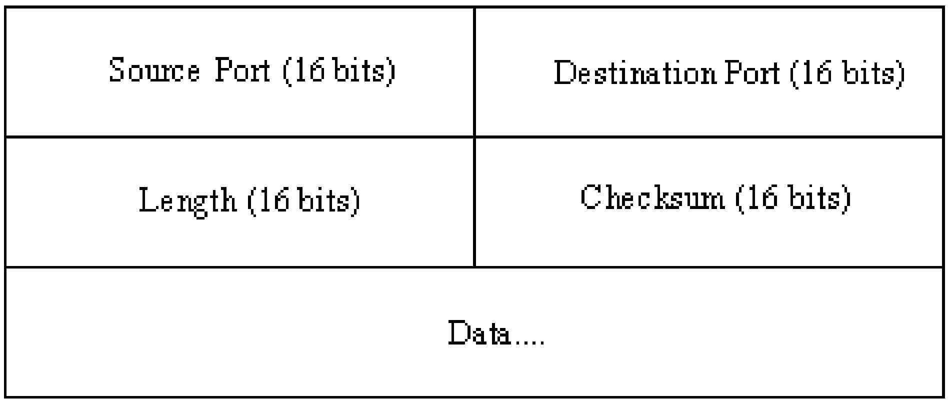
Offset

Number of 32-bit words in TCP header, minimum value of 5. Multiply by 4 to get byte count.

RFC 793

Please refer to RFC 793 for the complete Transmission Control Protocol (TCP) Specification.

(a) TCP header



(b) UDP header

Figure 1.2: Different fields in the (a) TCP header and (b) UDP header

Chapter 2

Related Work

This chapter describes the background and current work. Section 2.1 discusses existing work on botnet attack prevention measures. Section 2.2 discusses the recent work on change detection in (non)hierarchical data. Section 2.3 discusses the recent work on traffic classification along with their advantages and disadvantages.

2.1 Botnet Protection

Different approaches have been suggested in the literature to detect and mitigate DDoS attacks (due to botnets) [50, 51]. IT operators commonly use blacklists to block connections. However keeping the blacklists up to date is challenging. Moreover, an attacker can exploit a system if the white list information is compromised. Attacks on popular sites like Google or Yahoo! will be hard to detect.

With the growth of encrypted traffic in a network, it becomes difficult for in-network analysis [107], calling for more research on HIDS. HIDS typically have two components, signature detection and anomaly detection. Signature detection plays an important role, but it is not useful in detecting previously unknown attacks. Anomaly detection tracks pre-defined features of the end-host traffic, defines normal behavior and then raises alerts when abnormal behaviors are observed. In the context of a program execution, one class of anomaly detectors has been used where alerts are raised when rules on expected program behavior are violated [107, 58, 47, 38]. In another class, anomaly detectors build statistical models of application layer or networking layer traffic[54, 65, 74]. However, the drawback of this class of detectors is that they, routinely, generate false alarms. However, it is important for the enterprise to control the false alarm rate of the anomaly detectors and prevent IT security operations center (SOC) from getting overwhelmed with false alarms. In this thesis, we use the example of statistical anomaly detectors at the networking layer that are intended to help thwart Distributed Denial-of-Service (DDoS) attacks (often conducted through botnets).

According to [49], there are three approaches for stopping botnets: (1) stopping systems from being infected; (2) detecting the *command and control* (C&C) communication within botnets; and (3) detecting the secondary features of bot infections such as the attack behavior itself. Antivirus software addresses the infection problem, while other research activity (such as [65]) studies how to detect the command and control channel activity. In [49], the authors have called for more research to be done in the third approach and our thesis falls into that category. Several proposals exists which opt for approaches 1) and 2). Our thesis orients in the direction of detecting the secondary features of bot infections. BotSniffer [50] falls under the second approach and it proposes a framework to detect botnet traffic within a network by exploiting the spatio-temporal correlation and similarities of responses to control commands issued by botmaster on C&C Channels. The idea in the paper is that bots within a botnet act in a synchronized fashion in that they execute the same command (e.g., do scanning or gather system information), and report the results to the botmaster. The author observe the responses the commands that bots take from a centralized server over HTTP or IRC and find two invariants. The bots in a botnet remain connected to the C&C servers to obtain commands. Since the bots perform similar tasks, their responses to the server bear strong similarities and the network traffic has crowd-like behavior.

Several solution approaches address the botnet mitigation through changes or additions in network architecture or configuration. Authors in [55] have studied the characteristics of Storm Worm botnet and have suggested that polluting the contents in Storm P2P network might deter botnet zombies from using P2P network. The idea is that pollution in the search content will deter the bots from searching (based on keys) and therefore stop them from propagating. In another approach [46], the authors propose a network architecture in which an end-host communicates with a destination and traffic is forwarded through random intermediate nodes such that an attacker can cause only a fraction of a given flow to get lost.

Commercial vendors offer a number of anomaly detectors and mitigation solutions[32, 7]. Comcast blocked port 25 for customers generating heavy traffic to prevent its network from being spam hub but it might lead to high false positives and legitimate mails from the customers could be blocked. According to Comcast, port blocking resulting in 35% reduction in spam in their network. However, Comcast is having hard time defending against botnets which use open relays or tunneling through compromised nodes outside network. While port blocking is a conservative approach, it might lead to high false positives and legitimate mails from the customers could be blocked.

2.1.1 Diversity Approach in Detecting Stealthy Attacks

If a botmaster recruits an enormous number of zombies, then in order to launch a DDoS attack on a victim, each zombie need not create a blatantly obvious flood. Commanding each zombie to send small amount of traffic to the victim could overwhelm the victim, i.e., stealthy botnets. Since the attacker relies on the sum of the activity of its zombies, it can try to evade detection by commanding each zombie to send apparently reasonable amounts of traffic to the victim, i.e., stealthy botnets.

In this thesis, we see the effectiveness of our approach on the success of a botmaster who attempts to hide the attack inside normal user traffic. By this we mean, that the attacker issues commands to transmit an amount of traffic towards a victim that is not dissimilar to the user's regular traffic patterns, termed as *evasion* or *mimicry* attacks. In order to prevent an attacker from evading mimicry attacks, many defensive mechanisms try to eliminate homogeneity in the systems at different levels. Eliminating homogeneity increases diversity in the systems which forces attackers to do more work in launching attacks. Diversity as such has been applied in different contexts of computer systems. Forrest et. al proposes several methods of achieving software diversity in [58]. They proposed that randomization can be introduced in the code generation process of compilers. They showed that by allocating random padding on stack frame size, several buffer overflow attacks could be deterred. While such proposals are promising, their work does not explore diversity at the layer of network traffic. Although there is a large body of research work which propose to introduce diversity in the form of randomization at different abstraction levels (such as in applications or compilers [58]), we feel the concept of diversity has not been explored in anomaly detection systems formally [97]. We believe this takes a stride to explore the benefits and trade-offs in this domain.

2.2 Hierarchical Change

Hierarchies on data attributes have played a significant role in data warehouses, for which database operators such as the datacube have been developed to summarize and navigate the data at the different levels of aggregation [30]. In the data mining literature, several tools have been proposed for summarizing hierarchical data at a single time instance, including GMDL regions [76], Icecubes [56], and Hierarchical Heavy Hitters (HHH) [45, 34].

With respect to detecting changes in data, recent approaches include velocity density estimation [16] for visualizing change, windowed statistical testing [71] for detecting distributional changes, and histogram differencing [35] for identifying items which exhibit the largest changes in frequencies. However, these papers deal with flat (non-hierarchical) data. There have been a few papers explicitly dealing with hierarchical data. Zhang et al. studied change detection of (aggregated) time series corresponding to HHH IP prefixes in the

IP address hierarchy [114]. Chawathe et al. studied the problem of change detection on semi-structured data, but for topological changes [101]. Estan et al. [36] studied the computation and change in hierarchical data however their approach is based on heuristics and their approach does not address the issue when the count measures in the leaves are not exact.

The problem of path explorations of hierarchies was studied in [100]. Here the user defines a set of linear constraints and the values in the datacube cells are predicted using the Maximum Entropy Principle. Given a supplied model, the technique finds the cells that are significantly different values from the expected values. Our problem is essentially the opposite: to find the best model that explains the changes. There is also some marginally related work on identifying bursts in hierarchical time series data, that is, the time intervals tightly capturing high arrival frequencies [115, 73].

Most related to our work is the DIFF operator for explaining differences in the datacube [99]. In their problem, a user selects two aggregates at the same level in the datacube which fixes some of the dimensions. The ratio between the selected aggregates is then explained in terms of the free dimensions, and subaggregates having deviating ratios explained recursively. The aggregates in the subcubes corresponding to the free dimensions are examined to find those which deviate most from the ratio, and the remaining pairs are approximated with the ancestor ratio, recursively. This puts constraints on the intermediate node ratios, whereas our solution has the freedom to explain leaf aggregate changes in terms of intermediate node ratios, and is thus more parsimonious. In the example (Figure 1.1) involving the IP prefixes 1.2.3.10/31 and 1.2.3.13/31, the need to additionally "explain" the ratio of $\frac{3}{2}$ at the IP prefix 1.2.3.10/30 level internal node results in a verbose explanation using the DIFF approach.

The problem of using compact hierarchical histograms for approximating leaf-level data was studied in [94] which employed a predefined hierarchy, similar to our approach but solved the dual problem: given a bound on the size of the synopsis (i.e., the number of explanations), find the synopsis that minimizes the error. Further, the paper considered three different partitioning functions and solved via dynamic programming to reduce distributive error metrics given a space bound. Their LPM variant is the same problem studied in [99] but solved heuristically due to the expensive cost of distribute error metrics; the other partitioning functions find inferior solutions to LPM. Our work is based on the initial problem formulation presented in [24].

2.2.1 Connection to Wavelets

Recently [85] investigated a problem similar to our work. The proposed solution used the Haar wavelet representation to construct dataset synopses of minimum space. The use of the wavelet representation restricts this approach to less efficient (i.e., less parsimonious) explanations than our hierarchical parsimonious explanations. Another problem relevant to ours (for the case of binary hierarchies) is Haar wavelet compression with maximum-error metrics, introduced in [79]. The best current solution requires $O(n^2)$ time and $O(n)$ space to solve the dual problem [60] and, just as in [99], constraints are imposed at all nodes rather than just at the leaves, leading to less parsimonious solutions. [61] introduced the notion of unrestricted Haar wavelets and [69] defined the Haar+ tree as an improvement, but these exploit discretization of values and therefore are not comparable with our approach which allows for any (potentially infinite sized) domain. However [69] is equivalent to the model given by [94] when the hierarchy is restricted to binary trees. [69] presents provably good approximate algorithms to solve this problem in $O(R^2 n \log n \log^2 B)$ or $O(R^2 n \log^2 B)$ time (n is the size of the input, B the maximum number of coefficients in the synopsis and R the number of the examined values per coefficient), for general error metrics. Interestingly, our problem (with binary hierarchies) offers an alternative to Haar wavelet compression, yielding better answers with smaller complexity: $O(n \log n)$ for the primal problem and $O(n \log n \log \epsilon^*)$ for the dual. An algorithm that solves the dual problem (as in [94] or [69]) can be modified to solve the primal problem using a binary search procedure on B. Thus, these algorithms would need to run an additional $\log B$ factor slower if modified to solve our problem.

2.3 Traffic Classification

Here we will discuss about traffic classification using different methods. We will discuss the advantages and disadvantages of these methods.

2.3.1 Port-based approach

Traffic classification based on port numbers [2] is a fast and simple method, but several studies have shown that it performs poorly, e.g., less than 70% accuracy in classifying flows in an enterprise data set [39, 81]. We acknowledge the coarseness of assessing performance over an entire trace rather than for the applications actually using well-known ports [43]. This performance metric essentially indicates the amount of traffic in the trace using well-known ports, which can vary widely, and does not classify traffic that is mis-using well-known ports assigned to a different application.

2.3.2 Payload-Based Approach

Payload-based classification algorithms inspect the packet contents to identify the application. Once a set of unique payload signatures is available for an application, this approach produces an extremely accurate classification. After early works showed the value of payload signatures in traffic classification [31, 81, 102], others have proposed automated ways to identify such signatures [62, 78, 89]. However, [62, 78] have evaluated these automated schemes only on conventional applications such as FTP, SMTP, HTTP, HTTPS, SSH, DNS, and NTP, not on newer applications such as P2P, games, and streaming, while [89] evaluated their proposed scheme on a few P2P file sharing applications. We use the payload-based classifier developed in earlier efforts [68, 40, 111] to establish ground truth for our traces.

2.3.3 Host-behavior-based approach

The host-behavior-based approach was developed to capture social interaction observable even with encrypted payload [67, 66, 68, 64]. For example, BLINC [68] captures the profile of a host, in terms of the destinations and ports it communicates with, identifies applications the host is engaged in by comparing the captured profile with (built-in to BLINC) host behavior graphlets/signatures of application servers, and then classifies traffic flows. While BLINC's approach is promising on edge links, it assumes the observation of both directions of traffic, which limits its applicability. Recently Iliofotou, *et al.* proposed a network-wide behavior-based traffic classification method, Traffic Dispersion Graphs (TDGs) [64], which focuses on network-wide behavioral patterns of interacting hosts.

2.3.4 Flow Features-based Approach

Substantial attention has been invested in data mining techniques and machine learning algorithms using flow features for traffic classification [80, 82, 96, 26, 40, 41, 37, 113, 44, 109, 77]. Nguyen *et al.* surveys, categorizes and qualitatively reviews these studies in terms of their choice of machine learning strategies and primary contributions to the traffic classification literature [88]. Their survey is complementary to our work, where we pursue quantitative, measurement-based, performance evaluation of the seven machine learning algorithms using multiple datasets collected from Japan, Korea, and the US.

Machine learning algorithms are generally categorized into *supervised learning* and *unsupervised learning* or *clustering*. Supervised learning requires training data to be labeled in advance and produces a model that fits the training data. The advantage of these algorithms is that they can be tuned to detect subtle differences and they clearly label the flows upon termination, unlike the unsupervised ones. Unsupervised learning essentially clusters flows with similar characteristics together [40, 75]. The advantage is that it does not require training, and new applications can be classified by examining known applications in the same cluster. Erman *et al.* [40] compared the performance of unsupervised machine learning algorithms in traffic classification. Since our main focus is on evaluating the predictive power of a trained traffic classifier rather than on detecting new applications or flow clustering, we focus on supervised machine learning algorithms in this thesis.

Chapter 3

Diversity in Configuration

In this chapter, we will discuss the benefits of diversity by studying the traffic profile of a population of employees in a large enterprise. In order to illustrate the effectiveness of diversity, we propose a model in which the end hosts are required to find thresholds for their anomaly detectors so that malicious traffic are detected readily. The chapter shows that the problem of threshold computation comes with myriad of issues that need to be addressed but are currently overlooked. Our approach proposes generalization techniques including two components - a way to combine false alarm rate and mis-detection rate and a way to classify the end hosts into groups that can be managed effectively as a single unit. Finally, the chapter evaluates the proposals on real traces assuming a full-knowledge attacker strategy.

3.1 Network Setup

The problem we study is the configuration of alert thresholds in HIDS for a *group* of employees in an enterprise setting, when a single IT policy is followed. Given a set of features F_i^j, where i is the user index, and j is the feature index, set the threshold for each feature such that when it exceeds the threshold an alarm is raised. On the one hand, enterprise IT usually approaches this problem by applying a single policy, such as “set the thresholds the same for all users”. On the other hand, as we will show, there is a great deal of diversity in the end-hosts’ traffic distribution. This diversity implies that such a policy will have unintended consequences, namely that the performance in terms of false positives and false negatives differs dramatically across the end-hosts. Let each end-host’s performance be measured by the tuple $\langle$ false positive rate, false negative rate $\rangle$, denoted by $\langle \mathsf{FP}_i, \mathsf{FN}_i \rangle$ for user i. The problem we address is to determine a good enterprise policy so as to achieve a better balance across all the performance tuples of all end-hosts. We will provide algorithms to carry out different policies.

We focus on a set of features that are either in use today in industrial products, or have been proposed in the literature. Our intent is not to promote any particular feature, but rather to develop a methodology for configuring cutoff thresholds for features. Today’s systems use one set of features, however because the security threat landscape is continuously evolving, that set of features is likely to change. We believe our methodology will be useful to any set of features that are additive. Examples of additive features include the `number_of_DNS_connections` (used in Damballa's botnet detection system[1]), `number_of_HTTP_connections`, `number_of_TCP_SYN` (used in BRO on a per source basis[90]) , `number_of_TCP_connections` (used in Cisco's endhost CSA product[32]), etc.

We consider an enterprise network in which end-host i, $i \leq i \leq n$, has been enslaved by a botmaster commanding either through an IRC C&C channel or a P2P system. We assume that these n zombies could be simultaneously used to attack an external host, H. Let g_i^j represent the normal value for end-host i of a specific feature j. This could represent any of the features (i.e., detectors) mentioned above because they are all additive features. We assume that when the attacker (a term we use interchangeably with the term *botmaster*) issues a command, each zombie responds in a way that increases the value of feature j by b_i^j. Thus b_i^j captures the attack size for any particular feature. (For ease of discussion, we drop the index j, as the same reasoning applies to each feature, as long as it is additive. Similarly we discuss, as an example, connection counters rather than packet counters to simplify the presentation.) Each end-host has a probability distribution $P(g_i = G)$ that describes the probability that the host will open G connections in the next time window. Thus, the total size of traffic corresponding to a particular feature, emanating from end-host i, is described by a random variable $b_i + g_i$.

The IT operator configures the i^{th} host with a threshold parameter T_i such that if the outgoing traffic on host i exceeds T_i, the host raises an alert and sends it to the central IT

[1]http://www.damballa.com/

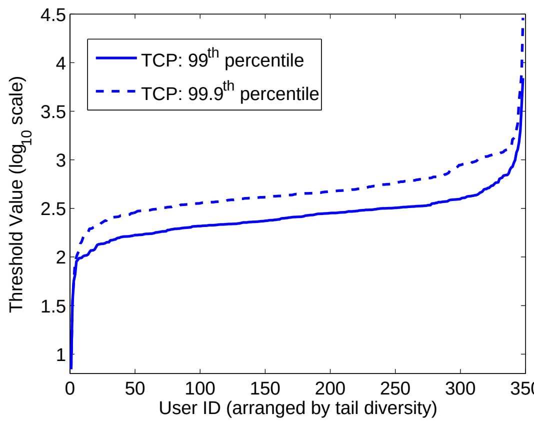
security operations center (SOC). The probability of missing an attack of size b_i, a false negative (FN_i), is given by $P(g_i + b_i < T_i)$, and the probability of a false alarm (FP_i) for that user is simply $P(g_i > T_i)$. The problem is to determine a set of thresholds values T_i for all users. We propose different policies for threshold selection, and provide algorithms to implement such policies. We compare the performance of the different policies from three perspectives (using three metrics). First, we look at the $\langle \mathsf{FP}_i, \mathsf{FN}_i \rangle$ performance tuple for a group of users using scatter plots and compare which policy achieves a good balance for the majority of the users. Second, we compare the average number of false positives generated at the centralized IT operations center under different policies. Third, we compare the sizes of the attacks a full-knowledge attacker is able to successfully launch (i.e. evade detection) under the different policies.

3.1.1 Threat Model

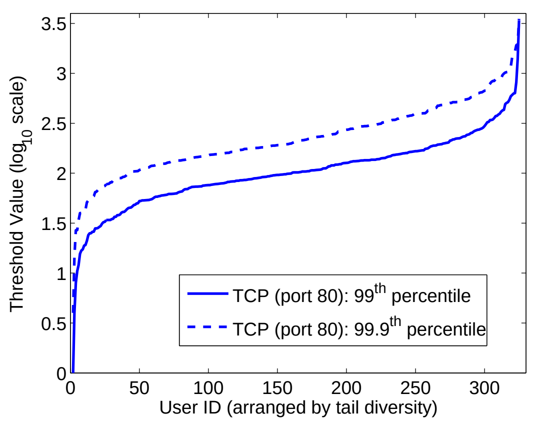
We consider a strong threat model for the attacker. One can devise an attacker strategy based on the amount of information the attacker has about the zombies and the amount of extra work the attacker is willing to do to acquire detailed information about its zombies. We consider an attacker that has complete knowledge of each zombie's normal traffic profile because he/she installs monitoring code on the compromised host. Namely, our full-knowledge attacker knows the end-host's probability density function, pdf for each feature j, $P(g_i^j)$, and their respective cutoff thresholds, T_i^j. The attacker can use this knowledge to select b_i^j tailored to each zombie, such that the probability of hiding inside the user profile traffic, namely

$P(g_i + b_i < T_i)$, for any j, is fairly high (we set it to 90%). Although we have not used but one can think of another strategy in which the attacker chooses b_i which maximizes the damage given by $\sum b_i P(g_i + b_i < T_i)$.

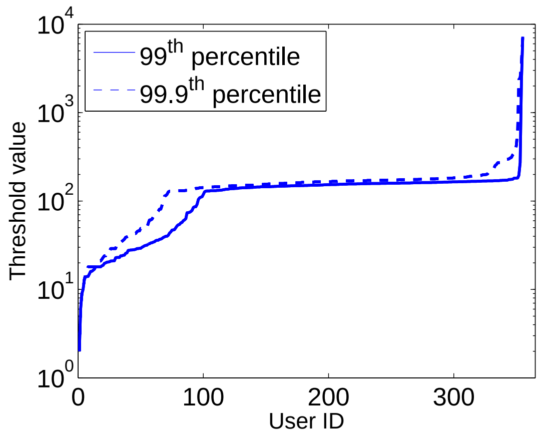
3.2 Diversity in User Population



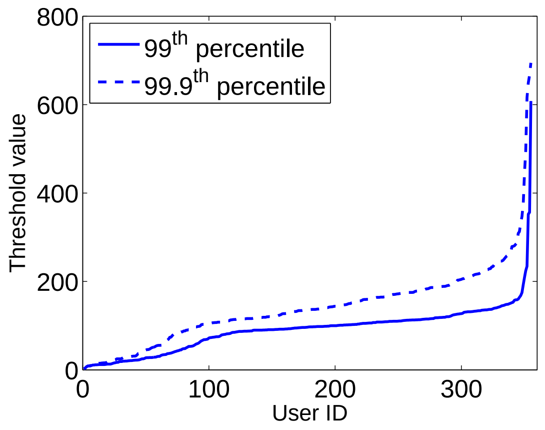
(a) # TCP Connections



(b) # TCP Connections on port 80



(c) # Distinct Connections



(d) # DNS Connections

Figure 3.1: Tail Diversity: (a) # TCP connections can indicate TCP basic DDoS; (b) # HTTP connections can indicate end-hosts participating in click-frauds; (c) # Distinct connections can reveal address scanners; (d) # DNS connections can detect scanning, worm spread;

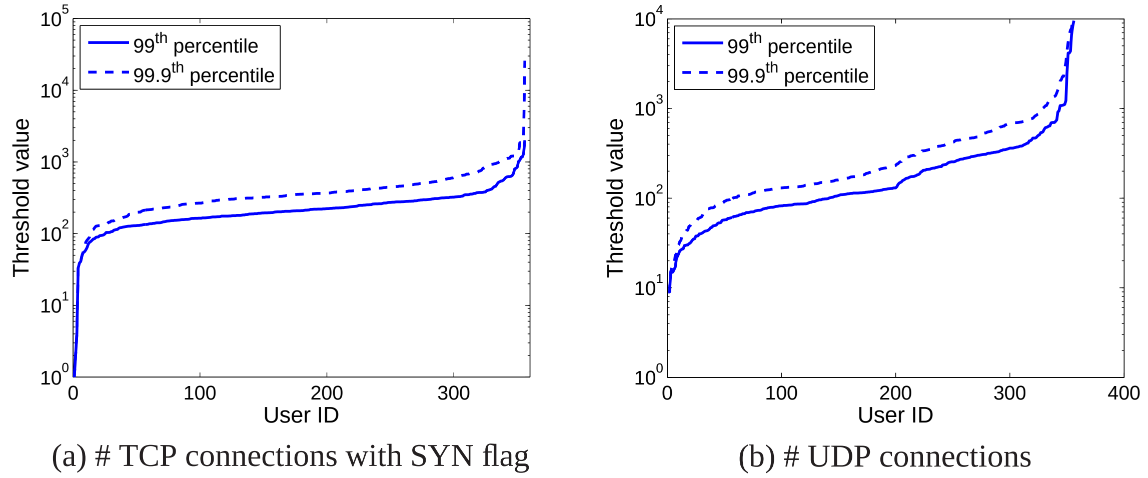


Figure 3.2: Tail Diversity: (a) # Connections with SYN flags set can reveal end-hosts participating in SYN flood; and (b) # Connections with UDP connections

In this section, we will present the characteristics of normal traffic from 350 end-hosts in a large enterprise.

