AUTOMATED NETWORK ANOMALY DETECTION
WITH LEARNING, CONTROL, AND QoS MITIGATION

by

DENNIS IPPOLITI

M.S. University of Colorado, Colorado Springs, 2006

A dissertation submitted to the Graduate Faculty of the
University of Colorado at Colorado Springs
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy
Department of Computer Science
2013
This dissertation for the Doctor of Philosophy degree by

Dennis Ippoliti

has been approved for the

Department of Computer Science

by

__________________________________________________________
Xiaobo Zhou, Chair

__________________________________________________________
Terry Boult

__________________________________________________________
Edward Chow

__________________________________________________________
Song Fu

__________________________________________________________
Chuan Yue

Date
Ippoliti, Dennis (Ph. D., Computer Science)

Automated Network Anomaly Detection with Learning, Control, and QoS Mitigation

Dissertation directed by Professor Xiaobo Zhou

Anomaly detection is a challenging problem that has been researched within a variety of application domains. In network intrusion detection, anomaly based techniques are particularly attractive because of their ability to identify previously unknown attacks without the need to be programmed with the specific signatures of every possible attack. There is a significant body of work in anomaly based intrusion detection applying statistical analysis, data-mining, and machine learning disciplines. However despite more than two decades of active research, there is a striking lack of anomaly based systems in commercial use today. Many of the currently proposed anomaly based systems do not adequately address a series of challenges making them unsuitable for operational deployment. In existing approaches, every step of the anomaly detection process requires expert manual intervention. This dependence makes developing practical systems extremely challenging.

In this thesis, we integrate the strengths of machine learning and quality-of-service mitigation techniques for network anomaly detection, and build an operationally practical framework for anomaly-based network intrusion detection. We propose methods for self-adaptive, self-tuning, self-optimizing, and automatically responsive network anomaly detection. In specific, we propose and develop methods for adaptive input normalization adjusting scaling parameters online based on evolving values in observed traffic patterns, adaptive algorithms for flow-based network anomaly detection that respond to feedback to account for concept drift, and evolving methods for aggregated alert correlation that consolidate individual alarms into network events. We propose and design a model for dictating optimal performance in an anomaly detection system and reinforcement learning algorithms for automated tuning and optimization and a confidence forwarding model to support automated response. Furthermore, we develop a fair bandwidth sharing and delay differentiation mechanism for scalable automated response that insulates network resources from malicious traffic while minimizing collateral damage. We develop a prototype network anomaly detection system that integrates the proposed and developed techniques. We evaluate developed approaches using the 1999 Knowledge Discovery and Data-mining Cup and MAWI Lab
datasets, but also we create a new dataset based on a combination of live network traces and controlled simulated data injects. Results demonstrate the effectiveness and capability of automated means.
Dedication

This thesis is dedicated to my father, Dennis Ippoliti Sr., who taught me my first line of code and the magic of “xyzzy” many years ago. Something Happened!
Acknowledgements

I would like to thank my advisor, Dr. Xiaobo Zhou for his support and guidance during both my Master's and Ph. D. programs. His continuous encouragement and understanding as the Air Force repeatedly moved me from station to station and country to country as I attempted this journey were invaluable to me. I would also like to thank my Ph. D. committee members Dr. Boult, Dr Chow, Dr. Fu, and Dr. Yue for their help and guidance. Their feedback at the proposal stage of this work tremendously improved the quality of this product. I would also like to thank Dr. Chamillard and Dr. Kalita for enabling me to complete course work from around the world.

My most sincere thanks goes to my wife Teresa. None of my accomplishments would be possible without her patience, assistance, and love. She believed in me when I didn't.

Finally, this research and dissertation were supported in part by the US National Science Foundation CAREER Award CNS-0844983 and research grant CNS-0720524.
Contents

Acknowledgements .......................................................................................................................... vi

Contents ........................................................................................................................................ vii

List of Figures ............................................................................................................................... xiv

Chapter 1. ..................................................................................................................................... 1

1.1 Network Anomaly Detection ............................................................................................... 1

1.2 Technical Issues and Challenges ...................................................................................... 3

1.3 Research Objectives ........................................................................................................... 6

1.3.1 Self-Adaptive .................................................................................................................. 6

1.3.2 Self-Tuning ..................................................................................................................... 7

1.3.3 Self-Optimizing ............................................................................................................. 7

1.3.4 Automated Response .................................................................................................... 7

1.4 Technical Approaches ....................................................................................................... 8

1.4.1 Statistical Counting for Augmented Traffic Flows: ................................................... 8

1.4.2 Dynamic Input Normalization: .................................................................................... 8

1.4.3 Feedback Adaptive Detection Models: ....................................................................... 9

1.4.4 Confidence Forwarding: ............................................................................................. 9