3.2.1 Data Collection

Our data consists of network packet traces collected at 350 end-hosts (95% of them are laptops and all hosts were using Windows XP) in a large enterprise network. The traces span over 5 weeks in Q1 of 2007. Each end-host corresponds to an individual user and all users enrolled on a volunteer basis. Users from many different geographies participated: with 73% of the users from the United States, 13% from Asia, 13% from Europe, the Middle East and Africa and 1% from South America. Most users were located in large office sites in metropolitan areas. All end hosts were using the Microsoft Windows XP operating system.

The data collection was performed by a stand alone application (a wrapper around the `windump` tool). In addition to collecting packet headers, our collection tool watched for changes in IP address, interfaces (e.g., wired/wireless) and location. Because the collection was performed directly on the end-host, all packet activity was captured, even when the mobile laptops changed environments (home,work, different wireless interfaces, etc). This dataset captures an unusually complete view of users' behaviors; data collections that are carried out at gateways and routers do not capture user activity when they leave work or switch to another network. Note that in the organization, each employee is given one laptop and these laptops are not shared across employees. We believe that this is common practice, so unless employees share their laptops with family members at home in the evening, the laptops should correspond to one user.

3.2.2 User Traffic Characteristics

We processed the tcpdump traces from 350 end-hosts using Bro tool [90] and constructed time-series for each of 5 anomaly detection features. The features we studied are the number of TCP connections, HTTP connections, distinct connections, all connections and SYN Flood connections[2], aggregated the counts into 5 and 15 mins interval bins. In this paper, we present the results for the 15mins interval bins (having 5mins interval bins does not change the conclusions). We selected these features because they are actual features used on various systems ([32, 5, 90])[3]. We treat each bin count as a sample point of the distribution $P(g_i)$ for the i^{th} end-host. We assume that the time-series are stationary. Once we obtain the distributions, $P(g_i)$, we compute 99^{th} and 99.9^{th} percentiles of each feature distribution as cut-off

[2]number of connections in which TCP SYN Flag is set

[3]Features chosen from [90] were only those that are computed on a per source basis.

thresholds and plot the sorted thresholds in Fig. 3.1 and 3.2. We consider this definition of outlier here because it is adopted in practice due to its simplicity.

If all hosts were to self select their cutoff thresholds to be either the 99^{th} and 99.9^{th} percentile values, then the threshold would be meaningful in terms of their own behaviors (all users would experience a common false positive rate). In Fig. 3.1 and 3.2 we show the tremendous diversity in choice of thresholds that would result in this personalized policy were adopted. Interestingly, the range of diversity varies by 3 to 4 orders of magnitude. This demonstrates in a loose sense, that the "tail", or fringe, of the user's behavior begins in very different places for different users. Fig.3.1(a) illustrates that these "tails" can range from 7 to 7000 for a false positive rate of 1% [4].

3.3 Basic Policy Comparison

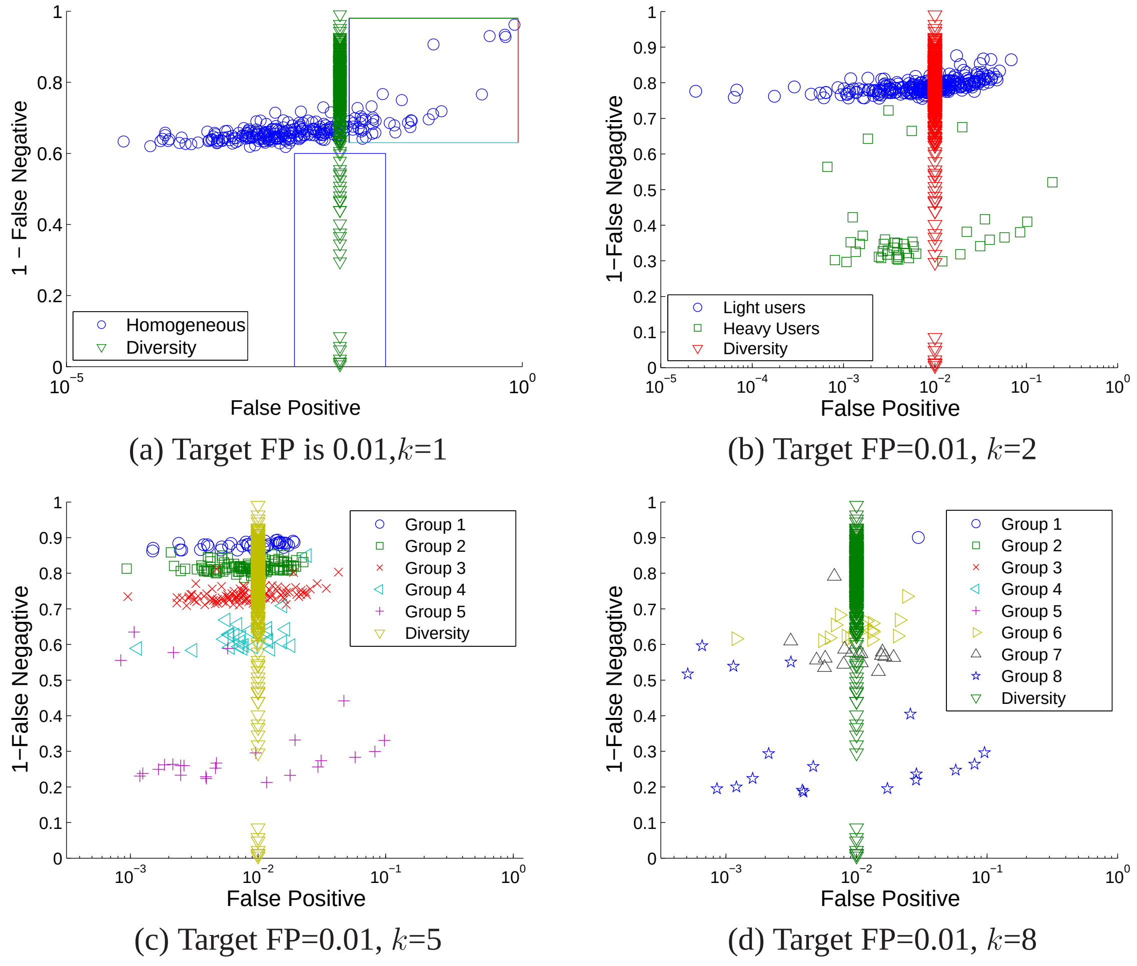
We now compare the impact of homogeneous and diversity threshold policies on the end-hosts and enterprise as a whole. We look at three metrics, the $\langle \mathsf{FN}_i, \mathsf{FP}_i \rangle$ tuple for all users, the false alarms received inside an enterprise operation center, and the attack effectiveness. For this initial policy comparison, we consider a simple percentile detector, a method commonly used in practice. A percentile detector uses the distribution of a feature, $P(g_i)$ and computes a threshold T_i such that $P(g_i > T_i)$ equals desired pre-determined false positive rate. Such a T_i does not ensure any false negative rate since false negative depends on the attack size

[4]The end-host with a tail starting at 7 most likely comes from a machine that is rarely used. The few connections occurring could correspond to IT scanning events. Some users might have installed our tracing code on machines that they do not really use much. We cannot be sure as all user identities have been anonymized.

distribution. As an example, we work with 99^{th} percentile detectors, a target many in our IT survey confirmed was common.

Under the full-diversity policy, all the end-hosts target a false positive rate of 1%. A simple distributed solution for this policy is letting each end host compute its own histogram for the relevant features and extract the 99^{th} percentile value. For the homogeneous policy, we assume a central solution that merges all the time series from the end-hosts into a composite histogram and calculates the thresholds on this aggregate. We use both simulated attack data (so we can vary and test attacks of all possible sizes), and real attack data from live malware traces - both of which are replayed on the actual user traces. For the simulations, we assume the full knowledge attacker aims to evade detection with a 90% success rate, and thus selects a tailored attack size b_i for each user such that $P(g_i + b < T_i)$ =0.9. We generalize this definition of FN_i as $\int_0^{B_{\max}} P(g_i + b < T_i) f(b) db$ where $f(b)$ denotes a general distribution of attack sizes. Because the size of DoS attacks is not well documented, and is forever evolving, we study the full range of possible sizes - from near zero to the largest number of connections among all users [1,B_{max}]. Beyond this size, DoS attacks will easily stand out as they no longer mimic user traffic patterns. We use a uniform distribution on the attack size, i.e., $f(b) = \frac{1}{B_{\max}}$, for simulations (since no such distribution is known, and our only goal here is to try them all exhaustively).

To understand the $\langle \mathsf{FP}_i, \mathsf{FN}_i \rangle$ trade-offs that each user incurs, under a particular policy, we use End-User Performance Characteristics Curves (EPC) as in Figure 3.3. Each point on the EPC corresponds to an end-host. y-axis shows the detection rate (1-FN_i) and the x-



(a) Target FP is 0.01,k=1

(b) Target FP=0.01, k=2

(c) Target FP=0.01, k=5

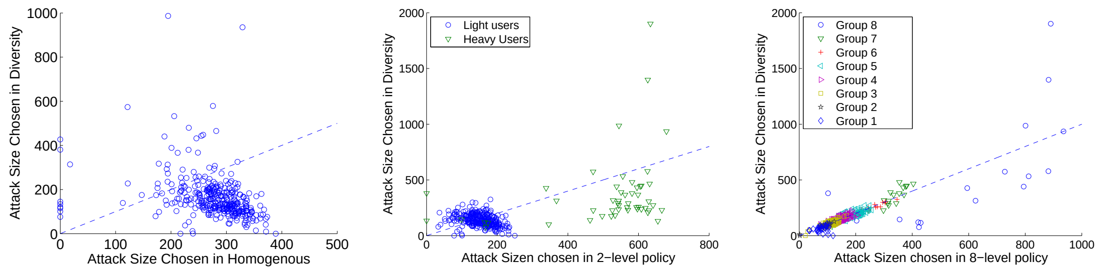
(d) Target FP=0.01, k=8

Figure 3.3: End-User Performance Characteristics (EPC), each point corresponds to (FP,1-FN) for that user, using 99^{th} percentile detectors. (a) end-hosts using diversity approach has better trade-offs; (b)-(d) performance under homogeneous approach can be improved under 2-level, 5-level and 8-level approaches.

axis shows the false positive rate. Figure 3.3 (a) illustrates what happens to the ensemble of enterprise hosts. Under the diversity approach, all the end-hosts experience the same FP_i rate of 0.01, however the detection rate of the users are spread throughout the 0 and 1 range. The bulk of the users (about 85%) achieve high detection rates over 65%. The homogeneous approach has the opposite effect: the end-hosts experience a fairly similar detection rate (between 60-70%), but the false positives are spread across a large range. We believe that this dramatic differentiation in FP performance across hosts is a ramification that is unintended by IT operators, and arises because of their ignorance of diversity in user fringe behavior.

The IT operators are faced with a fundamental trade-off in terms of policy: would they prefer the end-hosts to experience similar FP rates or similar FN rates? Neither of these policies is ideal in that neither can benefit *all* the end-hosts in the same way. Although these policies will benefit different subsets of end-hosts, the subset of end-hosts (45 out of 348) appearing inside the two boxes (low performance areas) are primarily the heavy end-hosts (high mean and variance). Given the choice between these two effects on the user population, we believe that IT operators should choose the diversity approach. There is a way to compensate for the end-hosts with poor performance in the diversity scheme, namely via collaboration. Enterprise end-hosts can collaborate by sharing information like detectives [52], in which the early-to-detect hosts inform others about attacks they cannot see. However there is no way to compensate for the poor performing end-hosts under the homogeneous policy.

False Alarms: The diversity approach is also preferable from the point of view of the



(a) FP=0.01, diversity (b) FP=0.01, 2-level policy (c) FP=0.01, 8-level policy

Figure 3.4: Attack sizes chosen by attacker under different policies. (a) under homogeneous policy, more end-hosts enable attacker to pick larger attack sizes; (b)-(c) partial diversity leads to smaller attack sizes than homogeneous. $x = y$ line indices similar performance between two policies being compared.

enterprise, because it reduces the total number of false alarms arriving to the SOC. Because the nodes with FP_i rates > 0.01 in the EPC curve under the homogeneous approach are the "heavy" ones, they will result in an enormous number of false alarms - can be approximated by $\sum_i E[g_i]\mathsf{FP}_i$. For these test scenarios, the homogeneous approach generates on average 1450 alerts whereas the diversity approach generates on average only 200, which is lesser by a factor of 7!

Reduction of Attack Sizes: Fig. 3.4 (a) shows the attack values chosen by an attacker under different policies when the attacker's goal is to evade detection with 90% success rate. Each point corresponds to one end-host and we observe most of the points are below $x = y$ line implying that the attacker can choose larger attack sizes under homogeneous approach. This illustrates that the diversity approach limits the effectiveness of DDoS attacks, even for full knowledge attackers. (We will explain the (b) and (c) plots of these figures later on.)

3.4 New Policies

In this section, we discuss more general policies for computing the thresholds for the end-host anomaly detectors. The IT operators have to decide upon two components: (1) a particular utility function which explicitly states a balance between FN_i and FP_i and (2) a grouping policy to batch the end-hosts with similar profiles and thus choosing a level of diversity.

3.4.1 Utility Function

Ideally an end-host would like to experience no false positive and 100% detection rate. However, in practice a statistical anomaly detector will have an operating point $\langle \mathsf{FP}_i, \mathsf{FN}_i \rangle$ but an end-host can explicitly combine FP_i and and FN_i through utility functions. We assume that all the end-hosts use the same form of utility function which is given by:

$$\mathsf{U}_i(T_i, B_{\max}) = -[w\mathsf{FN}_i(T_i, B_{\max}) + (1-w)\mathsf{FP}_i(T_i)] \tag{3.1}$$

where $0 \leq w \leq 1$ and U_i is the i^{th} end-host's utility function. The goal is to find the optimal threshold, $T_i^* = \arg\max_{T_i} U_i(T_i, B_{\max})$. Depending on the values of T_i and $B_{\max}$, the values of U_i can vary between 0 (most desired) to -1 (least desired). If T_i is very large and $B_{\max}$ is small, $\mathsf{FP}_i \to 0$ and $\mathsf{FN}_i \to 1$. If T_i is very small and $B_{\max}$ is small, $\mathsf{FP}_i \to 1$ and $\mathsf{FN}_i \to 0$. If $B_{\max}$ is very large, then FN_i might be 0 and become insensitive to T_i.

The IT operators will first choose a particular utility function (determined by w) to be used by the end-hosts. In one case, the end-hosts will optimize their utility functions to compute

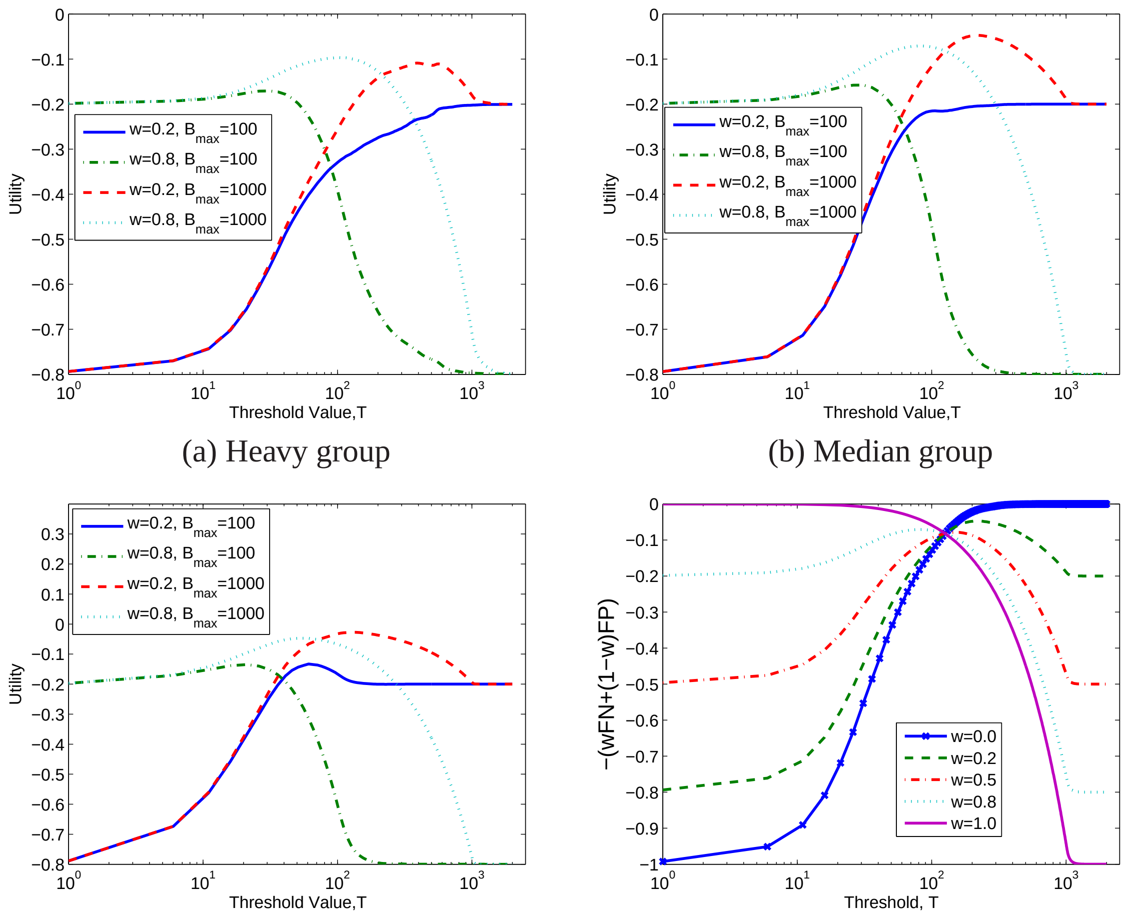
T_i^* and the behavior of each will have no impact on the threshold selection of others. In another case, IT operators will opt for a centralized solution where they will collect the data from all the end-hosts at SOC and compute the thresholds for the end-hosts. We will discuss in Section 3.4.2 the issue how many thresholds should the IT operators compute. Should the IT operators want to compute one threshold for all the end-hosts, they will use one combined utility function which is given by:

$$\mathsf{U}(T, B_{\max}) = -\sum_i [w\mathsf{FN}_i + (1-w)\mathsf{FP}_i] \qquad (3.2)$$

The goal for the IT operator is to optimize the utility function and find an optimal threshold, $T^* = \arg\max_T \mathsf{U}(T, B_{\max})$ and configure the end-hosts with T^*.

Behavior of Utility Functions: Having defined the utility functions, we will discuss the characteristics of the utility functions for three different end-host groups - *heaviest*, *median* and *lightest*. We group the end-hosts based on the 99^{th} percentile of a given feature distribution, $P(g_i)$. In deriving the false negative values, we choose $B_{\max}$= 1000 which is comparable to largest tail value, 7000. If $B_{\max}$ is too small, no scheme will be able to perform well, and if $B_{\max}$ is blatantly large, any approach will be able to catch an attacker. Therefore, the challenge is to work with an intermediate value of $B_{\max}$. In rest of the thesis, we work with $B_{\max}$= 1000 unless other values are specified and use `number_of_TCP_connections` as the feature for g_i.

First, we study the variation of the utility values as a function of threshold, T_i. Fig. 3.5



(a) Heavy group (b) Median group

(c) Light group (d) Median Group

Figure 3.5: TCP Connections: Utility functions for three different groups – Heavy, Median and Light. Groups are formed based on 99^{th} percentile values. (d) Utility values as a function of w and T_i using number of TCP connections as feature (for median group)

shows the utility values, $U(T_i, B_{\max})$ for the heaviest, median and lightest end-host groups, respectively (for $B_{\max}$ = 100, 1000). Here, we present results corresponding to $w = 0.2$ and $w = 0.8$ as these two values capture the trends.

We make the following observations: (1) An end-host group experiences drastically different false positive and false negative rates for different values of w and $B_{\max}$. This is expected as the end-hosts in different groups have different distributions; (2) different end-hosts experience different false positive and false negative rates using the same utility function; (3) the utility values are not stable when the attack size $B_{\max}$ is around the value of T_i^*. This means if the computed threshold is not exactly equal to the optimal threshold, an end-host will experience drastically different FN_i and FP_i; (4) the utility functions in Fig. 3.5 attain highest utility value for a certain threshold value which we denote by T_i^*. For the same highest utility values, we select the smaller threshold.

We have compared the performance of median users and average performance of the median group. We observe that the performances under both the cases are very similar. We construct the median group using those users whose 99^{th} percentile values are between 150 and 350 inclusive. The tail value of the median user is 222. Fig.3.5(d) shows the utility values for different values of w and T_i. As expected, as w shifts from 1 to 0, the optimal threshold, T_i^* increases. We observe that when the end-hosts have large T_i^*, the thresholds tend to be more diverse.

3.4.2 k-Level Diversity Policy

Although diversity has appealing benefits, we recognize the resistance to this policy IT operators exhibit. We now ask if there is a middle ground between the two extreme policies of purely homogeneous and full n-user diversity (using n thresholds for n end-hosts). Can using a small number of thresholds also be advantageous? With this view, we consider a hybrid approach denoted by k-level diversity policy (alternatively denoted by k-level policy in short).

Combined Utility: In k-level policy, each end host anomaly detector will be configured with one of the k distinct threshold values. IT operators will batch the end-hosts into k distinct groups and compute one threshold for each group. The motivation is that IT operators may want to group the end-hosts with similar profiles together and maintain one threshold for that group. We use the notation $T_{j,k}$ to denote the threshold computed for the j^{th} group when k-level policy is used and let $S_{j,k}$ be the set of end-hosts in the j^{th} group. The composite utility function used for the j^{th} group is given by:

$$\mathsf{U}_{j,k}(T_{j,k}, B_{\max}) = \sum_{i \in S_{j,k}} \mathsf{U}_i(T_{j,k}, B_{\max}) \tag{3.3}$$

We denote the optimal threshold by $T^*_{j,k} = \arg\max \mathsf{U}_{j,k}$. However, when we use percentile detector instead of utility function, we compute a composite distribution of the end-hosts' traffic within a group and then compute 99^{th} percentile threshold of the composite distribution.

Algorithm 1 Distributed algorithm executed in i^{th} end-host

```
ComputeThresholdLocal (utilityFunction, $w$, $\alpha$)
Ensure: $T_i$
1: if utilityFunction is Percentile Detector then
2:     Find $\alpha$ percentile, $T_i$ of $P(G_i)$
3: else
4:     Optimize utility function (specified by $w$) to get optimal threshold, $T_i$
5: end if
```

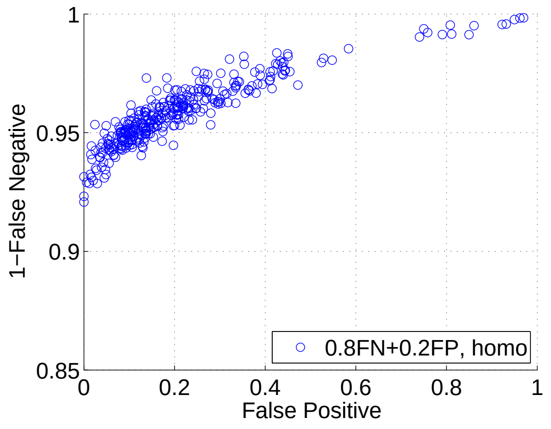
Grouping Policy: There are several ways to form groups. In this thesis, we have experimented two ways of grouping the end-hosts. In both the approaches, we have used 99^{th} percentile values as profile of the users. The motivation behind the first approach is to group together the users which have similar tail values. For $k = 2$, we put all the end-hosts whose 99^{th} percentile values are in the bottom 85% in one group and the remaining form the heavy end-hosts group. For $k = 3$, we first find 2 groups (as above) and then find another group which comprises of the bottom 15% of the end-hosts. Thus, the middle 70% form the second group and upper 15% form the third group. For $k > 3$, we first find 3 groups as we do for $k = 3$. Then, we sub-divide these three groups (with heavy users first) until k groups are formed. For example, for $k = 5$, we create three groups first. Then, we bisect the heavy and the middle groups.

We have also tried the k-means clustering technique on the 99^{th} percentile values as k-means clustering is widely used in practice. However, we find that this clustering technique does not perform as well as the above mentioned approach.

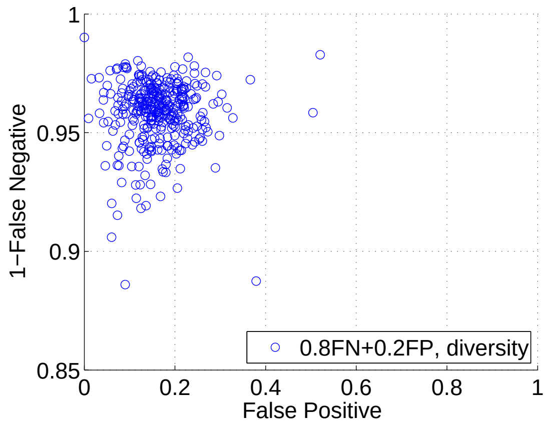
Algorithm for computing thresholds: Having described the k-level policy and grouping policy, we summarize our approach in terms of two algorithms: (i) ComputeThresholds() is executed at the SOC to compute thresholds under homogeneous or k-level diversity policies (see Algorithm); and (ii) ComputeThresholdLocal() is executed by an end-host to find its threshold (see Algorithm). An enterprise policy guides the IT operators to decide how to compute the thresholds. If IT operators opt for a distributed approach, it informs the end-hosts to compute their thresholds locally (Line 2). Otherwise, the centralized algorithm at SOC computes the thresholds for the end-hosts and downloads the configuration on the end-hosts. In this paper, we have used a specific grouping policy but one can define any grouping policy through the function FormKGroups() (Line 15).

Algorithm 2 Algorithm run centrally at Security Operations Center (SOC)

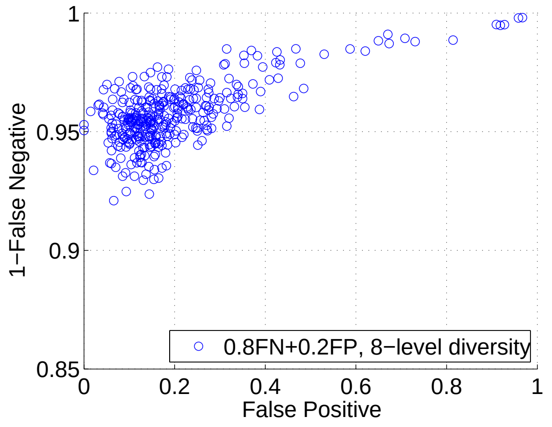
```
ComputeThresholds (policy, utilityFunction, $w$, $\alpha$, $k$)
Ensure: $T_i$ or $T$
 1: if policy is diversity then
 2:     Inform end-hosts to execute ComputeThresholdLocal()
 3: else if policy is Homogeneous then
 4:     Fetch $P(G_i)$ from the end-hosts
 5:     if utilityFunction is PercentileDetector then
 6:         Form a composite distribution, $P(G)$
 7:         Find $\alpha$ percentile threshold, $T^*$ of $P(G)$
 8:     else
 9:         Form a composite utility, $U$=-[$w\mathsf{FN}_i + (1-w)\mathsf{FP}_i$]
10:         Optimize $U$ to get optimal threshold, $T^*$
11:     end if
12:     Distribute $T^*$ to the end-hosts
13: else if policy is $k$-level partial diversity then
14:     Fetch $P(G_i)$ from the end-hosts
15:     FormKGroups(utilityFunction)
16:     for group 1 to $k$ do
17:         if utilityFunction is PercentileDetector then
18:             Form a composite distribution of $j^{th}$ group, $P(G^j)$
19:             Find $\alpha$ percentile threshold, $T^*_{j,k}$ of $P(G^j)$
20:         else
21:             Form a composite utility function, Eq. 3.3
22:             Find $T^*_{j,k}$ from Eq. 3.3
23:         end if
24:         Distribute $T^j$ to the end-hosts in the $j^{th}$ group
25:     end for
26: end if
```



(a) $0.8\mathsf{FN}_i + 0.2\mathsf{FP}_i$, Homogeneous



(b) $0.8\mathsf{FN}_i + 0.2\mathsf{FP}_i$, Diversity



(c) $0.8\mathsf{FN}_i + 0.2\mathsf{FP}_i$, 8-level partial diversity

Figure 3.6: EPC using utility function, $0.8\mathsf{FN}_i + 0.2\mathsf{FP}_i$**; (a) high detection rate but high** FP_i **under homogeneous policy; (b) high detection rate but bounded** FP_i **under diversity; (c) improved** FP_i **compared to homogeneous policy**

3.5 Evaluation

We first present the evaluation of our new policies using simulated attacks as that in Section 3.3. In the second part, we use real attack traces to evaluate and demonstrate the effectiveness of our new policies against real attacks. We consider the feature, `number_of_TCP_connecti`
for $P(g_i)$ and the feature, `number_of_distinct_connections` for attack traffic.

3.5.1 Simulations

First, we evaluate the performance of k-level partial diversity policy using percentile detectors. Then, we consider utility functions rather than percentile detectors.

Partial Diversity: First, we illustrate the performance of k-level policy. We have tried a set of different k values. Among them we find that $k < 10$, gives the results which are sufficiently close to full diversity case. We illustrate using $k = 8$ here (We realize that the value of k depends on the datasets). We have already seen the EPC plot comparing homogeneous and diversity policies in Figure 3.3 (a). Comparison between diversity and 2-level policies in terms of EPC is shown in Figure 3.3 (b) where we observe that the detection rate of the light users increases to 0.8 whereas that of the heavy users drops to 0.33 (as compared to homogeneous policy). The benefit of the 2-level policy is that it allows a reduction in the number of users with severely high FP_i, without compromising high detections rates for the bulk of users (unlike the homogeneous policy that induces this trade-off). EPC plot for the 8-level approach is shown in Figure 3.3 (c). We observe that except for a few end-hosts in groups 6, 7 and 8, all other end-hosts experience same FP_i and FN_i as in diversity approach. As we form more groups, the number of end-hosts in each group becomes lesser and the diversity thresholds tends to become equal to the group threshold (homogeneous within a group). The potential of 8-level policy shows that IT operators do not need to compute individual thresholds for all 350 end-hosts. However, we do realize that this level of diversity depends on the inherent clusters existing among the users' behaviors.

Attack Sizes: Next, we evaluate the performance based on the attack sizes chosen by the attacker (see Section 3.3). Figure 3.4 (b)-(c) show the attack sizes under diversity and k-level policy. Under 2-level policy, most of the attack sizes are between 100-200 (compared to 200-300 in Figure 3.4 (a)). However, under the 2-level policy, the heavy end-hosts are penalized as they have to set larger thresholds. In the 2-level policy, the end-hosts in different groups do not interact in setting the thresholds and therefore, the heavy end-hosts are not constrained in choosing large thresholds, unlike in the homogeneous policy. However, under the 8-level policy, the end-hosts experience similar performance as that in diversity policy. The effectiveness of diversity and 8-level policies in reducing the total malicious traffic can be seen in Table 3.1.

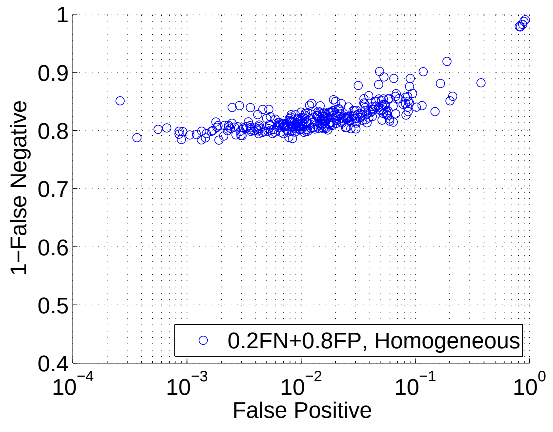
The FP/FN Balance: Here, we illustrate the performance of the end-hosts when they use the utility functions $0.8\mathsf{FN}_i + 0.2\mathsf{FP}_i$ and $0.2\mathsf{FN}_i + 0.8\mathsf{FP}_i$ where FN_i and FP_i are computed at the optimal threshold values T_i^*.

The EPC plots corresponding to utility function $0.8\mathsf{FN}_i + 0.2\mathsf{FP}_i$ in Figure 3.6 show that most of the end-hosts experience high detection rate under all policies ($>$ 0.92 under homogeneous and 8-level policies and $>$ 0.86 under diversity). This is expected as false negative component is considered costly, which tend to keep the optimal thresholds small. Improvement on detection rate, results in extremely high false positive rate (between 0 and 1 range under homogeneous and 8-level policies). Specially, the “heavy” users tend to have false positive rate close to 1. Although diversity policy outperforms others by bounding FP_i within 0.4 for most users, the majority of the users experience false positive around 0.2 which is still high for practical purpose. We believe that this utility function can be useful if the IT operators have additional resource to handle the false alarms.

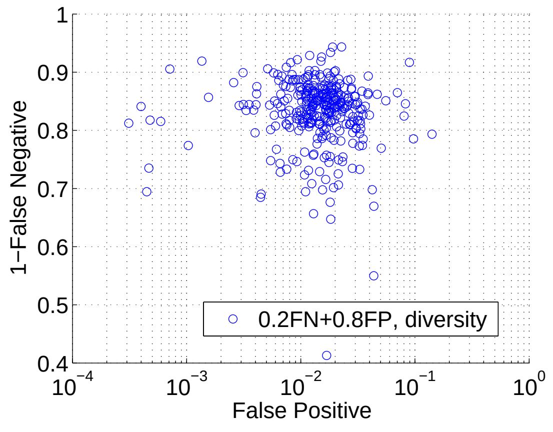
The EPC plots corresponding to utility function $0.2\mathsf{FN}_i + 0.8\mathsf{FP}_i$ in Figure 3.7 show different trade-offs. As shown in Figure 3.7 (a), the detection rate under the homogeneous policy is more than 0.8 for most of the end-hosts, and FP_i is restricted within 0.1 except for some heavy end-hosts. We have seen similar trade-offs in Figure 3.3 (a). For diversity approach in Figure 3.7 (b), FP_i of most of the end-hosts is less than 0.02 and detection rate of them are above 0.7, which indicate that benefits of diversity shows up when the utility function favors false positives. The effectiveness of the 8-level policy shows up in bounded FP_i (within 0.1), but the detection rate of some end-hosts reduces to 0.4 (Figure 3.7 (c)).

Attack Sizes: Figure 3.9 (a)-(b) and (c)-(d) show the attack sizes chosen by an attacker (as that in Section 3.3) when the utility functions are $0.8\mathsf{FN}_i + 0.2\mathsf{FP}_i$ and $0.2\mathsf{FN}_i + 0.8\mathsf{FP}_i$, respectively. In Figure 3.9 (a), more circles below the line x=y indicates that more end-hosts yield to higher attack sizes. The effectiveness of the diversity is due to smaller thresholds for bulk of the users. Figure 3.9 (b) shows that comparatively more circles are above the line $x = y$, implying a similarity with diversity approach. Similar comparison with the utility function $0.2\mathsf{FN}_i + 0.8\mathsf{FP}_i$ are shown in Fig. 3.9 (c)-(d).

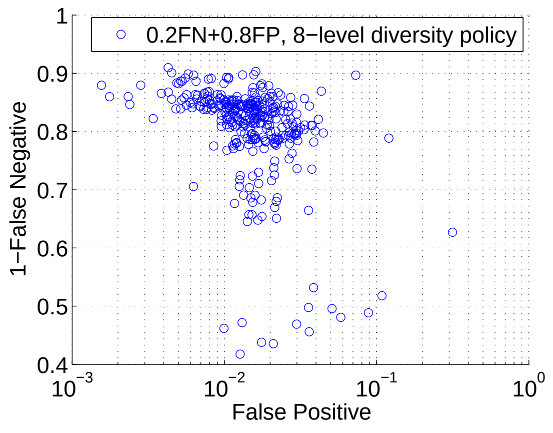
In order to compare the cumulative attack traffic, we compute $\sum_i b_i$ under different policies and summarize the results in Table 3.1. The damage under the diversity policy is smaller than that under the homogeneous policy. Although the diversity policy outperforms others consistently, the damage under 8-level policy can be worse than the homogeneous policy as some of the heavy end-hosts have very large thresholds under the 8-level policy resulting in



(a) $0.2\mathsf{FN}_i + 0.8\mathsf{FP}_i$, Homo



(b) $0.2\mathsf{FN}_i + 0.8\mathsf{FP}_i$, Diversity



(c) $0.2\mathsf{FN}_i + 0.8\mathsf{FP}_i$, 8-level partial diversity

Figure 3.7: EPC using the utility function, $0.2\mathsf{FN}_i + 0.8\mathsf{FP}_i$; (a) high detection rate and high FP_i under homogeneous policy; (b) low and bounded FP_i and high detection rate under diversity policy; (c) improved FP_i performance (but low detection rate for some) under 8-level policy.

large attack sizes. Both EPC and damage results show that diversity benefits are significant when false positive component is favored.

utility	Diversity	Homogeneous	8-level Policy
Percentile	69839	95456	61961
$0.8\mathsf{FN}_i + 0.2\mathsf{FP}_i$	242.2	1728.1	808.6
$0.2\mathsf{FN}_i + 0.8\mathsf{FP}_i$	34243	39825	43365

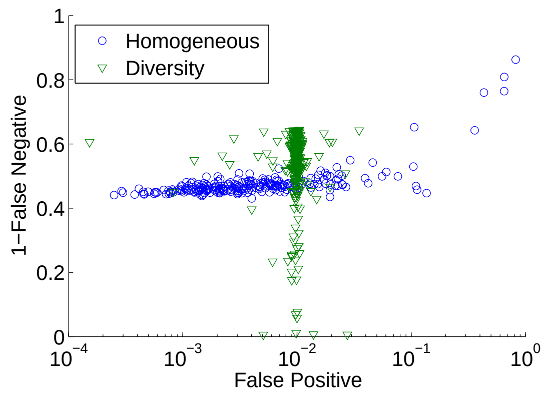
Table 3.1: Damage $= \sum_i b_i$. The diversity policy outperforms the homogeneous policy.

3.5.2 Evaluation with Real Attack Traces

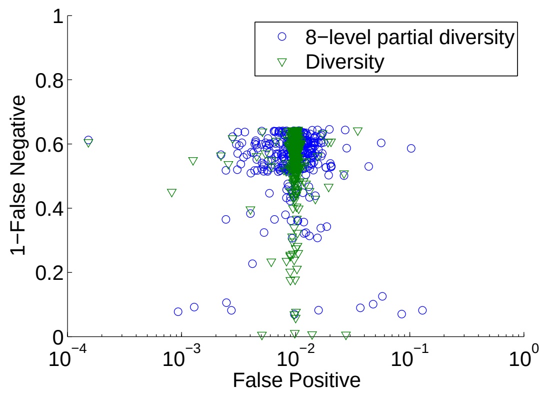
In this section, we show the evaluation of different policies using real attack traces. In order to collect malicious traces, we installed malicious binaries of SDBot and Storm bot, each on a separate laptop which did not have other applications running. Thus, all the traffic collected originated from the corresponding malware. Due to lack of space, we only show the results using Storm Bot trace. In the Storm trace, we find 319891 TCP connections and 12238 UDP connections. The Storm bot is aggressive in opening new connections which are mostly SMTP flows (port 25 SPAM). The typical number of connections opened during a typical 5 minute period is in the range 1 to 2000. During a few intervals, the number increases to 5000.

Using `distinct_IP_destination addresses` as the feature, we obtain the time-series of the malicious trace. Subsequently, we superimpose the malicious time-series on the normal traffic time-series. FN_i count denotes the fraction of time bins when traffic count is below threshold and traffic count consists of malicious flows.

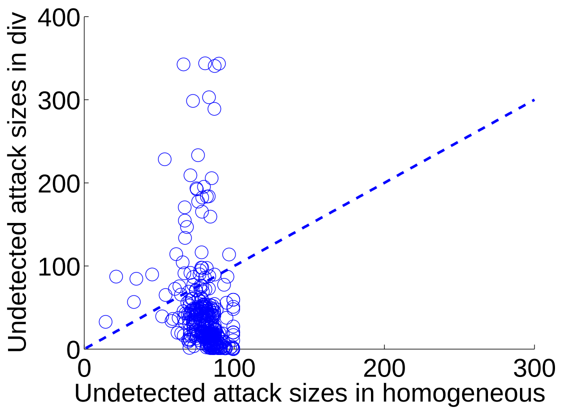
Policy Comparison: First, we show the performance of percentile detector on real attack traces. Figure 3.8 (a) reveals similar trends as we observe with synthetic attacks (in Figure 3.3, Section 3.3), but the numbers are shifted. Under the diversity policy the best detection rate is around 0.62. Under the homogeneous policy, the detection rate of most



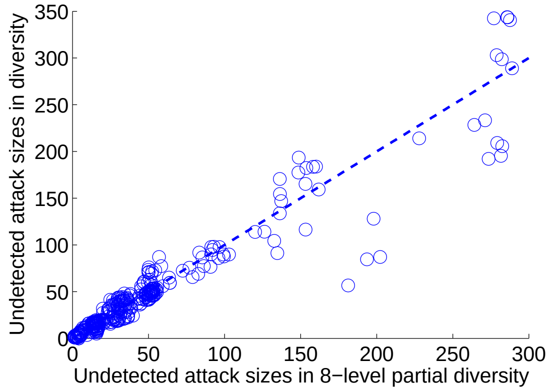
(a) Homogeneous vs. Diversity



(b) 8-level vs. Diversity



(c) Homogeneous vs. Diversity



(d) 8-level vs. Diversity

Figure 3.8: (a)-(b) EPC with real attacks replayed on normal traffic using 99^{th} percentile detector. (a)-(b) bulk of users under homogeneous policy have similar detection rate and their false positive rate is improved under 8-level policy; (c) attack sizes under homogeneous policy are larger; (d) 8-level policy is similar to diversity

end-hosts is slightly more than 0.4, with FP$_i$ spreading between 0 and 1.

Figure 3.8 (c)-(d) show the attack sizes that are not detected by the end-hosts when we superimpose malicious traffic on normal traffic. Each point in Figure 3.8 (c)-(d) represents an end-host's average undetected attack size. Undetected attack size, $b_i > 0$ in an interval is such that $b_i + g_i < T_i$. For each end-host, we compute the average of all b_i varying the locations of super impositions. Similar performance between the diversity and 8-level policy shows the impact of grouping criteria on the results (percentiles used in clustering).

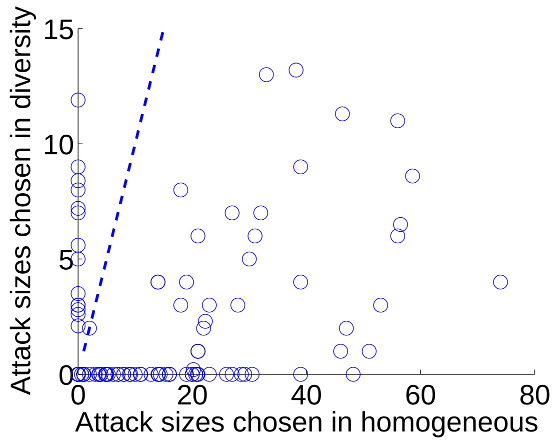
The attack sizes under the 8-level policy are similar to that in the diversity policy and almost all the attack sizes fall on the line $x = y$.

Utility Functions: Here, we present the performance using utility functions. Figure 3.10 (a)-(b) show the EPC using $0.8\mathsf{FN}_i + 0.2\mathsf{FP}_i$. The false negative rate of most end-hosts have an inversely proportional relationship with the false positive rate. Heavy users experience high detection rate at the cost of high false positive rate. The minimum detection rate we observe is 0.65 which is close to the highest detection rate using percentile detector. Here, the thresholds are small and there is not much difference in the performance between homogeneous and diversity approach. Figure 3.10 (c)-(d) show the undetected attacks under different approaches, and we see that the attack sizes are smaller than 3 under all approaches.

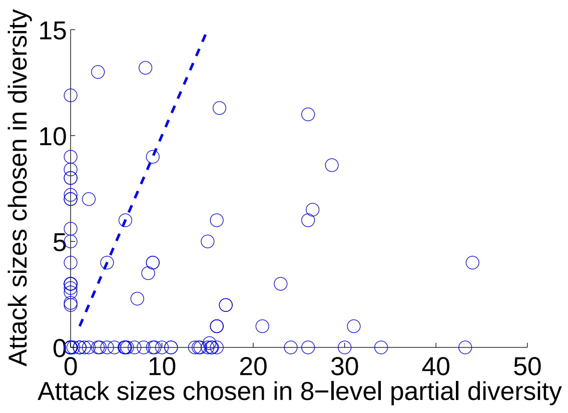
utility	diversity	homogeneous	8-level
Percentile	15949	27775	16351
$0.8\mathsf{FN}_i + 0.2\mathsf{FP}_i$	414.03	238.19	224.12
$0.2\mathsf{FN}_i + 0.8\mathsf{FP}_i$	3679.2	6600.4	10140

Table 3.2: Damage when real attack traces are used. Significant reduction in damage under diversity using the percentile detector and $0.2\mathsf{FN}_i + 0.8\mathsf{FP}_i$.

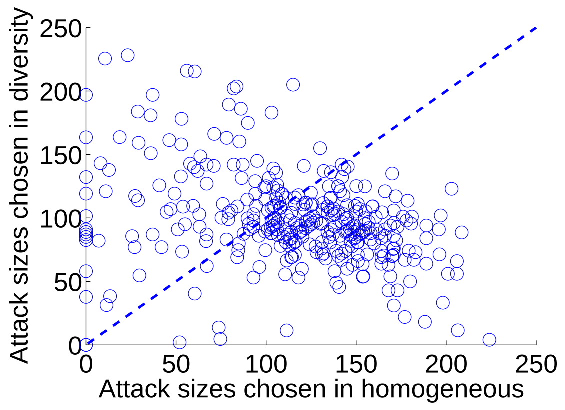
Figure 3.11 (a)-(b) show the EPC when the end-hosts use the utility function $0.2\mathsf{FN}_i + 0.8\mathsf{FP}_i$. The FP_i under all policies are bounded within 0.2 and the best detection rate is around 0.62. Comparing with the EPC of the percentile detector, we find that detection rate of most end-hosts under diversity approach has increased, but at the cost of increased FP_i by a factor of 10 or 20. The 8-level approach helps some of the heavy users in improving their false positive rate. Undetected attack size plots in Figure 3.11 (c)-(d) show similar trends as in that previous experiments. Table 3.2 shows the total damage under different utility



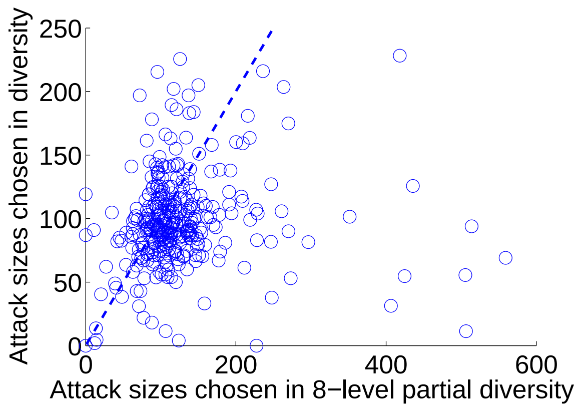
(a) $0.8\text{FN}_i + 0.2\text{FP}_i$, diversity



(b) $0.8\text{FN}_i + 0.2\text{FP}_i$, 8-level



(c) $0.2\text{FN}_i + 0.8\text{FP}_i$, diversity



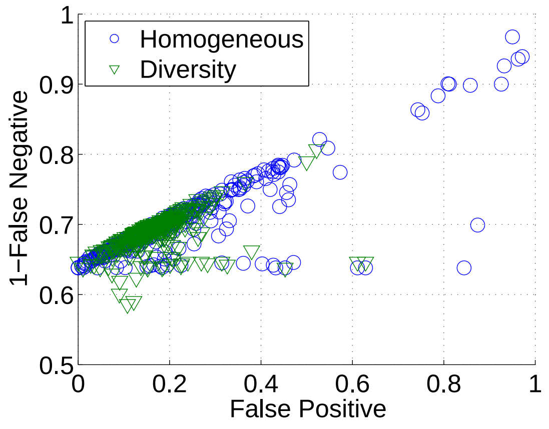
(d) $0.2\text{FN}_i + 0.8\text{FP}_i$, 8-level

Figure 3.9: Attack values chosen by attacker: (a)-(b) attack sizes under homogeneous policy are larger than under diversity policy using $0.8\text{FN}_i + 0.2\text{FP}_i$. 8-level policy improves upon homogeneous policy. Similar observation in (c)-(d) using $0.2\text{FN}_i + 0.8\text{FP}_i$.

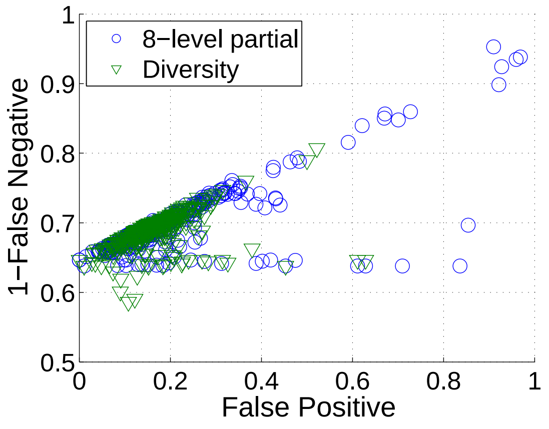
functions and policies.

3.5.3 Summary

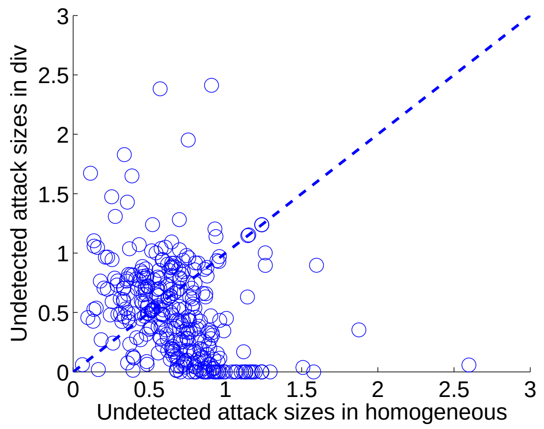
In this chapter we investigate methods of configuring thresholds for statistical anomaly detectors deployed across all end hosts in an enterprise network. HIDS usually have a statistical component which keeps track of some traffic features and set a threshold to define normal traffic profile. The feature values falling outside the thresholds are deemed to be suspicious.



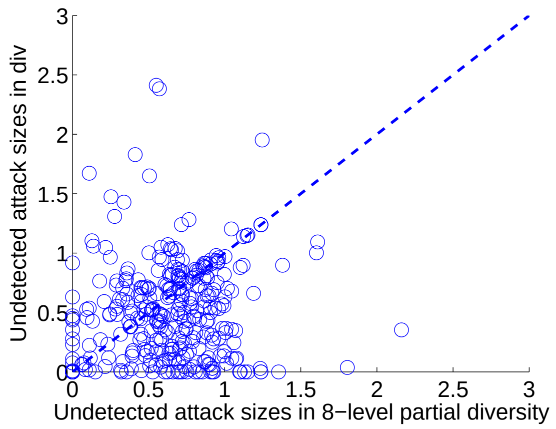
(a) Homogeneous vs. Diversity



(b) 8-level vs. Diversity



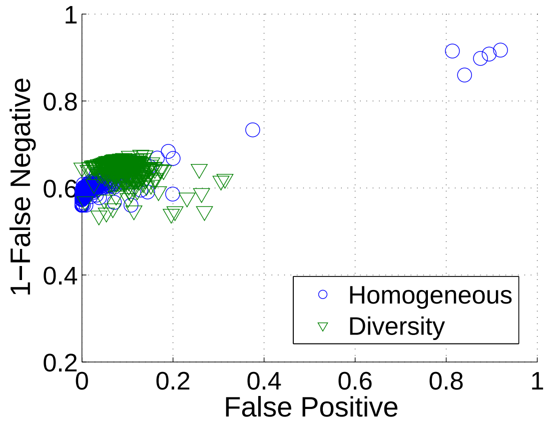
(c) Homogeneous vs. Diversity



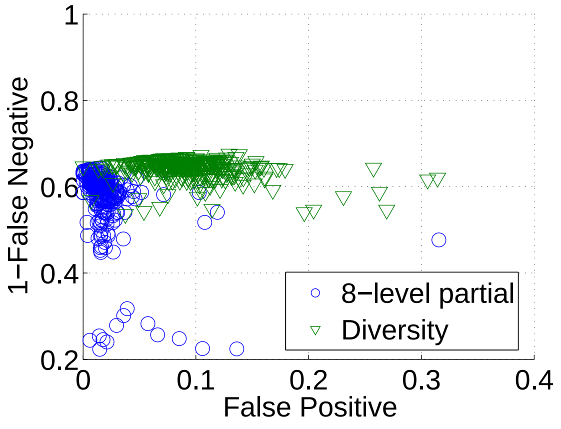
(d) 8-level vs. Diversity

Figure 3.10: (a)-(b) EPC when the end-hosts use $0.8\mathsf{FN}_i + 0.2\mathsf{FP}_i$ and real attacks are replayed; (c)-(d) undetected attack sizes ($<$ 3)

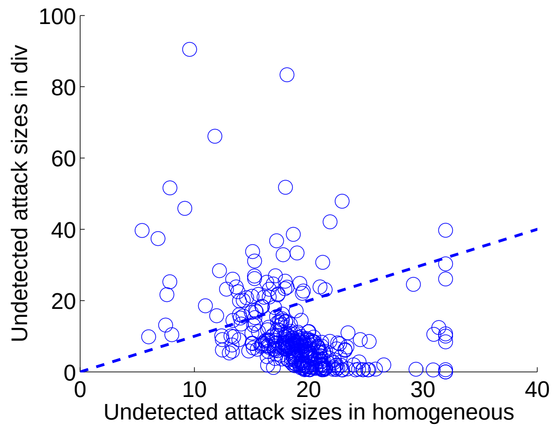
This chapter elucidates using real datasets that the end hosts traffic is very diverse and the goal of setting the thresholds needs to consider several trade-offs that are often overlooked. The chapter concludes that have diversified configuration can improve the detection rate of the end hosts and can limit an attacker to launch only small sized attacks. The chapter also shows that the full benefits of diversified approach can be harnessed even by incurring low management complexity in the form of partial diversity. Finally, the framework offers the flexibility of choosing different trade-offs by proposing grouping policy of the end hosts and



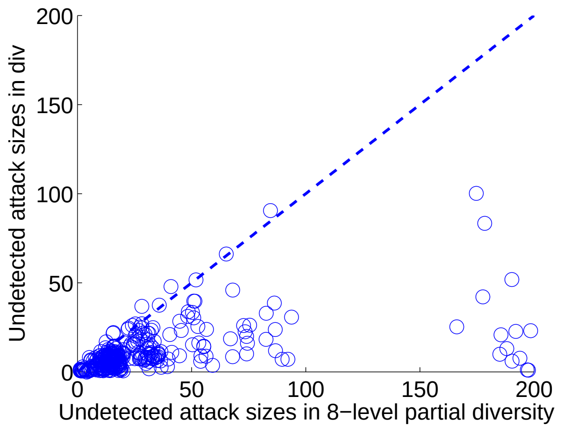
(a) Homogeneous vs. Diversity



(b) 8-level vs. Diversity



(c) Homogeneous vs. Diversity



(d) 8-level vs. Diversity

Figure 3.11: (a)-(b) EPC when the end-hosts use $0.2\mathsf{FN}_i + 0.8\mathsf{FP}_i$ and real attacks are replayed; (c)-(d) undetected attack sizes and effectiveness of diversity policy.

utility function to combine false positive and false negative rates.

Chapter 4

Hierarchical Change Explanation

In this chapter, we formulate the problem of change detection in data with hierarchical identifiers. We discuss several solution approaches and then propose an efficient optimal algorithm. We show the effectiveness of our approach using real datasets.

4.1 Problem Statement

First, we define a natural change explanation model, which expresses the change between the leaf nodes of two hierarchical summaries as a composition of changes top-down from the root to the leaves of the tree.[1] We then discuss the model in the context of Occam's Razor to find a parsimonious explanation of change. Let S be a set of items from a domain D where the elements come from a well-defined hierarchy. Each item $i \in S$ has an associated measure value $v \in V$. The ordered pairs (i, v) could have been obtained by summing over

[1] Our method works for both multiplicative and additive compositions by transforming the former to latter using logarithms; we illustrate using the additive scale.

the (projected) columns in a data warehouse fact table containing a multiset of `(itemID, value)` pairs where `itemID` is a dimension attribute and `value` is a measure attribute. Or they could have been aggregated over some time series window (eg, moving window average). Let T be a rooted tree obtained by inducing the dimension hierarchy on S, where the nodes correspond to different prefixes in the dimension hierarchy. We do not assume a total ordering over the dimension hierarchy, only that it is partially ordered with maximum height h. Let ℓ denote a leaf node and $m(\ell)$ denote some value attached to the leaf node ℓ. Given values attached to leaf nodes that represent some measure of change, we define a class of *hierarchical* change explanation models below.

Definition 4.1.1 Hierarchical Change Explanation: *Given a hierarchy T and change values $m(\ell)$ attached to leaves ℓ, a hierarchical change explanation model is a complete, top-down composition of changes ("weights") $w(n)$ between nodes along the root-to-leaf path, for each leaf node.*

More formally, for each leaf node ℓ,

$$m(\ell) = W(\ell) \tag{4.1}$$

where

$$W(root) = w(root) \tag{4.2}$$

$$W(n) = w(n) + W(p(n)) \tag{4.3}$$

for tree nodes n where $p(n)$ is the parent node of n. A solution to this system gives weights $w(n)$ for each n. In fact, if $\mathcal{P}(n)$ denotes the ancestor path from the root down to a tree node n, then by unraveling Equations 4.1-4.3, our problem is to find weights $w(n)$ of each node n in the tree subject to the constraints $m(\ell) = \sum_{n \in \mathcal{P}(\ell)} w(n)$. Since this system of equations is under-specified, there are multiple solutions each of which provides a hierarchical change explanation.

In general, the change values $m(\ell)$ are obtained as some discrepancy measure $d(m_1(\ell), m_2(\ell))$ between two sets of values observed for the hierarchy T. For example, consider the Census dataset [28] where we have population counts $m_1(.)$ and $m_2(.)$ for zip codes and a geographical hierarchy that defines aggregations at state, county and city levels at two different snapshots T_1 and T_2 as exemplified in Figures 4.1(a) and (b). Here, $m(\ell) = d(m_1(\ell), m_2(\ell))= log(m_2(\ell)/m_1(\ell))$. In general statistical anomaly detection problems, $m_1(\ell)$ is forecasted value based on some statistical model that captures normative behavior and $m_2(\ell)$ is the actual observed value with higher discrepancy being indicative of anomalous behavior.

4.1.1 Parsimonious Explanation

Definition 4.1.1 provides a rich class of hierarchical change explanation models; we provide a couple of examples that are trivial to compute but sub-optimal and then provide a notion of an optimal or parsimonious hierarchical change explanation model.

Non-Hierarchical Approach: One possible assignment of weights that is used in anomaly detection applications is the one that completely ignores the hierarchical structure and assigns

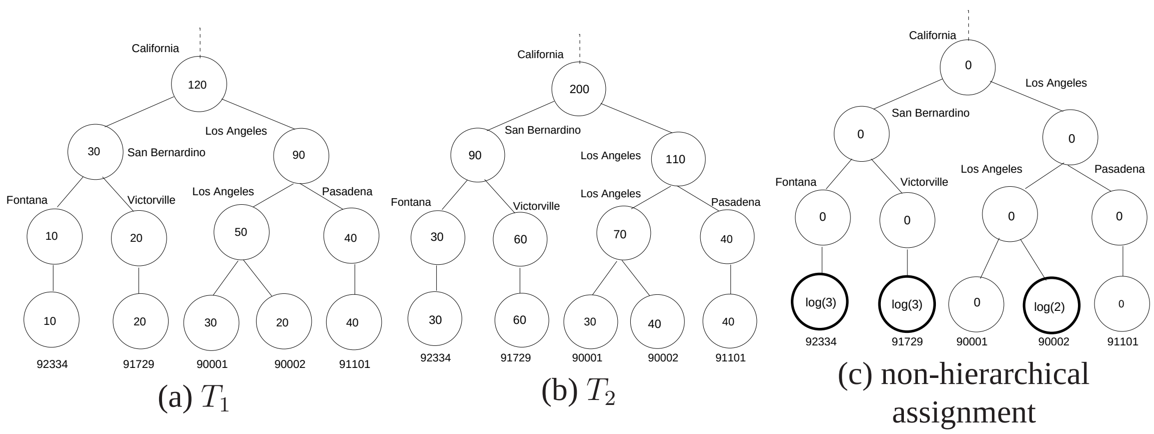


Figure 4.1: Each distinct ZIP code appears as a leaf in the tree along with its associated population counts (shown inside the node). The population count for internal nodes is the sum of the population counts from the leaves in its subtree. (a) and (b) as trees represent two snapshots T_1 and T_2, of hierarchy T respectively; (c) shows a non-hierarchical weight assignment

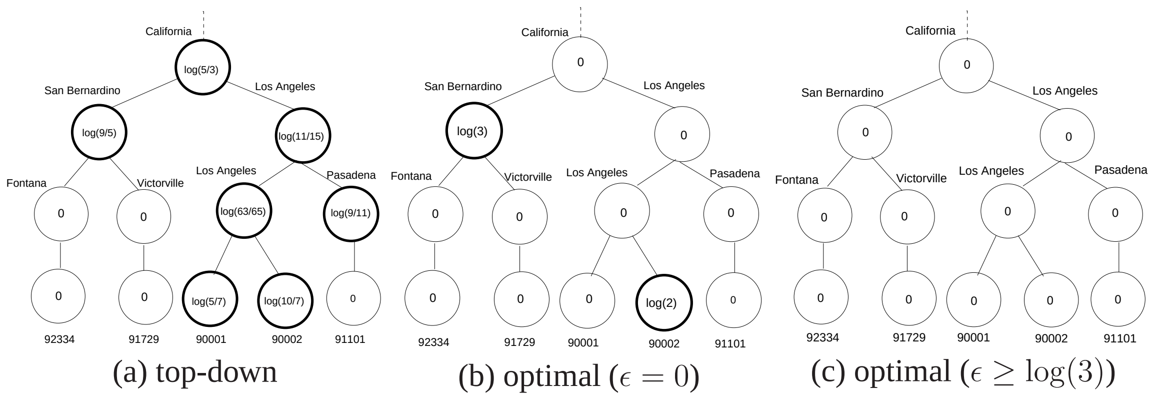


Figure 4.2: Weight assignment based on top-down and optimal assignments.

each leaf node ℓ in T a weight of $m(\ell)$, and 0 to the non-leaf nodes. We call this the "non-hierarchical" model, comparison w.r.t this model helps in quantifying the gain achieved by using the hierarchy. Figure 4.1(c) shows a non-hierarchical assignment for the trees T_1 and T_2 shown in Figures 4.1(a) and (b), respectively. The leaf-level nodes encircled boldly have non-zero assigned weights. Using trees T_1 and T_2, we construct a third tree as in Figure 4.1(c) such that the value associated with a leaf is $\log$ of the ratio of the correspoding leaf counts.

Top-Down Hierarchical Approach: Assuming the existence of a rollup operator that aggregates values of children to the parent, another possible assignment is top-down, which recursively assigns weights from the root down such that $m(\ell) = W(n) = \sum_{u \in \mathcal{P}(n)} w(u)$, for all leaves *and* intermediate nodes n. Figure 4.2(a) provides an example where values at each snapshot are rolled up using the sum operator. Both of these assignments satisfy the equations of hierarchical change explanation but are not necessarily parsimonious: the former ignores all opportunities to group leaves with equal differences in the same subtree whereas the latter is too greedy in that it groups unequal leaf differences.

Parsimonious Hierarchical Approach: A node weight $w(n) = 0$ implies no change to node n relative to $p(n)$ and does not need to be reported in an explanation. Thus, the *explanation size* is the number of non-zero weights in the explanation. Applying Occam's Razor, we prefer an explanation of the smallest size. Therefore, we define a *parsimonious explanation* of hierarchical change as one with the smallest explanation size, that is, the minimum number of weights not equal to zero. Consider Figure 4.2(b), which is able to explain the changes using only 2 non-zero weights compared to 3 for the non-hierarchical strategy and 7 for the top-down one; in fact, it is optimal. We describe the algorithms which lead to assignments in Figure 4.2(b)-(c) in Section 4.2.

However, this explanation model has certain shortcomings. Often one wants to compare a snapshot with expected values; large deviations from these values can be reported as anomalies. Statistical forecasting models (e.g., based on moving averages) typically yield confidence intervals based on a supplied confidence level. An important shortcoming of the current model is that it does not work with such a forecasting model because its formulation does not deal with ranges of possible values. In addition, the model is sensitive to noise. Intuitively, we would like to capture similar changes among related leaves which may not have exactly equal differences but are roughly the same. For example, if two sibling leaves have differences of 1.98 and 2.02, we may wish to describe this at the parent using a difference of 2. Since the deviations from this description at the leaves are small (1%), we may tolerate this error as being a good enough approximation to report only significant changes and to avoid overfitting the data. Our original description above, which only allows exact matches, does not allow this.

In order to ameliorate this, we extend the definition to allow a tolerance parameter ϵ on the values of the leaves. We allow weights on the nodes that result in differences of at most ϵ between two leaves in the snapshots. We assume that in practice this tolerance parameter will be provided by the confidence interval of the prediction model, which can be different from leaf to leaf, so the model allows different tolerances $\epsilon(\ell)$ at each ℓ. Specifically, we assign weights such that $|m(\ell) - W(\ell)| \leq \epsilon(\ell)$ for each ℓ, where $W(\ell) = \sum_{u \in \mathcal{P}(\ell)} w(u)$.

To see the connection with a forecasting model, we assume $m(\ell) = d(m_1(\ell), m_2(\ell)) = m_2(\ell) - m_1(\ell)$. In fact, rewriting this equation $m_2(\ell) - (m_1(\ell) + \epsilon(\ell)) \leq W(\ell) \leq m_2(\ell) - (m_1(\ell) - \epsilon(\ell))$ and denoting $m_1(\ell) + \epsilon(\ell)$ and $m_1(\ell) - \epsilon(\ell)$ by $UB(\ell)$ and $LB(\ell)$, respectively, clearly shows how to use output from a forecasting model in our framework. $LB(\ell)$ and $UB(\ell)$ are lower and upper confidence bounds that are obtained from the estimated forecasting distribution. One possibility which works for symmetric distributions is to choose $m_1(\ell)$ as the predicted mean and $\epsilon(\ell)$ to be proportional to the predicted standard deviation, the constant of proportionality depending on the desired coverage of the confidence interval. For instance, a choice of 1.96 under a Gaussian assumption on the statistical distribution of our node values guarantees 95% coverage. In general, our method is agnostic to the particular choice of forecasting model; the only requirement is the availability of $LB(\ell)$ and $UB(\ell)$. This makes it a highly general purpose method with wide applicability in anomaly detection problems involving hierarchical data where changes are expected to be spatially clustered in subregions of the hierarchy.

We now define our parsimonious explanation model, which allocates a tolerance budget along each path that can be distributed among the individual path nodes in any fashion while maintaining the constraint $|m_\ell - \sum_{n \in \mathcal{P}(\ell)} w(n)| \leq \epsilon(\ell)$.

Definition 4.1.2 Hierarchical Parsimonious explanation:
Given a set of leaf changes $m(\ell)$ and a tolerance budget $\epsilon(\ell) \geq 0$ on the total sum of weights along the path to ℓ for all leaves ℓ, a hierarchical parsimonious explanation *of change finds the smallest explanation size, that is, minimum number of node weights $w(n)$ s.t. $w(n) \notin [-k, k]$; $k \geq 0$.*

In definition 4.1.2, for positive tolerances, only $k = 0$ is of interest to us in practice. However, to facilitate comparison with DIFF algorithm citeSarawagi99, extended definition which allows thresholding on positive values of k is necessary as we shall see later in section 4.3.

4.2 Algorithms

In this section, we describe an algorithm to compute optimal weight assignments (that is, minimizing the explanation size), for the problem defined in Section 4.1 (Definition 4.1.2). The algorithm presented here generalizes this problem by allowing any supplied error tolerances for the leaves as well as intermediate nodes of the hierarchy.

The problem with $\epsilon = 0$ is a special case of the following problem: *Given real matrix A and vector b, find x such that $Ax = b$ minimizing the number of non-zero x_i's.* That problem is not only NP-hard, but is not approximable within $2^{\log^{1-\delta} n}$ for any $\delta > 0$ (in polynomial time, assuming NP is not contained in quasi-polynomial time) [20, 23]. For the special case studied here we give a fast, exact algorithm. We first describe the algorithm intuitively, and then present it formally.

The algorithm makes two passes over the tree: the first *bottom-up* and the second *top-down*. In the first pass the algorithm computes a tentative set of "best" incoming partial sums for each node, using dynamic programming. We prove that the best partial sums for a node are those that allow the node to incur no cost for its own weight and, simultaneously, to provide best partial sums for the maximum number of children. This set of best partial sums for a node is a union of closed intervals, at most one for each leaf.

In the second pass the algorithm works down from the root to assign weights. Each node chooses its own weight so as to benefit the maximum number of its children. If the incoming partial sum for a node is one of the best for the node, it can do this without incurring a cost at the node. Otherwise, the node incurs a cost of 1 for its own weight, which it chooses to benefit the maximum number of its children. We illustrate this process in Figure 4.3 for the case $\epsilon = 1$.

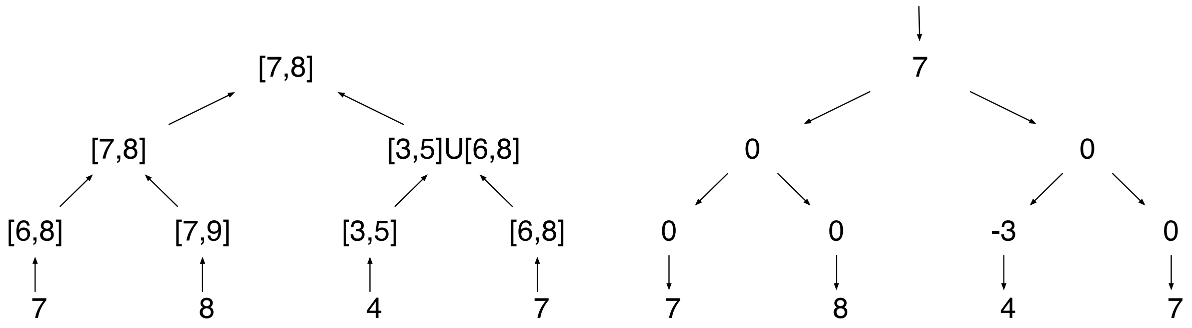


Figure 4.3: Computing an optimal node weighting $(k = 0, \epsilon = 1)$**.**

Note that instead of taking $k > 0$, one may add $k|\mathcal{P}(\ell)|$ to each $\epsilon(\ell)$, then take $k = 0$. This expands the set of feasible weightings. We present the algorithm for the general case $k > 0$ for compatibility with [99], which we shall compare against in Section 4.3.

Definition 4.2.1 *For any subtree T' and real value x, define $\mathrm{cost}(x, T')$ to be the minimum cost of any feasible labeling of T', given that the partial sum coming into the root of T' from above is x. (Formally, this is the minimum cost of any feasible labeling of the tree T' in isolation, where each leaf change $m(\ell)$ has been decreased by x.)*

Define $\mathrm{bestcost}(T') = \min_x \mathrm{cost}(x, T')$ *and* $\mathrm{bestsums}(T') = \{x : \mathrm{cost}(x, T') = \mathrm{bestcost}(T')\}$.

We start with the observation that a bad incoming partial sum increases the cost of T' by only 1. Let $[x \notin S]$ denote 0 (false) if $x \in S$ and 1 (true) otherwise.

Lemma 4.2.2 *For any subtree T' and real x,* $\mathrm{cost}(x, T') = \mathrm{bestcost}(T') + [x \notin \mathrm{bestsums}(T')]$

Proof 4.2.3 *Since the costs are integers, it's enough to prove that* $\mathrm{cost}(x, T') \leq \mathrm{bestcost}(T') + [x \notin \mathrm{bestsums}(T')]$.

Let x' be a partial sum achieving $\mathrm{bestcost}(T')$, and let w be a corresponding min-cost weighting of T' for partial sum x'. Adding $x - x'$ to the weight of the root of T' gives a feasible weighting for T' with partial sum x, and increases the cost of w by at most 1.

Next we prove a recurrence which will be the basis for the algorithm. Let $A \oplus B$ denote $\{a + b : a \in A, b \in B\}$.

Theorem 4.2.4 *Let T' be any subtree with immediate subtrees $T'_1, T'_2, \ldots, T'_c$. Then* $\mathrm{bestsums}(T')$ *equals*

$$[-k, k] \oplus \big\{ z : z \textit{ minimizes } |\{i : z \notin \mathrm{bestsums}(T'_i)\}| \big\}.$$

Proof 4.2.5 *Let* $\mathrm{kiddiff}(z, T')$ *denote* $|\{i : z \notin \mathrm{bestsums}(T'_i)\}|$. *Let* $\mathrm{bestkiddiff}(T')$ *denote* $\min_z \mathrm{kiddiff}(z, T')$. *Fix x and T'. By definition,* $\mathrm{cost}(x, T')$ *equals*

$$\min_y \;\; [y \notin [-k, k]] + \sum_i \mathrm{cost}(x + y, T'_i)$$

(y is the weight given to the root of T'). By lemma 4.2.2, this is

$$\big(\sum_i \mathrm{bestcost}(T'_i)\big) + \min_y \;\; [y \notin [-k, k]] + \mathrm{kiddiff}(x + y, T').$$

The term on the left is independent of x, while the $\min_y \ldots$ term on the right will equal $\mathrm{bestkiddiff}(T')$ *for some x (e.g. when $y = 0$ and x minimizes* $\mathrm{kiddiff}(x, T')$*).*

Thus, $x \in \mathrm{bestsums}(T')$ *(that is,* x *minimizes* $\mathrm{cost}(x, T')$*) iff*

$$\exists y \in [-k, k] : \mathrm{kiddiff}(x + y, T') = \mathrm{bestkiddiff}(T').$$

Taking $z = x + y$*, this condition is equivalent to*

$$x \in [-k, k] \oplus \{z : \mathrm{kiddiff}(z, T') = \mathrm{bestkiddiff}(T')\}.$$

Theorem (4.2.4) gives a recurrence relation for $\mathrm{bestsums}()$. Using this recurrence, computeDS(T, d) uses dynamic programming to compute, for all subtrees T', $\mathrm{bestsums}(T')$ and $\mathrm{kidopt}(T') = \big\{z : z \text{ minimizes } |\{i : x \notin \mathrm{bestsums}(T'_i)\}|\big\}$. The algorithm first calls computeDS(T, d)

Algorithm 3 computeDS(subtree T', leaf values m)

```
1: If $T'$ is a leaf (a single node $\ell$):
2:     let $\mathrm{kidopt}(T') \leftarrow \{m(\ell)\} \oplus [-\epsilon(\ell), \epsilon(\ell)]$
3: else:
4:     for each subtree $T'_i$ of $T'$: computeDS$(T'_i, m)$
5:     $\mathrm{kidopt}(T') \leftarrow \{z$ minimizing $|\{i : x \notin \mathrm{bestsums}(T'_i)\}|\}$
6: $\mathrm{bestsums}(T') \leftarrow \mathrm{kidopt}(T') \oplus [-k, k]$
```

to compute $\mathrm{kidopt}(T')$ and $\mathrm{bestsums}(T')$ for all subtrees T'. It then weights T by calling weightTree$(T, 0)$.

Algorithm 4 weightTree(partial sum x, subtree T')

```
1: if $x \in \mathrm{bestsums}(T')$:
2:     pick $y \in [-k, k]$ s.t. $x + y \in \mathrm{kidopt}(T')$
3: else: let $y \leftarrow x' - x$ for any $x' \in \mathrm{kidopt}(T')$
4: give the root of $T'$ weight $y$
5: for each subtree $T'_i$ of $T'$: weightTree$(x + y, T'_i)$
```

Lemma 4.2.6 *weightTree(x, T') finds a feasible weighting of T' (assuming incoming partial sum x) of optimal cost* $\mathrm{cost}(x, T')$.

Proof 4.2.7 *From Theorem 4.2.4, computeDS correctly computes* kidopt *and* $\mathrm{bestsums}$. *Theorem 4.2.4 assures that y exists in the second line of weightTree(). A standard proof by induction shows that the weighting is feasible.*

To finish we consider the cost. By inspection weightTree(x, T') chooses a root weight y so $x + y \in \mathrm{kidopt}(T')$. *Thus (assuming by induction that the subtrees are weighted optimally), the total weight for nodes in the subtrees $\{T_i'\}$ is* $\mathrm{bestcost}(T')$. *In addition, at the root we pay* $[y \notin [-k, k]]$. *By inspection of weightTree(), this equals* $[x \notin \mathrm{bestsums}(T')]$. *Thus, the total cost of our weighting is* $\mathrm{bestcost}(T') + [x \notin \mathrm{bestsums}(T')]$. *By Lemma 4.2.2, this is best possible.*

Lemma 4.2.8 *The running time of the algorithm is $O(hN \log N)$, where h is the height of the tree and N is the number of leaves.*

Proof 4.2.9 *(Sketch) The running time of the algorithm is dominated by the time it takes to compute the optimal shift for each node. We note that the total size of the optimal shifts for a node is bounded by the number of leaves in the subtree rooted at that node. Computing the optimal shifts of a parent node from the labeling of its children nodes requires sorting and merging of the children node labels.*

For any tree, assume that at depth d there are $c(d)$ nodes. A node v_i at depth d will have $lv(v_i)$ leaves in its subtree, but $\sum_1^{c(d)} lv(v_i) = N$ where N is the number of leaves. This *bounds the size of the labeling at that node. The cost of sorting and merging M nodes is $M \log M$. So the total processing time at level d is given by*

$$\sum_{1}^{c(d)} lv(v_i) \log lv(v_i) \leq N \log N$$

Hence, total time of processing over the entire tree is

$$\sum_{d=0}^{h} N \log N = O(hN \log N)$$

4.3 Evaluation

In this section, we investigate both the effectiveness and the efficiency of the proposed algorithms and the outputs they generate using real data. We evaluate the effectiveness of our proposed change detection model according to its ability to capture interesting hierarchical changes as well as the robustness and stability of the output under small perturbations of error tolerance.

4.3.1 Experimental Setup

We define *stability* to measure the sensitivity of the set of explanation weights as a function of confidence level c. Let S_l^c be the set of nodes at level l where "explanations" occur. Then the stability of the output at level l, given a change in tolerance parameter from $c - \Delta c$ to c, is given by $S_c = \frac{|S_l^{c-\Delta c} \cap S_l^c|}{|S_l^c|}$, where $c - \Delta c$ refers to the previous value of c.

Description of the datasets: We use the following two real data sets: Census, which gives population counts for a geographical hierarchy given by state/county/ city/zip_code [28]; and WorldCup, which is a Web log over a duration of several months of URL accesses to files having a maximum path length of 7 [112]. Note that the hierarchy induced by the URL file paths are not homogeneous, that is, the nodes have different fanouts and the paths have different depths. The Census data has approximately 81,000 leaf nodes and 130,000 total nodes in the tree. The maximum height of the tree is 5 (including the root which stands for the whole country). The World Cup datasets have about 4300 leaf nodes and around 4500 total number of nodes. In the non-homogeneous World Cup datasets, the maximum height of the tree is 8 including the root.

All experiments were run on a Pentium(R) machine with 4 CPU and clock speed 2.66GHz. Table 4.1 summarizes some statistics of these data sets.

Trace	# Leaves	# Nodes	Max Depth
CensusAvg-2004	81174	130585	5
Census 2000 vs. 2004	81129	130477	5
Census 2001 vs. 2004	81093	130478	5
Census 2002 vs. 2004	81161	130522	5
Census 2003 vs. 2004	81182	130551	5

Table 4.1: Census dataset statistics

4.3.2 Forecasting Model

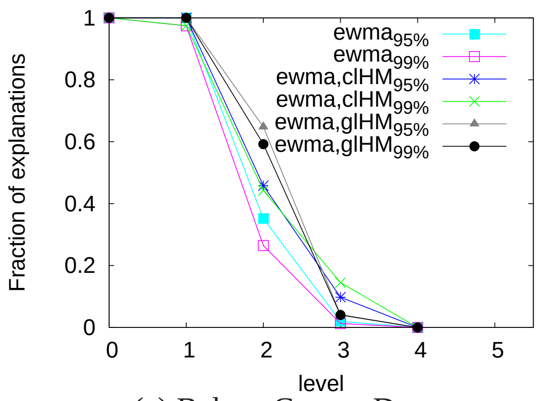
We provide a description of the models that we use in our experimental evaluation on real data. We do not claim any novelty here and use the popular exponentially weighted moving average (EWMA) for both our datasets. In our analysis, we assume a Gaussian distribution

EWMA		EWMA with Cluster Harmonic Mean of Variances	
Node	Weight	Node	Weight
Illinois/Lake/LibertyVille/27923	1.101	Illinois/Lake/66027/27923	1.066
Texas/Parker/FortWorth	1.005	Texas/Parker/FortWorth	1.005
Illinois/St. Clair/47423/69550	0.870	Illinois/Kankakee/Bourbonnais/46513	0.815
Minnesota/Le Seur/0/Mankato City/	0.868	Texas/Hays/Austin	0.655
Texas/Hays/Austin	0.655	Illinois/Lake/LibertyVille/27923	0.565

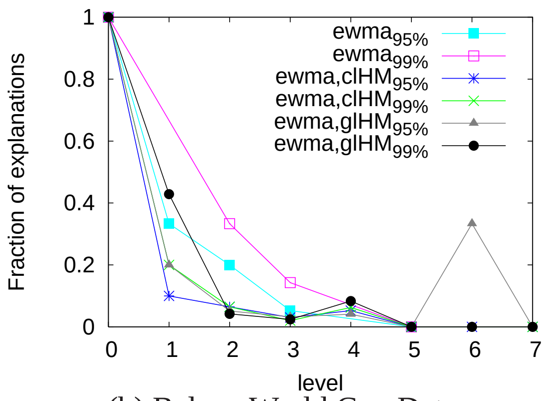
Table 4.2: Top 5 explanation nodes in the Census data sets in the descending order of relative error using the prediction model for 95% confidence values

EWMA		EWMA with Harmonic Mean of Cluster Variances	
Node	Weight	Node	Weight
/images/jersey_arg_res.gif	41.052	/english/playing/download /download_memo.html	42.925
/images/jersey_arg_off.gif	35.875	/images/f98dnld_memo_sc2.gif	40.236
/images/jersey_nor_off.gif	32.286	/images/f98dnld_memo_sc3.gif	39.713
/images/jersey_nor_res.gif	32.262	/images/f98dnld_memo_sc1.gif	39.396
/images/11189.jpg	10.019	/images/f98dnld_memo_sc5.gif	38.878

Table 4.3: Top 5 explanation nodes in the World cup datasets in the descending order of relative error using the prediction model for 95% confidence values



(a) Below, Census Data



(b) Below, World Cup Data

Figure 4.4: The fraction of explanation nodes at level l which have ancestors and descendants in the explanation, for Census and World Cup data. clHM refers to Harmonic Mean of Cluster Variances; glHM refers to Harmonic Mean of Global Variances; subscripts denote the confidence values.

for the node values. Although this may not be a reasonable assumption for count data on the original scale, it is often a good approximation on a transformed scale (log and squared-root are widely used for count data). For our example datasets, we consider an exponentially weighted moving average (EWMA) to model the transformed leaf counts. We use a single smoothing parameter for all our leaf nodes, the value being selected to minimize the average predictive squared-error loss on a tuning set across all nodes. We assume there is no seasonality in our time series. This is the case for both the data sets analyzed in this paper. Consider a single leaf node and let $\hat{x}_t$ denote predicted value at time t based on data until time $t-1$. For EWMA, $\hat{x}_t = m_{t-1}$; and

$$\hat{m_t} = \lambda x_t + (1-\lambda)m_{t-1}; \tag{4.4}$$

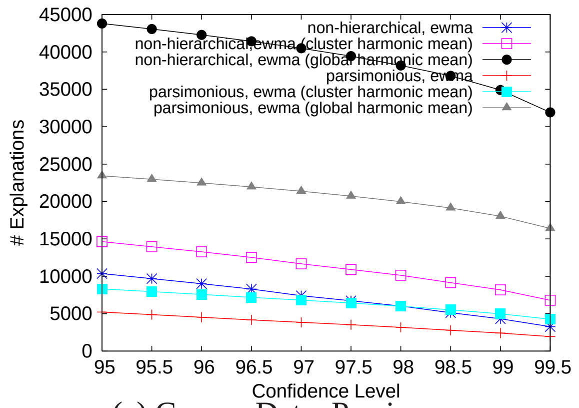
where $\lambda \in (0, 1)$ is a smoothing constant with higher values giving less weight to historical observations. Equation 4.4 can be shown to be a steady state model obtained form a simple random walk model given by

$$x_t = m_t + \epsilon_t \tag{4.5}$$

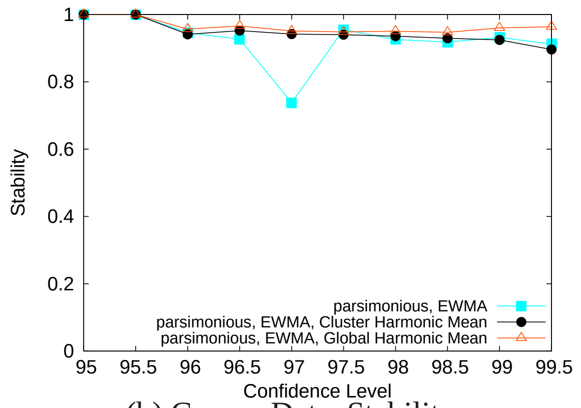
$$m_t = m_{t-1} + \gamma_t \tag{4.6}$$

where x_t is observed value at time t, m_t may be thought of as the truth, ϵ_t and γ_t are uncorrelated random variables with zero means and variances $V(\epsilon)$ and $V(\gamma)$ [92]. At steady state, the optimal prediction obtained through Equation 4.5 reduces to Equation 4.4 with an optimal value of λ given by $(\sqrt{(1 + 4R)} - 1)/2R$; $R = V(\epsilon)/V(\gamma)$ and the predictive variance at time t based on data up to time $t - 1$ is given as $V = V(\epsilon)/(1 - \lambda)$. Thus, estimators which give more weight to historical data achieve more smoothing and have lower predictive variance.

In our scenario, we are dealing with $N > 1$ time series corresponding to the leaf nodes giving rise to pairs $(\lambda_i, V_i(\epsilon))$ to be estimated. For simplicity, we assume $\lambda_i = \lambda$ for all i and estimate the optimal value by minimizing squared-error predictive loss on a tuning set (see [106] for an example of such an estimator). For $V_i(\epsilon)$, we test the following variations: a) separate parameter for each series; b) one parameter for a set of sibling leaf nodes (nodes sharing same parent); c) one parameter for all the time series. We select the best model as



(a) Census Data, Parsimony



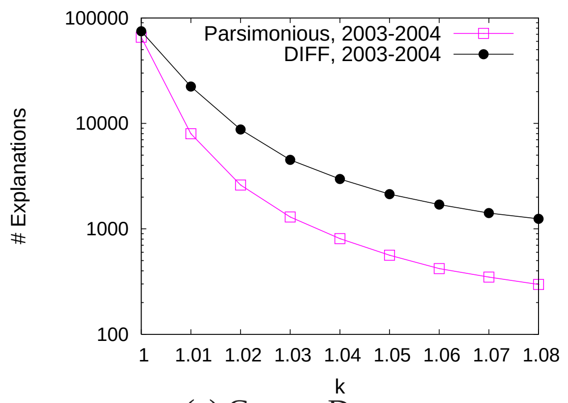
(b) Census Data, Stability

Figure 4.5: (a) Number of explanations; and (b) Stability as function of confidence value for non-hierarchical and parsimonious algorithm on Census data

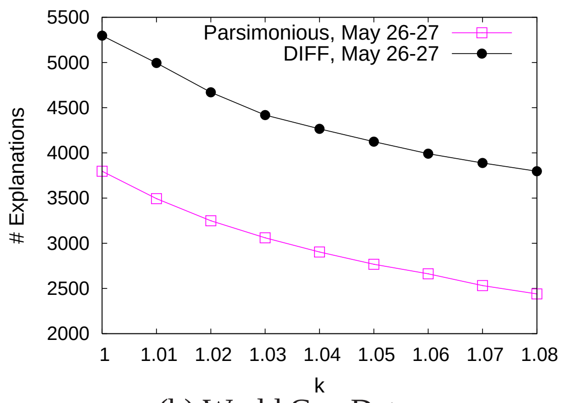
the one that minimizes average predictive log-likelihood on the tuning set; this captures both the mean and variance properties of the predictive distribution.

We analyze two datasets: Census and WorldCup. The Census data have yearly population numbers from $2000 - 2004$. We used $2000 - 2003$ as our training period and 2004 as our test period on which we detect anomalies. We have approximately 81,000 leaf nodes on the test period (see Table 4.1). Since all number are positive (a count of 0 is interpreted as missing data), we model the data using a EWMA of 4 time points on the log scale.

We consider daily counts for the World Cup data and use 32 time points. The 31^{st} time point is used as a tuning set to select the smoothing parameter λ and the variances. The last time point is used as our test set. The optimal value of λ in this case is 0.8 and the model with separate variances for each node also turns out to be the best one for this data. Unlike the Census data, the World Cup hierarchy is not homogeneous, i.e., the nodes have different fanouts and paths have different depths. Also, the structure of the tree is dynamic with new



(a) Census Data



(b) World Cup Data

Figure 4.6: Comparison with related work DIFF operator in terms of number of explanations on (a) Census data (b) World Cup data. k is per-node tolerance and is in linear-scale (not log).

nodes appearing and some old nodes becoming inactive over time. We restrict ourselves only to nodes that occurred at least twice in the last 10 time points. This removes nodes that have a small mean and are not of interest, providing a set of approximately $5.5K$ leaf nodes to be monitored. Since zero counts are common in these time series, a log transform to achieve symmetry is not an option here. Instead, we use a squared-root transformation which, for count data, is known to stabilize variance, achieve approximate symmetry and makes the assumption of a Gaussian distribution reasonable.

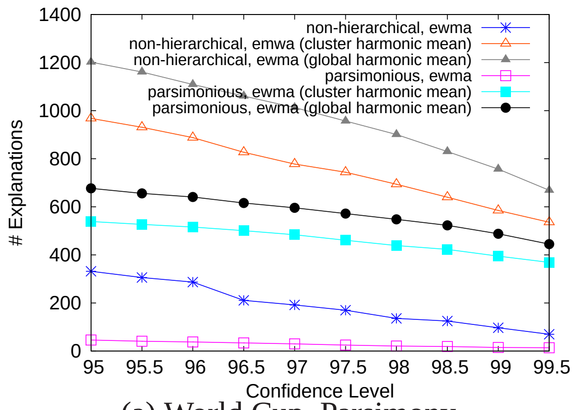
4.3.3 Goodness of Explanation Model

For illustrative purposes, we present the top 5 nodes in resulting explanations based on absolute magnitude (difference of the weights from 0). Table 4.2 shows the 5 nodes in the Census datasets which are explanations and whose absolute relative error is among the Top 5. We show the lists for two different prediction models: a superior EWMA model, with a separate variance component for each leaf and an inferior EWMA model with a separate but fixed variance for each leaf in a cluster (nodes under same parent) which is set to the harmonic mean of the individual leaf variances in the cluster. Note that some nodes are common explanation nodes under both the models such as Illinois/Lake/Libertyville/27923, Texas/Parker/ForthWorth and Texas/Hays/Austin. Similar examples are shown for World Cup datasets in Table 4.3.

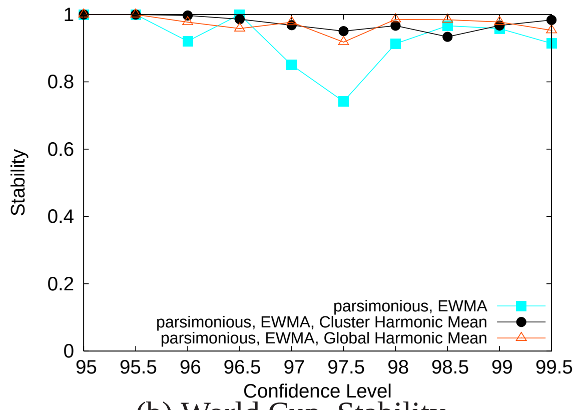
Figure 4.4 considers hierarchical relationships among the explanation nodes. If many nodes in the explanation set have descendant nodes that are also part of the explanation, then this indicates the importance of hierarchical explanations, as descendant nodes are needed to explain trends that are different from the ancestors in the explanation; these could be stronger trends or counter-trends compared to the ancestor node. Thus, for each node in the explanation set, we counted how many descendants below it are also part of the explanation. Let $V(l)$ be the number of explanation nodes at level l and $V(l)^B$ be the number of explanations nodes at level l which have at least one explanation node as descendant. Then we compute $V(l)^B/V(l)$. In these plots, level 0 indicates the root. We observe that significant number of counties ($> 25\%$) have cities which have different trends in population under all prediction models.

4.3.4 Parsimony

In Figure 4.5 we compare the parsimonious explanations against those obtained by the naive non-hierarchical approach. We use three different prediction models in which the mean of

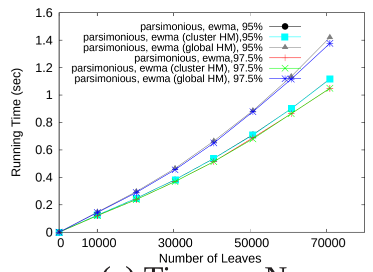


(a) World Cup, Parsimony

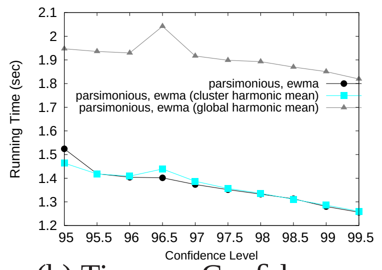


(b) World Cup, Stability

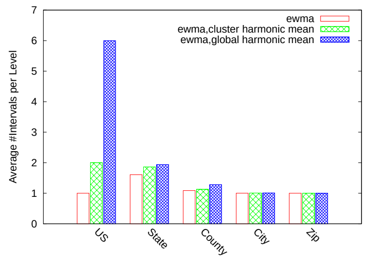
Figure 4.7: (a) Number of explanations; and (b) Stability as function of confidence for non-hierarchical and parsimonious algorithms World Cup data.



(a) Time vs. N



(b) Time vs. Confidence



(c) Avg. # intervals per level

Figure 4.8: Running time (in sec) of parsimonious algorithm as a function of (a) number of leaves, N and (b) confidence levels on Census data. (c) shows the average number of intervals per level on Census data.

prediction is given by the EWMA model but the variances $V(\epsilon)$ are different – EWMA with a separate variance per leaf, same variance for each element in a cluster, set to the harmonic mean of cluster variances (leaves belonging to same parent), single variance for each leaf, set to the harmonic mean of the global variances. Note that the latter two estimates underestimate variability for a large fraction of nodes; we choose them to study the effect of inferior prediction model on our algorithms.

As the confidence level increases, the precision decreases and therefore, all the curves show decreasing trends monotonically. In Figure 4.5(a) we observe that the parsimonious model with EWMA offers the best parsimony with the smallest number of explanations, followed by non-hierarchical on EWMA model, thus showing the advantage of parsimonious algorithm. As expected, the performance of EWMA model with global harmonic mean of the variances perform worst in terms of parsimony.

We show the parsimony of our algorithm by comparing with the DIFF operator [99]. Since the technique in [99] puts constraints on the intermediate nodes, we have to modify our algorithm so that we have a tolerance parameter k in each node, and we use this model in this comparison shown in Figure 4.6. We compare two different snapshots - year 2003 and 2004 from Census data; and May 26 and 27 from World Cup data. It is to be noted that x-axis in Figure 4.6 is per-node tolerance, $k > 0$ and it is not in log-scale.

The improvement in the number of explanations when using our model is significant, up to two orders of magnitude. The improvement is more evident in the Census data, which exhibit hierarchical trends, compared to the World Cup data.

Figure 4.5 (b) show the average stability across all levels for both non-hierarchical and parsimonious algorithms. We observe that with increase in confidence level, the stability decreases since the set of nodes which are explanations changes. Since Census data is homogeneous with 4 levels, we observe almost monotonic change in stability with increase in confidence level except for the parsimonious algorithm with the best EWMA model at confidence level 97. Similarly, we observe parsimony and stability on World Cup datasets in Figure 4.7.

4.3.5 Efficiency

In Figures 4.8 (a)-(b), we show the runtime of the parsimonious model on Census data (minimum time over 5 runs) as function of the number of leaves, N and confidence level respectively (using all three models). First, the increase in running time with N follows O($hN\log(N)$) growth. Second, we observe that all the algorithms show a decreasing trends (prominently in Census data) in running time with increase in confidence level (increasing error) leading to a small number of intervals in the non-leaf nodes. Third, we observe that the parsimonious algorithm with EWMA model has least running time complexity. That means, variances in individual leaves can summarize changes well whereas the algorithm which uses harmonic mean of variances cannot summarize the changes that well and thus leads to many explanations (and intervals) up in the trees. To vary the number of leaves, we sample each leaf with some probability to be included in the tree. We also show it for two different values of confidence levels.

In Figure 4.8(c), we show the space complexity of the 3 parsimonious algorithms on Census (similar trend on World Cup data but better), averaged over all nodes per level. We observe that the average number of intervals per node is very close to 1 except for parsimonious algorithm using EWMA model with same variance per leaf node (estimated by the Harmonic Mean of the individual leaf Variances). Furthermore, we observe that the same parsimonious algorithm has higher running time and larger number of average intervals at different levels.

4.4 Summary

Change detection is an underlying challenge for anomaly detection and understanding of trends in data. This chapter addresses the change detection challenges in hierarchical data and proposes a natural model for explaining the changes. The statistical model is general enough that it can handle the trade-offs between confidence level and accuracy in finding the explanations of the changes. The next chapter takes up the problem of application classification using different approaches.

Chapter 5

Application Classification

In this chapter, we critically re-visit the problem of application traffic classification. Accurate classification of network traffic can enable the operators to do a variety of operations such as debugging, accounting, security configuration. In addition, knowledge of the accurate applications can help in service differentiation and executing enterprise policies.

Initially, we will discuss the performance metrics for comparing different methodologies in Section 5.1.1. Then, we will describe the datasets that we use to evaluate various approaches in Section 5.1.2. We introduce the candidate Machine Learning Algorithms that we use in our study in Section 5.1.3.

5.1 Comparison Methodology

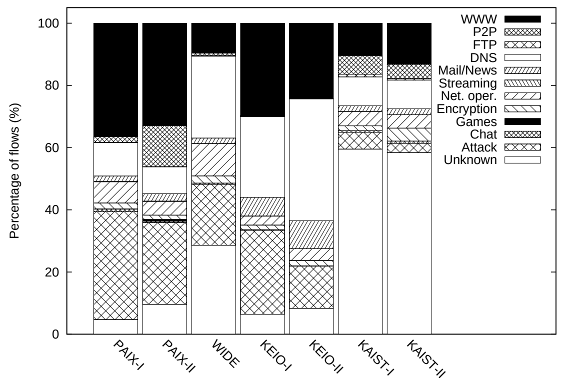
Here we describe our comparison methodology, including performance metrics, dataset, comparison benchmark, and experimental setup for machine learning algorithms. We use the definition of a flow based on its 5-tuple (source IP address, destination IP address, protocol, source port, destination port) with a timeout of 64 seconds [33].

5.1.1 Performance metrics

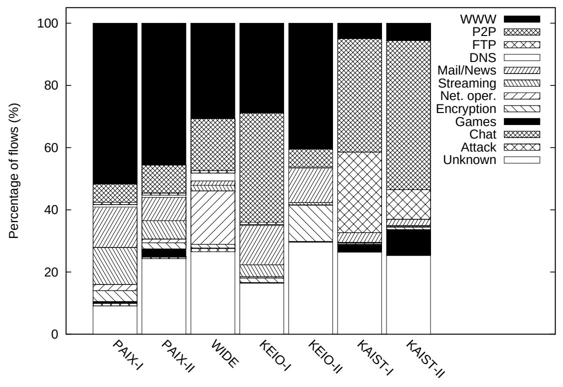
To measure the performance of CoralReef, BLINC, and machine learning algorithms, we use four metrics: *overall accuracy*, *precision*, *recall*, and *F-Measure*.

- *Overall accuracy* of an algorithm is the ratio of the sum of all True Positives to the p sum of all the True Positives and False Positives for all classes. [1] We apply this metric to measure the accuracy of a classifier on the whole dataset. The other three metrics are used to evaluate the quality of classification results for each application class.
- *Precision* of an algorithm is the ratio of True Positives to the sum of True Positives and False Positives, i.e., the percentage of flows (or bytes) that are properly attributed to a given application by this algorithm.
- *Recall* of an algorithm is the ratio of True Positives to the sum of True Positives and False Negatives, i.e., the percentage of flows in an application class that are correctly identified.
- Finally, *F-Measure*, a widely-used metric in information retrieval and classification [110], considers both *Precision* and *Recall* in a single metric by taking their harmonic mean:

[1]True Positives is the number of correctly classified flows, False Positives is the number of flows falsely ascribed to a given application, and False Negatives is the number of flows from a given application that are falsely labeled as another application.



(a)Percentage of flows



(b)Percentage of bytes

Figure 5.1: Application breakdown. Note that some of the filler-patterns are repeated.

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

Table 5.1: Characteristics of analyzed traces

$2 \times Precision \times Recall/(Precision + Recall)$. F-measure can be expressed in terms of True Positives, False Positives and False Negatives as $2\times$ True Positives / ($2\times$ True Positives + False Positives + False Negatives).

5.1.2 Data Set and Comparison Benchmark

Our dataset consists of seven payload traces were collected at two backbone and two edge links located in the US, Japan, and Korea (see Table 5.1). The PAIX backbone traces were taken on a bidirectional OC48 trunk of an US Commercial Tier 1 backbone link connecting San Jose and Seattle. The WIDE trace was captured at a 100 Mbps Ethernet US-Japan Trans-Pacific backbone link that carries commodity traffic for WIDE member organizations. The

Keio traces were collected on a 1 Gb/s Ethernet link in Keio University Shonan-Fujisawa campus. The KAIST traces were captured at one of four external links connecting a 1 Gb/s KAIST campus network and a national research network in Korea.

To establish a reference point in evaluating the algorithms, we use the payload-based classifier developed in [68], which we augment with more payload signatures from [102, 40, 111] and manual payload inspection. Our resulting classifier includes payload signatures of various popular applications, summarized in Table 5.2. The payload classification procedure examines the payload contents of each packet against our array of signature strings, and in case of a match, classifies the corresponding flow with an application-specific tag. Previously classified flows are not re-examined unless they have been classified as HTTP, in which case re-examination may enable identification of non-web traffic relayed over HTTP (e.g., streaming, p2p, etc.) [68].

After the payload-based classification process, we identify scanning activities using scan detection heuristics in [19]. Flows that could not be classified during the signature matching and scanning detection processes are categorized as unknown, which represents 4.7%-9.6% of flows in the PAIX and Keio traces, 28.6% in the WIDE trace, and around 60% in the two KAIST traces. Approximately 90% of those unknown flows in the KAIST traces were from/to three PlanetLab [57] machines. Our experience and Karagiannis *et al.*'s study [67] with payload classification suggest that the first 16 bytes of payload suffice for signature-based classification for most legacy and P2P applications except Gnutella, particularly on the PAIX and Keio traces where unknown flows represent less than 5%-10%. Gnutella (and its variants) uses variable length padding; Erman *et al.*'s measurements indicate that 400 payload bytes of each packet is required to identify 90% of the Gnutella flows using payload signatures [44]. We exclude attack and unknown flows from our analysis.

Figure 5.1 shows payload classification results for our traces. The traces vary widely in application mix, motivating our per-application analysis. Scanning traffic contributes 14%-35% of flows in the WIDE, Keio, and PAIX traces in Figure 5.1.

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

Table 5.2: Application categories

5.1.3 Machine Learning Approach

We address three main challenges of traffic classification approaches that use supervised machine learning algorithms and flow features to train the models. The challenges are as follows:

1. Finding a set of key flow features that capture fundamental characteristics of different

types of applications [82, 109].

2. Finding the most accurate classifier(s) with acceptable computational cost [109].

3. Obtaining representative datasets with ground truth for various applications, i.e., datasets that contain correct and complete instances of application flows, in terms of their fundamental flow features [44].

Flow features

We use unidirectional flow features of TCP and UDP traffic to build a classifier that handles both TCP and UDP as well as backbone and edge traffic. We use 37 unidirectional flow features most of which are inspired from 248 bidirectional features used in [82] and 22 bidirectional features in [108, 109]. The 37 features are: protocol, source and destination ports, the number of packets, transferred bytes, the number of packets without Layer 4 (TCP/UDP) payload, start time, end time, duration, average packet throughput and byte throughput, max/min/average/standard deviation of packet sizes and inter-arrival times, number of TCP packets with FIN, SYN, RST, PUSH, ACK, URG (Urgent), CWR (Congestion Window Reduced), and ECE (Explicit Congestion Notification Echo) flags set (all zero for UDP packets), and the size of the first ten packets. Figure 1.2 (a) shows different TCP header flags. FIN flag indicates that the sending end will not send any more data. SYN flag is set for the initial packets of a TCP connection where both the ends of the connection have to synchronize their TCP states. RST flag is set when the receiving end of the connection needs to be reset. This flag is also set when the sending side encounters any errorneous packet such as acknowledgment to a packet that was never sent. PUSH flag is set when the sending side of the connections wants to convey that the data in the receiver buffer should be sent to the application immediately. ACK flag establishes the validity of the acknowledgment number. URG flag is set when the urgent pointer is valid and it indicates that the data should be handled urgently, even before normal data is processed. CWR stands for Congestion Window Reduced and is applicable when ECN is enabled. The TCP sender sets this flag when it wants to inform the received that the congestion window has been reduced in response to congestion indication. ECE standars for Explicit Congestion Notification Echo and is used when ECN is enabled. The receiver sets this flag to inform the sending host know of congestion.

Feature Selection

Feature selection, as a preprocessing step to machine learning, is the process of choosing a subset of original features that will optimize for higher learning accuracy with lower computational complexity. The process removes irrelevant [91] and redundant [22] features, i.e., those that can be excluded from the feature set without loss of classification accuracy, thus improving algorithm performance.

There are two general approaches to feature selection: *filters* and *wrappers*. Filter methods are preprocessing steps performed independent of a classification algorithm. Wrapper methods attempt to search through the space of feature subsets using the criterion of the classification algorithm to select an optimal feature subset. Since wrapper methods involve repeatedly executing an algorithm for each possible subset of features, it is computationally expensive and impractical time-wise, particularly when evaluating several learning algorithms on large data sets [108]. For this reason, we only investigate filter methods. We select the Correlation-based Filter, which outperforms the other filter method (Consistency based Filter) in terms of classification accuracy and efficiency [108, 109]. The Correlation-based Filter examines the relevance [27] of each feature, i.e., those highly correlated to a specific class but with minimal correlation to each other [109]. We use the Best First search to generate candidate sets of features from the feature space, since it provides higher classification accuracy (percent of correctly classified instances) than Greedy search [108, 109]. The Best First search creates a new subset based on the addition or removal of features to the current subset. Unlike Greedy search, it can also backtrack in the selection process when it observes no improvement.

Supervised Machine Learning Algorithms

We use the WEKA machine learning software suite [15], often used in traffic classification efforts [80, 42, 40, 87, 82, 109], to evaluate the seven most commonly used supervised machine learning algorithms: Naive Bayes [82, 109], Naive Bayes Kernel Estimation [82, 109], Bayesian Network [109, 108], C4.5 Decision Trees [108], *k*-Nearest Neighbors [96], Neural Networks [108] and Support Vector Machines [108, 25, 77]. We explore the following questions:

1. Which algorithms perform best in classifying traffic ?

2. How does training set and its size affect the classification performance of learning

algorithms ? In order words, how many training instances each algorithm requires to achieve a certain level of accuracy and per-application performance ?

3. How consistent are the results acrosss different datasets ?

To this end, we conduct seven experiments for the comparison of the algorithms on each trace, varying the size of the sampled training set while using the same, fixed number of testing set. To separate the training and testing sets, 50% of each trace is chosen randomly to form a training dataset and the remaining flows form testing dataset. The original datasets contain more than millions or hundreds of thousands of flows. Our sampled training datasets contain 100, 500,1000,5000,10,000,50,000 and 100,000 training flows (collected from training datasets). We randomly sample 200,000 flows from the testing datasets. [2] We briefly describe all the evaluated algorithms below:

Naive Bayes is a simple probabilistic classifier based on Bayes' theorem, which analyzes the relationship between each feature and the application class for each instance to derive a conditional probability for the relationships between the feature values and the class. The naive aspect is the assumption that all attributes ($X_1, \ldots, X_n$) are conditionally independent of one another, given the class Y. This assumption dramatically simplifies the representation of $P(X|Y)$, and the problem of estimating it from the training data.

Naive Bayes Kernel Estimation is a generalization of Naive Bayes which models features using multiple Gaussian distributions, considered more accurate than using a single Gaussian distibution for traffic classification [82, 109].

[2]The smallest trace of ours contains approximately 420,000 flows labeled with payload classification results.

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

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

***k*-Nearest Neighbors** computes Euclidean distances from each test instance to the *k* nearest neighbors in the *n*-dimensional feature space. The classifier assigns the majority class label among the *k* nearest neighbors to the test tuple. This technique scales poorly with the number of training and testing instances, since each new test tuple is compared to every tuple in the training set. We use $k = 1$, by which we obtain the highest overall accuracy among the experiments where we test with $k = 1, 3, 5, 7, 9, 11, 13, 15, 17$, and 19.

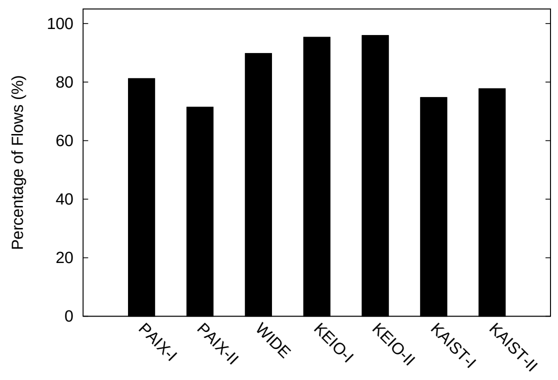
Neural Networks is a highly interconnected network of units, neurons, whose output is a combination of the multiple weighted inputs from other neurons. We use the simplest and most common Neural Network classifier called the Multilayer Perceptron, which consists of a single input layer of neurons (features), a single output layer of neurons (classes), and one or more hidden layers between them. Following [108, 15], we set the learning rate (weight change according to network error) to 0.3, the momentum (proportion of weight change from the last training step used in the next step) to 0.2 and we ran the training for 500 epochs (an epoch is the number of times training data is shown to the network).

Support Vector Machines (SVM) refers to a learning system based on recent advances in statistical learning theory. The basic principle of SVM is to construct the optimal separating hyperplane, which maximizes the distance between the closest sample data points in the (reduced) convex hulls for each class, in an n-dimensional feature space [25]. Intuitively, we would expect that this boundary to generalize better than other possible boundaries between classes. We use the Sequential Minimal Optimization (SMO) [93], a faster algorithm for training Support Vector Machines that uses pairwise classification to break a multi-class problem into a set of 2-dimensional sub-problems, eliminating the need for numerical optimization. The two most important parameters in SVM are the complexity parameter C and the polynomial exponent p [77, 108]. Li *et al.* [77] showed that varying the complexity parameter C influenced the overall accuracy of their SVM traffic classifier by only a little (around 1% at most). We use 1 for both parameters as in [108, 15].

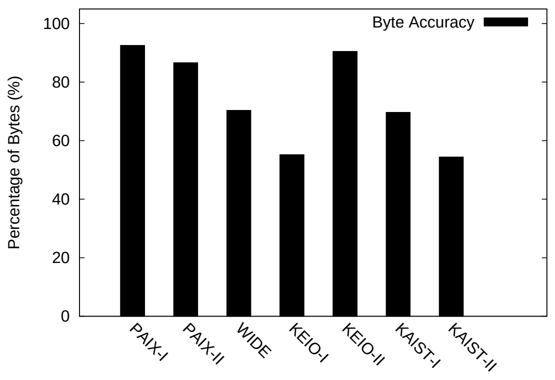
5.2 Performance Evaluation

We evaluate the performance of nine algorithms for Internet traffic classification: CoralReef, BLINC, and the seven machine learning algorithms described. To evaluate port-based classification, we compare the performance of CoralReef's port classification rules [2] with our payload-based classifier, which we use to find ground truth.

5.2.1 CoralReef

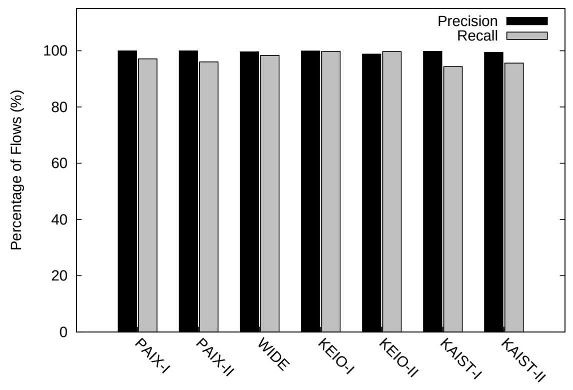


(a) Flow accuracy

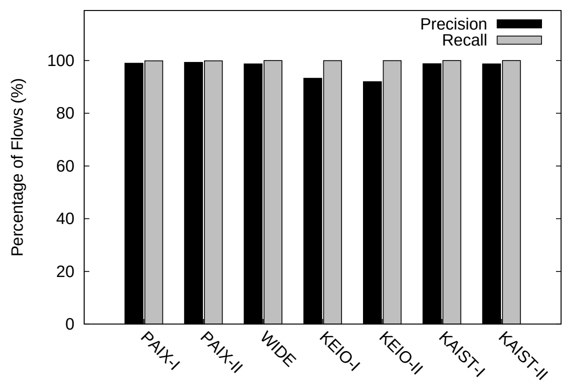


(b) Byte accuracy

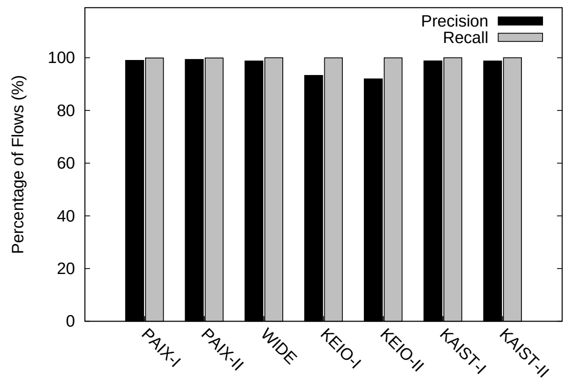
Figure 5.2: Overall Accuracy of CoralReef



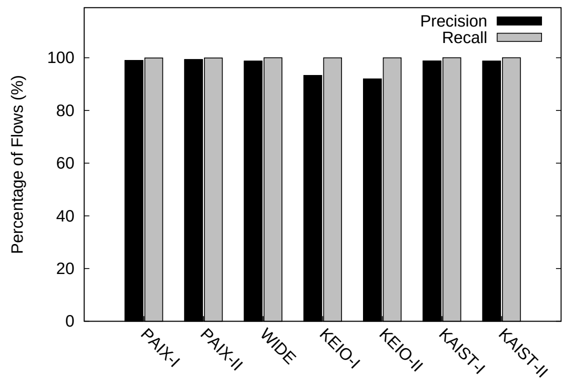
(a) WWW



(b) DNS



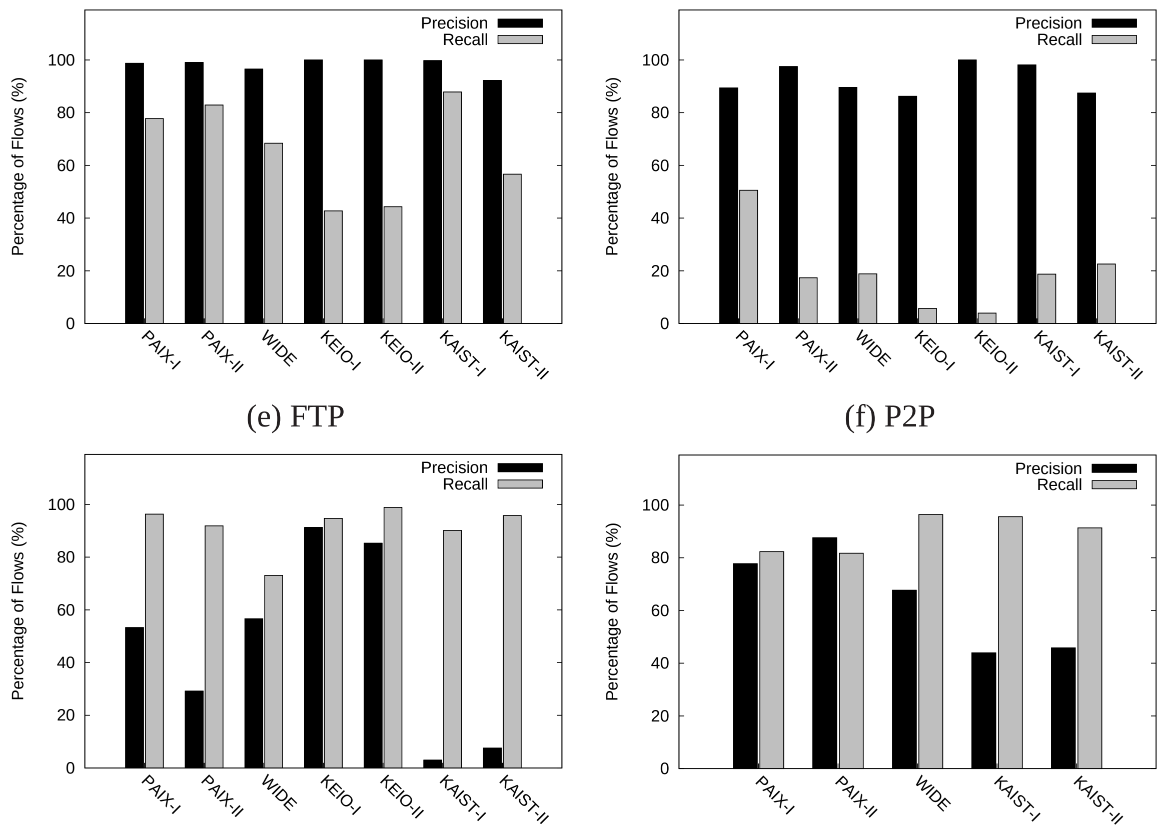
(c) Mail



(d) Chat

Figure 5.3: Per-application precision & recall of CoralReef (WWW,DNS,Mail,Chat)

Overall Accuracy:



(e) FTP (f) P2P

(g) Streaming (h) Game

Figure 5.4: Per-application precision & recall of CoralReef(FTP,P2P,Others)

The overall accuracy of any port-based classification reflects how much traffic in the examined traces obeys the default ports usage. Figure 5.2 (a) shows that the overall flow accuracy of CoralReef on the traces ranges from 71.4% to 95.9% and Figure 5.2(b) show that the overall byte accuracy ranges from 50% to 90%. Comparing Figure 5.2 with Figure 5.1 (a), we find that the overall accuracy of CoralReef is highly dependent on the traffic mix, e.g., inversely proportional to the fraction of P2P flows in a given trace. The PAIX-II and KAIST traces with the highest fraction of P2P flows (4.0%-13.2%) have the lowest overall accuracy with CoralReef classification. In contrast, the WIDE and Keio traces on which CoralReef achieves the highest overall accuracy contain the smallest portion of P2P flows (less than 1%) among all examined traces. These observations motivate our detailed study of per-application performance of CoralReef, which we summarize next.

Per-application performance

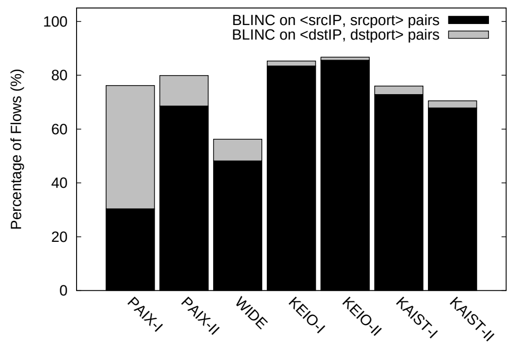
Figure 5.3 and 5.4 show the per-application precision and recall of CoralReef on eight major applications: WWW, DNS, Mail, Chat, FTP, P2P, Streaming, and Games, which comprise most (86.6%-95.7%) of the traffic flows whose ground truth we know. As shown in Figure 5.3 and 5.4, we find that each application consistently shares one of three sets of distinct characteristics across all traces – (i) high precision and high recall (WWW, DNS, Mail, and Chat); (ii) high precision but lower recall (P2P and FTP); and (iii) lower precision but high recall (Streaming and Game). The high precision of a port-based classifier such as CoralReef on an application implies that its default ports are seldom used by other applications whereas high recall implies that the corresponding application mostly uses its default ports.

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

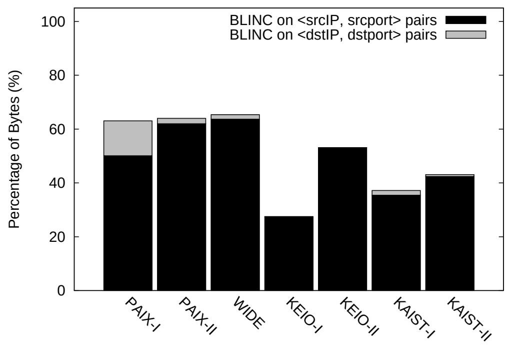
Nonetheless, it is important to recognize that port-based classification fails to yield accurate classification results in the following two cases: (i) when an application uses ephemeral non-default ports, e.g., P2P and passive FTP data transfer degrade the recall of CoralReef. In our data set, 49.4%-96.1%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

Finding 1 *Port-based classification can accurately identify legacy applications (though the two backbone traces were collected in 2004); its weakness is in identifying applications that use ephemeral ports or traffic masquerading behind a port typically used for another application.*

Although we did not apply our analysis to attack flows or those for which we did not have any ground truth, this finding suggests that ports still possess significant discriminative power in classifying certain types of traffic.



(a) Flow Accuracy



(b) Byte Accuracy

Figure 5.5: Overall accuracy of BLINC

5.2.2 BLINC

The approach taken by BLINC does not depend upon solely on port information, rather it depends on the relationship between hosts. In order to use BLINC on each datasets, we perform about 25 trials to configure BLINC's 28 threshold parameters for the best performance in precision and recall (precision takes precedence in tradeoffs, since recall errors can be mitigated by other methods [68]). Parameter values that optimize the precision may differ on different links, so separate (per-trace) tuning prevents degradation of overall accuracy by 10%-20%. Our experience also suggests that one should tune the BLINC parameters related to P2P applications first since almost every BLINC module relies on them.

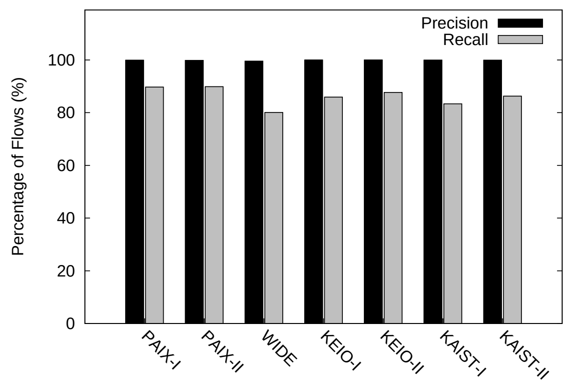
Overall Accuracy

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

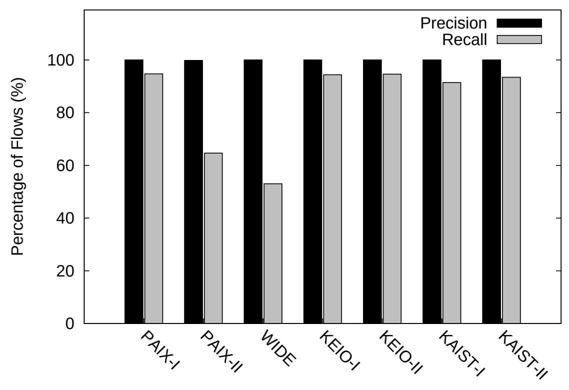
Figure 5.5 shows the overall accuracy of the modified code, Reverse BLINC, on our traces. Reverse BLINC on destination $\langle$IP, port$\rangle$ pairs improved the overall flow accuracy on the PAIX and WIDE backbone traces by as much as 45%, since in those traces one of the two directions of traffic is often missing due to asymmetric routing. Most of the flows that Reverse BLINC identified were of WWW and P2P clients.

Per-application performance

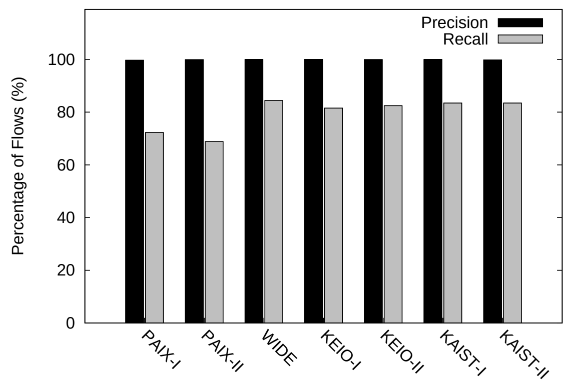
Figure 5.6 and 5.7 show BLINC’s per-application precision and recall. Once tuned, BLINC classifies WWW, DNS, Mail, Chat, FTP, and Streaming flows with greater than 90% precision. However, recall for these applications is weaker than precision, since all classification is threshold-based: the number of application flows from a given source must exceed a certain threshold in order to trigger classification. If there are too few flows from this source, its traffic remains unclassified. DNS, Mail and Chat have lower recall in backbone traces than in edge traces, because even Reverse BLINC could not capture those application flows



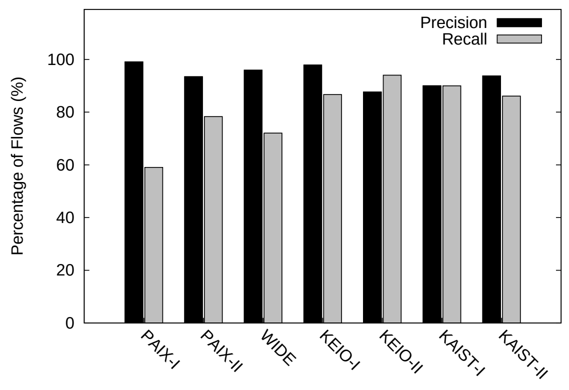
(a) WWW



(b) DNS



(c) Mail

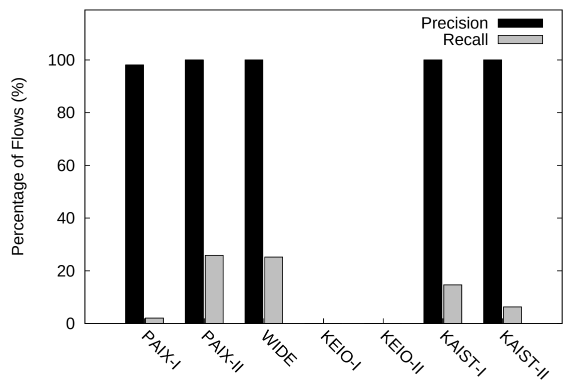


(d) Chat

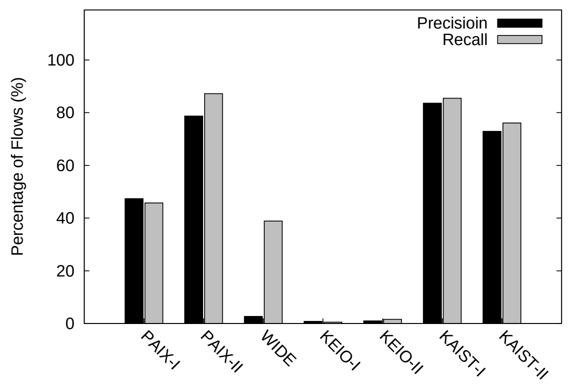
Figure 5.6: Per-application precision & recall of BLINC (WWW, DNS,Mail,Chat)

when server flows were missing from backbone traces. Recall for FTP, Streaming, and Game is always lower than 25.8% across all traces, since host behavior signatures of BLINC for these applications do not cover the following cases: (i) when a Streaming or FTP server concurrently provides any other application services; (ii) when a Game client sends any TCP flows or talks to only a few destination hosts.

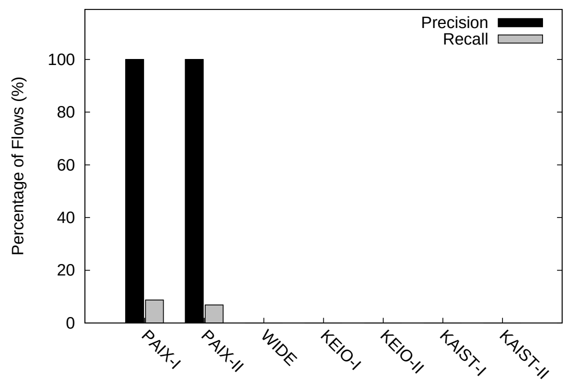
With proper tuning, BLINC reliably identifies P2P flows, particularly when we first apply port-based classification to filter out DNS server-to-server (indeed essentially P2P) flows which BLINC often misclassifies as P2P. When we filter out DNS flows first and then apply BLINC to the remaining flows, BLINC achieves >85% precision for P2P application flows.



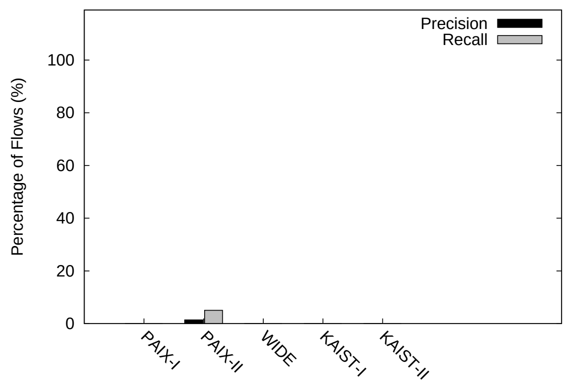
(e) FTP



(f) P2P



(g) Streaming



(h) Game

Figure 5.7: Per-application precision & recall of BLINC (FTP,P2P,Streaming,Game)

However, recall of P2P traffic measured in bytes is significantly (20.5%-61.9%) less than that measured in flows. This difference in recall is due to the fact that some P2P applications usually assign different ephemeral ports for every single data transfer. If such transfers are large, then they account for a large number of bytes, but the number of flows remains below our classification triggering threshold, so this traffic remains unclassified.

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

Computational Performance

When running BLINC, the running time and memory usage depend on the number of flows that need processing in a time interval. The BLINC code (in C++) processed the Keio, KAIST, WIDE, and PAIX-I traces in real-time using less than 2 GB of main memory. These traces contain less than one million flows per five minute interval on average. However, it took 16 hours to process the 2 hours of PAIX-II trace containing 4.7 million flows per interval on average, consuming around 9-10 GB of memory. We used a PC server with two 2.4 GHz Zeon CPUs and 4 GB of memory to run BLINC on the Keio, KAIST, and WIDE traces. For the PAIX backbone traces, we used a SUN Fire 15000 system with 228 GB of memory and 72 UltraSPARC3 900 MHz CPUs (used only one CPU).

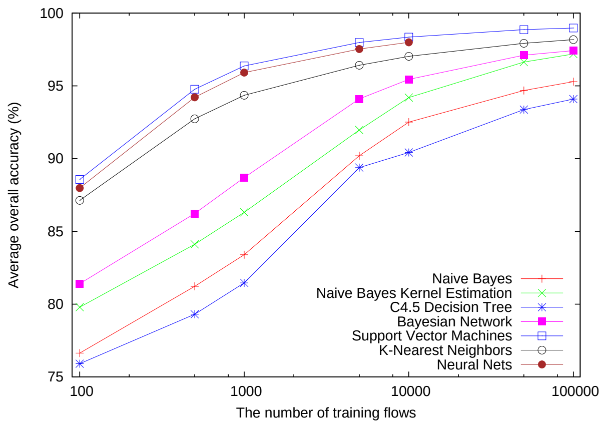


Figure 5.8: Average overall flow accuracy of machine learning algorithms by training set size

5.2.3 Supervised Machine Learning Algorithms

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

Key Flow Features

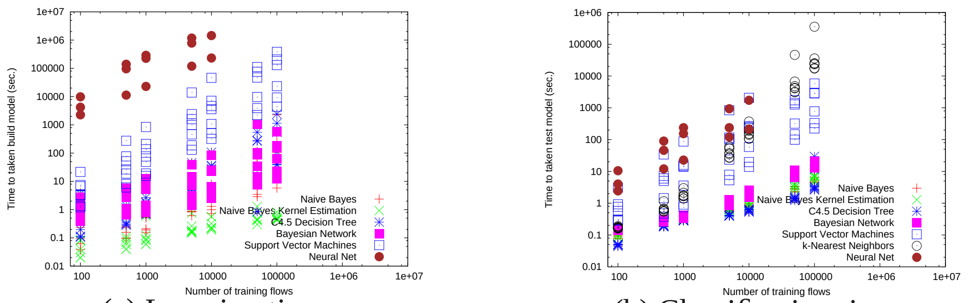
We first find key flow features for accurate traffic classification using the Correlation-based Filter (CFS) with Best First search. For every trace, the CFS selected four categories of features: protocol, ports, TCP flags and packet size information, reducing the number of features required from 37 to 6-10. Features such as inter-arrival times, which vary greatly by link, are not chosen as a key discriminator in any trace.

According to our analysis, using the selected feature subset degrades overall accuracy by only 0.1-1.4% compared to using all 37 features, while drastically reducing required training time, which increases the model (classifier) building speed by a factor of 3-10. The feature selection process thus provides an excellent trade-off between feature space reduction and loss of accuracy, confirming findings in [109]. Henceforth we will use the selected key features to evaluate the performance of the learning algorithms.

Overall accuracy

Figure 5.8 shows the overall flow accuracy of the seven machine learning algorithms as the training set size varies (from 100 to 100,000). Figure 5.8 does not show the results of the Neural Network method for larger training set sizes, since the algorithm was prohibitively slow in building a classifier with more than ten thousand training instances (Figure 5.9(a)).

For every trace, with any size training set, we always obtain consistent results. In our



(a) Learning time (b) Classification time

Figure 5.9: Computational performance of ML algorithms by training set size

experimental set up, the Support Vector Machines (SVM) method achieves the highest overall accuracy, followed by Neural Network (although it is quite slow to train) and *k*-Nearest Neighbors. The best performing algorithm, SVM, achieves more than 98.0% average accuracy on all traces with 5,000 training flows, which amounts only 2.5% of the size of the testing sets. SVM appears to need little training – around five to ten thousand training instances sufficed in our study – which makes it promising for practical Internet traffic classification since training data is scarce [44]. The Neural Net method achieves similar accuracy but is 10-1000 times slower than SVM in training and testing, when evaluated on the same dataset.

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

Computational Performance

Figure 5.9 (a) and 5.9 (b) show the learning time and classification time of the seven algorithms with increasing training set size.[3] Naive Bayes, Naive Bayes Kernel Estimation,

[3]Note that we have evaluated the performance of concrete implementations in the Java-based (slow) WEKA

Bayesian Networks, and C4.5 Decision Trees are the four fastest algorithms in both learning and classification followed by *k*-Nearest Neighbors, Support Vector Machines and Neural Network. Since *k*-Nearest Neighbors does not really involve any training, Figure 5.9(a) does not include plots for the algorithm. In general, it takes longer to train an algorithm than to perform actual classification except in case of Naive Bayes. The fastest classification algorithm is C4.5 Decision Tree. While *k*-Nearest Neighbors learns and classifies quickly with a smaller training set, its classification time curve shows the steepest increase as the training set size grows, eventually becoming slower than SVM when trained with more than ten thousand instances. While it takes longer to build an SVM classifier, its classification time is ten to hundred times shorter than its learning time, making it more practical than the *k*-Nearest Neighbors and Neural Network methods. The Neural Network method is the slowest particularly in learning.

We run WEKA on two different platforms: SUN Fire 15000 system with 228 GB memory and seventy two 900 MHz UltraSPARC3 CPUs, and IBM DataStar system with 256 GB memory and thirty two 1.7 GHz IBM Power4+ CPUs (used only one CPU).

Per-application performance

Figure 5.10 shows per-application performance, F-measure, of the seven machine learning algorithms by training set size. The SVM performs the best in terms of the per-application F-measure as well, attains over 95% F-measure for any application with more than a few

software suite on our test platform, not the theoretical complexity of the algorithms because (i) traffic classification efforts [80, 42, 40, 87, 82, 109] have often used WEKA, and (ii) this approach yields tangible performance numbers for and comparisons [109]. Optimized implementations would likely yield faster learning and classification speeds for all algorithms.

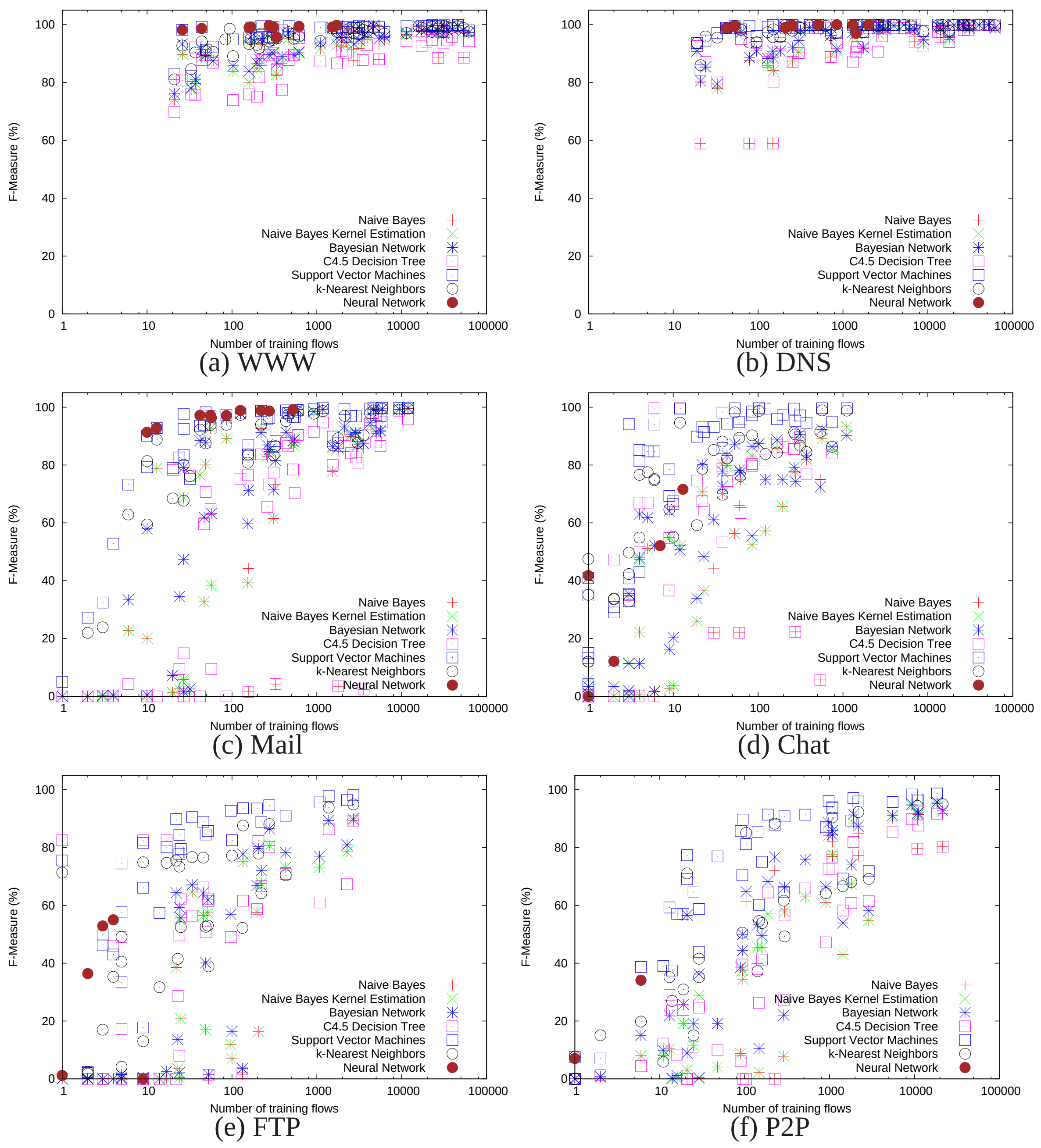


Figure 5.10: Per-application F-measure of machine learning algorithms by training set size

thousand training flows. Figure 5.10 shows that the per-application F-measure of the SVM significantly drops as the training set size decreases to fewer than 1000. *k*-Nearest Neighbors achieves lower F-measures than those of SVM particularly on P2P, FTP, Streaming, and

Chat. The Neural Network method also underperforms on our traces, though we have only limited results for per-application F-measure due to its extremely slow training.

All the algorithms classify Web and DNS traffic accurately on all datasets. A few hundred training flows are enough to identify them with more than 88%-95% F-measure. In contrast, P2P and FTP applications require the most training, not surprising since each application category itself contains multiple applications and/or communication patterns, e.g., data channel and control channel of FTP, etc. The F-measure of Naive Bayes, Bayesian Network, and C4.5 Decision Tree on P2P was at most 40%-80% even with more than ten thousand training flows. Other applications are in between those two groups of Web/DNS and P2P/Streaming/FTP in terms of F-measure.

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

5.2.4 Comparative Analysis

Figure 5.12 compares the overall accuracy of the evaluated methods: CoralReef, BLINC, and the seven machine learning algorithms, on our datasets, showing the highest two, the lowest two, and the median values we obtained from each method. In general, machine learning algorithms, and in particular, the SVM, k-Nearest Neighbors, and Neural Net show

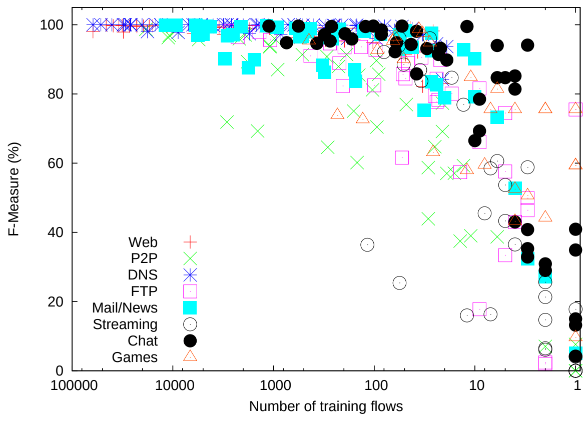


Figure 5.11: Per-application F-measure of the SVM by training set size

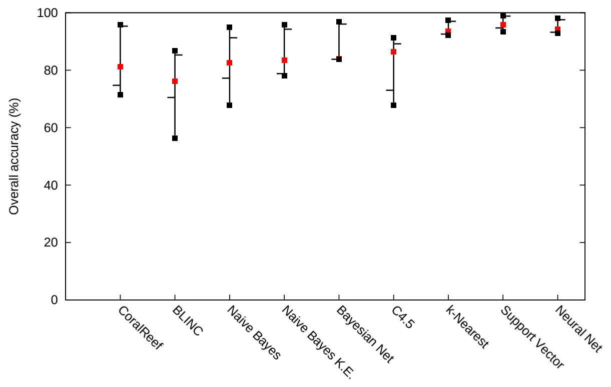


Figure 5.12: Overall accuracy of all methods (With 1000 training instances for machine learning algorithms)

higher and more consistent performance across all the traces than other methods. The overall accuracy of port-based classification such as CoralReef varies according to the proportion of flows using officially designated ports, while BLINC's accuracy strongly depends on both topological location (e.g., backbone vs. edge link, unidirectional vs. bi-directional link, international, domestic vs. local etc.) and traffic mix. The SVM consistently outperformed all other methods we evaluated. Although Neural Network seems to perform almost as well as SVM (only slightly less; 0.4%-1.1% in terms of overall accuracy) performance to SVM, its computational cost is prohibitively high (see Figure 5.9(a) and (b)).

5.3 Lessons learned

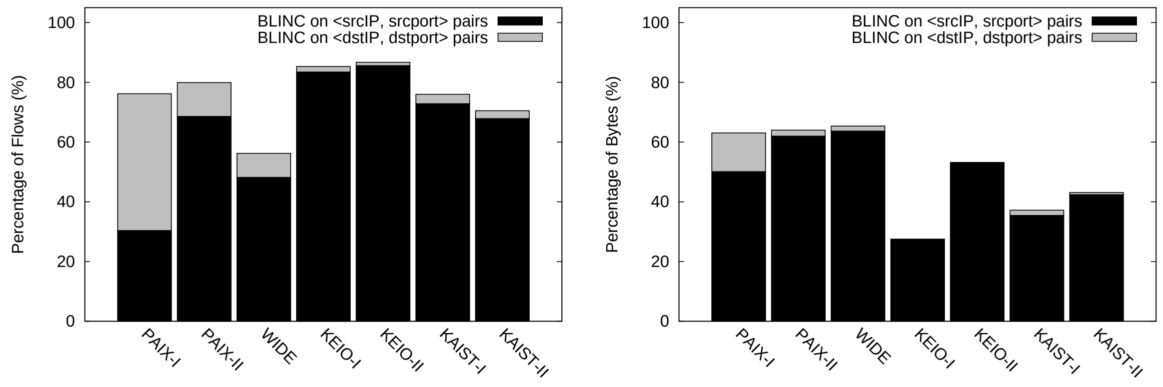
In this section, we summarize our findings and discuss on various implications of those findings.

Lesson 1 (On ports as key features): One of the key findings of this thesis is that port numbers are relevant to traffic classification. In particular, port lookup can reliably identify many conventional applications, especially when used with packet size information, TCP header flags and protocol. Excluding port information from the above key features in training an SVM classifier reduced overall accuracy from 95%-99% to 56%-70%. On the other hand, conventional applications are not what have catapulted traffic classification activities into the popular press. The more interest there is in identifying traffic in order to constrain or charge users, the more incentive there will be to hinder port-classification methods. Indeed, at any time, or on any link, traffic may use unrecognized ports, or misuse recognized ports to explicitly hide traffic. Thus, the real challenge (and fear) is not in recognizing conventional applications, but in detecting applications that are trying to hide, e.g., via port masquerading or encryption.

Lesson 2 (On behavior based classification): While port information and flow-features-based approaches make classification decisions on a per-flow basis, host-behavior-based classification as implemented in BLINC aggregates flow information for an interval to derive behavioral patterns of observed hosts. The accuracy of a host-behavior-based classification strongly depends on whether the observed link is located at a topologically appropriate place to collect enough behavioral information on hosts. Consequently, BLINC is effective on links that capture both directions of every flow to a host, such as the border link of a single-homed edge network. Host-behavior analysis is less powerful on aggregated, e.g., backbone, links, where often only a small portion of flows from/to an end-host can be observed, and where asymmetric routing prevents observation of both directions of traffic.

Lesson 3 (On byte accuracy): The other limitation of the aggregated-behavior-based approach is, even at a topologically appropriate place, these techniques will fail to classify traffic from/to entities whose flows seldom traverse the target link. As a result, they often misclassify as unknown a small number of large "elephant" flows from/to such entities, achieving lower byte accuracy (or recall) than flow accuracy (or recall). For example, the byte accuracy of BLINC was significantly lower (13.1%-59.3%) than its flow accuracy (56.2%-86.7%) on our traces. Karagiannis *et al.* had similar results in [68]. This weakness is a serious flaw for practical traffic classification, as elephant flows may account for over 70% of the bytes transferred on typical networks [29]. A complementary per-flow based classification process on remaining unclassified flows is needed to overcome this limitation.

Erman *et al.* showed that a cost-sensitive sampling approach allowed machine learning algorithms to achieve high byte accuracy and flow accuracy [43]. This approach trains a classifier with more of the rare but resource-intensive cases, i.e., elephant flows. They trained their classifier with a training set that contained 50% of flows below the 95% percentile of flow size and 50% of flows above the 95% percentile of flow size. This technique substantially improved the byte accuracy for the classifier, with only a marginal reduction in flow accuracy.



(a) Flow accuracy (b) Byte accuracy

Figure 5.13: Overall flow and byte accuracy of BLINC

Lesson 4 (On single vs. bidirectional flow features for backbone traffic classification): Accurate traffic classification is feasible only when a classifier possesses correct, complete fingerprints for target applications. Previous efforts on flow-features-based classification [80, 82, 96, 26, 40, 37, 113, 44, 109] have shown that bidirectional TCP flow statistics provide such fingerprints for various applications. However, these methods are not appropriate for classifying backbone traffic where one direction of a TCP flow is unobserved due to routing asymmetries. Backbone traffic classification is challenging because only partial information about bidirectional TCP connections is available. Erman *et al.* addressed this limitation by proposing an algorithm that uses the packets of an unidirectional TCP flow to estimate the flow statistics of the unobserved direction [41], leaving UDP traffic classification as future work. We address this problem by using ports as key features in addition to

TCP flags and packet size, based on (i) Lesson 1 and (ii) the results of the Correlation-based Feature Selection. The resulting classifiers show accuracies comparable to or higher than those of previous work, on all our backbone and edge traces, using only single-direction flow features. While port information does not seem necessary when we train a learning algorithm with bi-directional flow features to classify TCP traffic, it is indispensable when using only single-direction flow features to classify both TCP and UDP traffic.

5.4 Summary

Traffic classification is a challenging problem in which the network flows (at the level of transport layer) are associated with the higher layer applications. Accurate classification enables the IT operators to better provision and manage the network resources. This chapter addresses the disadvantages and advantages of the existing traffic classification methods. In-depth analysis of three different approaches (port-based, host-behavior-based and machine learning methods) on seven different traces reveal interesting results which provide valuable guidelines for the reseachers and engineers. The results show that the port-based classification can still be accurate for traditional applications and machine learning approach can be effective if a classifier is trained well with representative datasets. Among machine learning algorithms, we found that SVM attains highest accuracy and can therefore be a viable solution. We also find that the host-behavior method (BLINC) works well when the datasets have bi-directional flow information.

Chapter 6

Conclusion

In this chapter, we summarize the contributions and limitations of this work, and discuss directions for future research.

6.1 Contributions

This dissertation presents the design of *Godai*, a new framework to manage the enterprise networks. Godai aims to solve three problems in the current enterprise network management system. First, the current enterprise end-hosts are not properly configured when it comes to securing them against botmaster recruitement process. Second, the IT operators need traffic analysis tools which report to them important but compact summaries as well as changes among the summaries. Third, they need tools and proper tuning of their configuration to analyze traffic and identify application breakdown.

6.1.1 Configuration Management

Configuring the end-host IDS in an enterprise requires setting appropriate thresholds for different detection features. Setting right threshold values in the end-host IDS systems can force an attacker into an extremely stealthy mode or prevent it from launching attacks. *Godai* provides an approach to set thresholds in a *group* of end-host anomaly detectors. We challenge the common practice of the IT operators who opt for a single threshold across the population of end-hosts. We show that this IT practice can lead to unintended wild experience of false positives and false negatives by the end-hosts. In our attempt to configure a group of enterprise end-hosts, we observe that considering a group of end-hosts couple them and the behaviors of the end-hosts influence the common threshold value. In our unified approach we propose two components: (a) the choice of utility function to balance the trade-off between false positives and false negatives, and (b) the choice of diversity level which decides the number of different thresholds to be computed for the end-hosts. Based on the evaluation using real data, we find that:

1. the natural user diversity offers tremendous opportunities for an attacker to "hide",
2. the choice of utility function can have huge impact on the false positive and false negative experienced by the end-hosts,
3. the diversity in the thresholds can be beneficial if false positives are more important in an enterprise, and
4. a handful of thresholds seem to be capable of providing significant benefits compared

to homogeneous approach

The last observation is very promising as it argues that diversity is not an all-or-nothing proposition: it may be able to strike a compromise between the effectiveness and the operational simplicity. Our findings point to the need of a comprehensive re-evaluation of the way how IT operators set IDS thresholds.

6.1.2 Hierarchical Summary

Godai proposes a natural model for explaining the changes in hierarchical data and formulates two problem variants for finding a parsimonious explanation in this model. Our model makes an effective use of the hierarchy and describes changes at the leaf nodes as a composition of node weights along each path of each root-to-leaf path in the hierarchy. We design algorithms to minimize the explanation size for both the problem variants. Despite the fact that assigning node weights optimally is an under-constrained problem, we have shown that it is not NP-hard and that our algorithms require time proportional to the product of the number of leaves and the depth of the dimension hierarchy.

We evaluate our approach on real data to demonstrate both its efficiency and effectiveness. In practice, the performance and space usage of our algorithms are much less than the worst-case bounds. On population census data, the explanations discovered (counter) trends, mainly at the city-level. We have made similar observations when we analyze HTTP traffic logs from the FIFA World Cup hosting site. Our approach can also be used to reveal "interesting" anomalies in hierarchical data when used in conjunction with a statistically sound predictive model that forecasts values within confidence intervals. These anomalies are explained more parsimoniously using our algorithm compared to the leaf-level anomalies that the predictive model detects.

6.1.3 Application Classification

Godai conducts a detailed comparison of three well-studied approaches to traffic classification: ports-based, host-behavior-based, and flow-features-based. We believe this is the first study to evaluate the three families of traffic classification algorithms on several data sets of payload trace from different types of network links located in multiple countries. Diversity in the data sets allow us to test the approaches under a wide variety of conditions, facilitating our assessment of the strengths and weaknesses of each approach. Our study yields several insights:

1. The effectiveness of port-based classification in identifying legacy applications is still impressive and is further strengthened by the use of packet size and TCP flag. This fact explains why research attention has shifted to detecting and identifying new applications that use port-masquerading and encrytion, i.e., traffic deliberately trying to evade traffic classification. Unfortunately, increasing attention to classifying traffic for purposes not necessarily approved by originator of the traffic is likely to increase this category of traffic, inducing an arms race between those trying to classify traffic, and those trying to avoid having their traffic classified

2. Each approach has its own strengths and weaknesses, and careful combinations can provide synergy. When an approach has a fundamental weakness in classifying particular types of traffic, integrating aspects of other techniques can help. For example, host-behavior-based methods such as BLINC can be augmented with per-flow based classification process to increase byte accuracy.

3. The Support Vector Machines algorithm consistently achieved the highest accuracy.

Scientifically grounded traffic classification research requires that researchers share tools and algorithms, and baseline data sets from a wide range of Internet links to reproduce results.

6.2 Limitations

One of the motivations of our work is to come up with a management framework which can be deployed by the enterprise IT operators. It is true how readily the IT operators will adopt our solution is out of our control. We realize that accuracy of our traffic classification methods depends on the accuracy of the availability of ground-truth information.

6.3 Future Work

Regarding the end-host configuration work, we would like to investigate other methods for clustering the end hosts into groups using multiple features simultaneously. We believe that our methodology will foster a new research direction in network management.

As far as hierarchical change detection is concerned, our approach can be extended to multiple dimensions but it presents several non-trivial challenges due to the existence of multiple parents in the hierarchy. Another natural extension we have considered for future work is where there is a global budget on error tolerance for the entire tree. Although we have found a polynomial solution, its complexity appears to be significantly higher than the problems studied in this thesis, and its feasibility on massive data sets remains to be shown.

There could be several extensions to our traffic classification work. Here we have treated Machine Learning Algorithms more like black boxes but one needs to understand thoroughly the fundamental limitations of each algorithm. Regarding BLINC, one needs to come up with automatic tuning of BLINC parameters depending on trace.

Bibliography

[1] *AVG AntiVirus*. `http://www.avgantivirus.com`.

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

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

[4] *FlowScan - Network Traffic Flow Visualization and Reporting Tool*. `http://www.caida.org/tools/utilities/flowscan`.

[5] *Intel Active Management Technology*. `http://www.intel.com/technology/platform-technology/intel-amt/,http://www3.intel.com/cd/business/enterprise/emea/ENG/310547.htm`.

[6] *McAfee AntiVirus*. `http://www.McAfee.com`.

[7] *Nagios*. `http://www.nagios.org`.

[8] *New Massive Botnet Twice the Size of Storm*. `http://www.darkreading.com/document.asp?doc_id=150292&WT.svl=news1_1`.

[9] *Norton AntiVirus*. `http://www.norton.com`.

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

[11] Qosmos. `http://www.qosmos.com`.

[12] *Storm Worm DDoS Attack*. `http://www.secureworks.com/research/threats/storm-worm`.

[13] *TCP Header*. `http://freebie.fatpipe.org/~mjb/Drawings/TCP\_Header.png`.

[14] *UDP Header*. `http://goethe.ira.uka.de/seminare/rkt/tcp\%2Budp/UDP-Header.png`.

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

[16] On change diagnosis in evolving data streams. *IEEE TKDE*, 17(5):587–600, 2005. Charu C. Aggarwal.

[17] *Bot infections in the enterprise underestimated, bigger than thought*. `http://www.darkreading.com/document.asp?doc_id=137602`, October 2007.

[18] 'surge' in hijacked pc networks, March 2007. `http://news.bbc.co.uk/2/hi/technology/6465833.stm`.

[19] Mark Allman, Vern Paxson, and Jeff Terrell. A brief history of scanning. In *ACM IMC*, April 2004.

[20] E. Amaldi and V. Kann. On the Approximability of Minimizing Nonzero Variables or Unsatisfied Relations in Linear Systems. *TCS*, 209(1-2):237–260, 1998.

[21] N. Anderson. Vint cerf: one quarter for all computers part of a botnet. January 2007. `http://arstechnica.com/news.ars/post/20070125-8707.html`.

[22] Annalisa Appice, Michenlangelo Ceci, Simon Rawles, and Peter Flach. Redundant feature elimination for multi-class problems. In *International Conference on Machine Learning*, July 2004.

[23] Sanjeev Arora, László; Babai, Jacques Stern, and Z. Sweedyk. The hardness of approximate optima in lattices, codes, and systems of linear equations. *J. Comput. Syst. Sci.*, 54(2):317–331, 1997.

[24] Dhiman Barman, Flip Korn, Divesh Srivastava, Dimitris Gunopulos, Neal E. Young, and Deepak Agarwal. Parsimonious Explanations of Change in Hierarchical Data. In *Proc. of ICDE 2007*.

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

[26] Laurent Bernaille, Renata Teixeira, and Kave Salamatian. Early application identification. In *CoNEXT*, December 2006.

[27] Avrim Blum and Pat Langley. Selection of relevant features and examples in machine learning. *Artificial Intelligence*, 97(1-2):245–271, 1997.

[28] Census (population vs. location), 2000-2004. `http://www.census.gov/popest/datasets.htm`.

[29] Kun chan Lan and John Heidemann. A measurement study of correlation of Internet flow characteristics. *Computer Networks*, 50(1):46–62, January 2006.

[30] Surajit Chaudhuri and Umeshwar Dayal. An overview of data warehousing and olap technology. *SIGMOD Record*, 26(1):65–74, 1997.

[31] Taesang Choi, Changhoon Kim, Seunghyun Yoon, Jeongsook Park, Byungjun Lee, Hyunghan Kim, and Hyungseok Chung. Content-aware internet application traffic measurement and analysis. In *IEEE/IFIP NOMS*, April 2004.

[32] Cisco. *Always Vigilant Endpoint*. `http://www.cisco.com/en/US/products/sw/secursw/ps5057/index.html`.

[33] KC Claffy, Hans-Werner Braun, and George C. Polyzos. A parameterizable methodology for internet traffic flow profiling. *IEEE JSAC Special Issue on the Global Internet*, 1995.

[34] Graham Cormode, Flip Korn, S. Muthukrishnan, and Divesh Srivastava. Finding Hierarchical Heavy Hitters in Data Streams. In *Int. Conf. on Very Large Databases*, pages 464–475, 2003.

[35] Graham Cormode and S. Muthukrishnan. What's new: Finding significant differences in network data streams. In *Proc. of IEEE INFOCOM*, pages 1534–1545, 2004.

[36] Cristian Estan and Stefan Savage and George Varghese. Automatically Inferring Patterns of Resource Consumption in Network Traffic. In *ACM SIGCOMM*, 2003.

[37] Manuel Crotti, Maurizio Dusi, Francesco Gringoli, and Luca Salgarelli. Traffic classification through simple statistical fingerprinting. *ACM SIGCOMM CCR*, 37(1):7–16, January 2007.

[38] David Wagner et al. Mimicry Attacks on Host-BAsed Intrusion Detection Systems. In *Proc. of CCS'02*, 2002.

[39] Holger Dreger, Anja Feldmann, Michael Mai, Vern Paxson, and Robin Robin Sommer. Dynamic application-layer protocol analysis for network intrusion detection. In *USENIX Security Symposium*, July 2006.

[40] Jeffrey Erman, Martin Arlitt, and Anirban Mahanti. Traffic Classificaton Using Clustering Algorithms. In *ACM SIGCOMM MineNet Workshop*, September 2006.

[41] Jeffrey Erman, Martin Arlitt, Anirban Mahanti, and Carey Williamson. Identifying and Discriminating Between Web and Peer-to-Peer Traffic in the Network Core. In *WWW*, May 2007.

[42] Jeffrey Erman, Anirban Mahanti, and Martin Arlitt. Internet Traffic Identification using Machine Learning. In *Globecom*, November 2006.

[43] Jeffrey Erman, Anirban Mahanti, and Martin Arlitt. Byte Me: A Case for Byte Accuracy in Traffic Classification. In *ACM SIGMETRICS MineNet Workshop*, June 2007.

[44] Jeffrey Erman, Anirban Mahanti, Martin Arlitt, Ira Cohen, and Carey Williamson. Offline/Realtime Traffic Classification Using Semi-Supervised Learning. In *IFIP Performance*, October 2007.

[45] Cristian Estan, Stefan Savage, and George Varghese. Automatically inferring patterns of resource consumption in network traffic. In *ACM SIGCOMM*, pages 137–148, 2003.

[46] Colin Dixon et al. Phalanx: Withstanding Multimillion-Node Botnets. In *Proc. of USENIX NSDI*, 2008.

[47] David Brumley et al. Towards Automatic Discovery of Deviations in Binary Implementations with Applications to Error Detection and Fingerprint Generation. In *Proc. of USENIX Security Symposium*, 2007.

[48] Denver Dash et al. When gossip is good: distributed probabilistic inference for detection of slow network intrusions. In *Proc. of AAAI*, 2006.

[49] Evan Cooke et al. The Zombie roundup: understanding, detecting, and disrupting botnets. In *Proc. of SRUTI*, 2005.

[50] Guofei Gu et al. BotSniffer: Detecting Botnet Command and Control Channels in Network Traffic. In *Proc. of NDSS'08*.

[51] James Binkley et al. An algorithm for anomaly-based botnet detection. *SRUTI Workshop*, 2006.

[52] Mark Allman et al. Fighting Coordinated Attackers with Cross-Organizational Information Sharing. In *Proc. of ACM HotNets'05*, 2005.

[53] Patrick McDaniel et al. Enterprise Security: A Community of Interest Based Approach. In *Proc. of NDSS*, 2006.

[54] Thomas Karagiannis et al. Profiling the End Host. In *Proc. PAM*, 2007.

[55] Thorsten Holz et al. Measurements and Mitigation of Peer-to-Peer based Botnets: A Case Study on Storm Worm. In *Proc. of USENIX LEET*, 2008.

[56] Min Fang, Narayanan Shivakumar, Hector Garcia-Molina, Rajeev Motwani, and Jeffrey D. Ullman. Computing iceberg queries efficiently. In *Proc. of VLDB*, pages 299–310, NY, NY, August 24-27 1998.

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

[58] Stephanie Forrest, Anil Somayaji, and David. H. Ackley. Building diverse computer systems. In *Proc. of HotOS*, 1997.

[59] Terrence S. Furey, Nello Cristianni, Nigel Duffy, David W. Bednarski, Michel Schummer, and David Haussler. Support vector machine classification and validation of cancer tissue samples using microarray expression data. *Bioinformatics*, 16(10):906–914, 2000.

[60] Sudipto Guha. Space Efficiency in Synopsis Construction Algorithms. In *Proc. of VLDB*, pages 409–420, 2005.

[61] Sudipto Guha and Boulos Harb. Approximation algorithms for wavelet transform coding of data streams. In *Proc. of SODA*, pages 273–279, 2006.

[62] Patrick Haffner, Subhabrata Sen, Oliver Spatscheck, and Dongmei Wang. Acas: Automataed construction of applicatin signatures. In *SIGCOMM MineNet Workshop*, August 2005.

[63] IANA. *IANA Port Numbers*. `http://www.iana.org/assignments/port-numbers`.

[64] Marios Iliofotou, Prashanth Pappu, Michalis Faloutsos, Michael Mitzenmacher, Sumeet Singh, and George Varghese. Network monitoring using traffic dispersion graphs (tdgs). In *ACM IMC*, October 2007.

[65] James R. Binkley et al. An algorithm for anomaly-based botnet detection. In *Proc. of USENIX SRUTI*, 2006.

[66] T. Karagiannis, D. Papagiannaki, N. Taft, and M. Faloutsos. Profiling the end host. In *PAM*, April 2007.

[67] Thomas Karagiannis, Andre Broido, Michalis Faloutsos, and kc claffy. Transport layer identification of p2p traffic. In *ACM IMC*, October 2004.

[68] Thomas Karagiannis, Konstantina Papagiannaki, and Michalis Faloutsos. Blinc: Multilevel traffic classification in the dark. In *ACM SIGCOMM*, August 2005.

[69] Panagiotis Karras and Nikos Mamoulis. The Haar+ Tree: a Refined Synopsis Data Structure. In *Proc. of the IEEE 23rd ICDE*, April 2007.

[70] Zeus Kerravala. Enterprise Networking and Computing: the Need for Configuration Management. In *Yankee Group Report*, January 2004.

[71] Daniel Kifer, Shai Ben-David, and Johannes Gehrke. Detecting changes in data streams. In *Proc. of VLDB*, 2004.

[72] Hyunchul Kim, Marina Fomenkov, kc claffy, Nevil Brownlee, Dhiman Barman, Michalis Faloutsos, and Kiyoung Lee. Internet traffic classification demystified: Myths, caveats, and the best practices. Technical report, CAIDA, 2008.

[73] Jon Kleinberg. Bursty and hierarchical structure in streams. In *Proc. of the 8th ACM SIGKDD*, 2002.

[74] C. Kreibich, A. Warfield, J. Crowcroft, S. Hand, and I. Pratt. Using Packet Symmetry to Curtail Malicious Traffic. 2005.

[75] Anukool Lakhina, Mark Crovella, and Christophe Diot. Mining anomalies using traffic feature distributions. In *ACM SIGCOMM*, pages 217–228, August 2005.

[76] Laks V. S. Lakshmanan, Raymond T. Ng, Christine Xing Wang, Xiaodong Zhou, and Theodore Johnson. The generalized mdl approach for summarization. In *VLDB*, pages 766–777, 2002.

[77] Zhu Li, Ruixi Yuan, and Xiaohong Guan. Accurate Classification of the Internet Traffic Based on the SVM Method. In *ICC*, June 2007.

[78] Justin Ma, Kirill Levchenko, Christian Kreibich, Stefan Savage, and Geoffrey M. Voelker. Unexpected means of protocol inference. In *ACM IMC*, 2006.

[79] Yossi Matias, J.S. Vitter, and M. Wang. Wavelet-Based Histograms for Selectivity Estimation. In *Proc. of ACM SIGMOD'98*, 1998.

[80] Anothony McGregor, Mark Hall, Perry Lorier, and James Brunskill. Flow clustering using machine learning techniques. In *PAM*, April 2004.

[81] Andrew Moore and Konstantina Papagiannaki. Toward the accurate identification of network applications. In *PAM*, April 2005.

[82] Andrew Moore and Denis Zuev. Internet traffic classification using bayesian analysis techniques. In *ACM SIGMETRICS*, June 2005.

[83] David Moore, Ken Keys, Ryan Koga, Edouard Lagache, and KC Claffy. CoralReef software suite as a tool for system and network administrators. In *USENIX Lisa*, 2001.

[84] David Moore, Vern Paxson, Stefan Savage, Colleen Shannon, Stuart Staniford, and Nicholas Weaver. Inside the slammer worm. *IEEE Security and Privacy*, 1(4):33–39, July 2003.

[85] S. Muthukrishnan. Subquadratic algorithms for workload-aware Haar wavelet synopses. In *Proc. of FSTTCS*, 2005.

[86] N. Ianelli and A. Hackworth. Botnets as a vehicle for online crime. In *CERT RFC 1700*, 2005.

[87] Thuy T.T. Nguyen and Grenville Armitage. Training on multiple sub-flows to optimise the use of machine learning classifiers in real-world ip networks. In *IEEE LCN*, November 2006.

[88] Thuy T.T. Nguyen and Grenville Armitage. A survey of techniques for internet traffic classification using machine learning. *IEEE Communications Surveys and Tutorials, to appear*, 2008.

[89] Byung-Chul Park, Young J. Won, Myung-Sup Kim, and James W. Hong. Towards automated application signature generation for traffic identification. In *IEEE NOMS*, April 2008.

[90] Vern Paxson. Bro: A System for Detecting Network Intruders in Real-Time. *Computer Networks*, 1999.

[91] Vladimir Pervouchine and Graham Leedham. Extraction and analysis of forensic document examiner features used for writer identification. *Pattern Recognition*, 40(3):1004–1013, March 2007.

[92] P.J.Harrison. Exponential smoothing and short-term sales forecasting. *Management Science*, 13(11):821–842, 1967.

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

[94] Frederick Reiss, Minos Garafalakis, and Joseph Hellerstein. Compact Histograms for Hierarchical Identifiers. In *Proc. of VLDB*, 2006.

[95] M. Roesch. Snort network intrusion detection system. `http://www.snort.org/`.

[96] 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 *ACM IMC*, October 2004.

[97] Roy A. Maxion. *Use of Diversity as a Defense Mechanism*, 2002.

[98] Sunita Sarawagi. Explaining differences in multidimensional aggregates. In *Proc. of VLDB*, pages 42–53, Scotland,UK, 1999.

[99] Sunita Sarawagi. Explaining differences in multidimensional aggregates. In *The VLDB Journal*, pages 42–53, 1999.

[100] Sunita Sarawagi, Rakesh Agrawal, and Nimrod Megiddo. Discovery-driven exploration of olap data cubes. In *Proc. of EDBT*, pages 168–182, March 1998.

[101] Sudarshan S.Chawathe. Differencing data streams. In *Proc. of the Database Engineering and Applications Symposium (IDEAS)*, Montreal,Canada, July 2005.

[102] Subhabrata Sen, Oliver Spatscheck, and Dongmei Wang. Accurate, scalable in-network identification of p2p traffic using application signatures. In *WWW*, May 2004.

[103] Stanislav Shalunov and Benjamin Teitelbaum. *Internet2 NetFlow Statistics*. `http://netflow.internet2.edu`.

[104] Stanislav Shalunov and Benjamin Teitelbaum. *NFSTAT*. `http://netflow.internet2.edu/weekly/nfstat.pdf`.

[105] Jin-Qiao Shi, Bin xing Fang, Bin Li, and Fu liang Wang. Using support vector machine in traffic analysis for website recognition. In *International Conference on Machine Learning and Cybernetics*, August 2004.

[106] S.Hill, D.Agarwal, R.Bell, and C.Volinsky. Building an effective representation for dynamic graphs. *Journal of Computational and Graphical Statistics*, 15:584–608, 2006.

[107] Suresh N. Chari et al. BlueBox: A policy-driven, host-based intrusion detection system. 6:173–200, 2003.

[108] Nigel Williams, Sebastian Zander, and Grenville Armitage. Evaluating machine learning algorithms for automated network application identification. Technical Report 060401B, CAIA, April 2006.

[109] 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):7–15, October 2006.

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

[111] Young J. Won, Byung-Chul Park, Hong-Taek Ju, Myung-Sup Kim, and James W. Hong. A hybrid approach for accurate application traffic idenficiation. In *IEEE/IFIP E2EMON*, April 2006.

[112] WorldCup 1998. `http://ita.ee.lbl.gov/html/contrib/WorldCup.html`.

[113] Sebastian Zander, Thuy Nguyen, and Grenville Armitage. Automated traffic classification and application identification using machine learning. In *IEEE LCN*, November 2005.

[114] Yin Zhang, Sumeet Singh, Subhabrata Sen, Nick Duffield, and Carsten Lund. Online identification of hierarchical heavy hitters: Algorithms, evaluation and applications. In *Proc. of ACM IMC'04*, October 2004.

[115] Yunyue Zhu and Dennis Shasha. Efficient elastic burst detection in data streams. In *Proc. of ACM SIGKDD'03*, pages 336–345, New York, NY, 2003.