1.4.5 Reinforcement Learning for Dynamic Tuning and Control ....................................... 9

1.4.6 Distinct Value Estimation for Evolving Alert Correlation ........................................ 10

1.4.7 Bandwidth Sharing and Delay Differentiation Mechanism for Scalable Response .... 10
1.5 Research Contributions ...........................................................................................................12

1.6 Thesis Roadmap ......................................................................................................................13

Chapter 2. .............................................................................................................................................15

Related Work...........................................................................................................................................15

2.1 Classification of Anomaly Detection Systems .........................................................................15

2.1.1 Host Based Anomaly Detection ..........................................................................................16

2.1.2 Statistical Approaches to Network Anomaly Detection ......................................................18

2.1.3 Rule Based Approaches to Network Anomaly Detection: ..................................................19

2.1.4 Network Anomaly Detection with Machine Learning .........................................................21

2.1.5 Hybrid Techniques ..............................................................................................................23

2.1.6 Dynamic Anomaly Detection ..............................................................................................24

2.2 Flow Based Data Collection ....................................................................................................28

2.3 Intrusion Detection Performance Models ................................................................................28

2.4 Open Issues in Network Anomaly Detection ........................................................................30

2.4.1 False Alarm Rate ..............................................................................................................30

2.4.2 Relationship between anomaly and intrusion .............................................................30

2.4.3 Correlation of attacks .......................................................................................................31

2.4.4 Tuning and optimization .................................................................................................31

2.4.5 Evaluation of approaches ...............................................................................................31

2.5 Attack mitigation though quality of service allocation ...........................................................32

Chapter 3. .............................................................................................................................................33

Data Sets for System Evaluation ...........................................................................................................33

3.1 Introduction ...................................................................................................................................33
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1.1.2</td>
<td>Feedback Based Threshold Adaptation:</td>
<td>64</td>
</tr>
<tr>
<td>5.1.1.3</td>
<td>Dynamic Pruning</td>
<td>66</td>
</tr>
<tr>
<td>5.1.1.4</td>
<td>A-GHSOM Prediction Confidence</td>
<td>67</td>
</tr>
<tr>
<td>5.1.2</td>
<td>A-GHSOM Performance Evaluation</td>
<td>69</td>
</tr>
<tr>
<td>5.1.2.1</td>
<td>Experimental Setup</td>
<td>69</td>
</tr>
<tr>
<td>5.1.2.2</td>
<td>Baseline Performance</td>
<td>69</td>
</tr>
<tr>
<td>5.1.2.3</td>
<td>Impact of Dynamic Input Normalization</td>
<td>70</td>
</tr>
<tr>
<td>5.1.2.4</td>
<td>Impact of On-Line Adaptation</td>
<td>72</td>
</tr>
<tr>
<td>5.1.2.5</td>
<td>Summary and Discussion</td>
<td>75</td>
</tr>
<tr>
<td>5.2</td>
<td>Adaptive Support Vector Machine</td>
<td>75</td>
</tr>
<tr>
<td>5.2.1</td>
<td>Background and overview</td>
<td>76</td>
</tr>
<tr>
<td>5.2.1.1</td>
<td>One-class support vector machines</td>
<td>76</td>
</tr>
<tr>
<td>5.2.1.2</td>
<td>Sequential Minimal Optimization</td>
<td>78</td>
</tr>
<tr>
<td>5.2.2</td>
<td>Adaptive SVM Auxiliary Training Set</td>
<td>78</td>
</tr>
<tr>
<td>5.2.2.1</td>
<td>Adding a new vector to the auxillary set</td>
<td>79</td>
</tr>
<tr>
<td>5.2.2.2</td>
<td>Removing a vector from the auxillary set</td>
<td>81</td>
</tr>
<tr>
<td>5.2.3</td>
<td>Support Vector Machine Confidence Rating for Anomaly Detection</td>
<td>82</td>
</tr>
<tr>
<td>5.2.4</td>
<td>A-SVM Performance Evaluation</td>
<td>84</td>
</tr>
<tr>
<td>5.2.4.1</td>
<td>Experimental Setup</td>
<td>84</td>
</tr>
<tr>
<td>5.2.4.2</td>
<td>Efficiency of maintaining Auxiliary Set</td>
<td>85</td>
</tr>
<tr>
<td>5.2.5</td>
<td>Impact of Depth Parameter on SVM Confidence</td>
<td>87</td>
</tr>
<tr>
<td>5.2.6</td>
<td>Baseline Performance</td>
<td>89</td>
</tr>
<tr>
<td>5.2.7</td>
<td>Impact of Dynamic Input Normalization</td>
<td>90</td>
</tr>
</tbody>
</table>
5.2.8 Impact of Online Adaptation ............................................................................................... 91
5.2.9 Summary and Discussion .................................................................................................... 94
5.3 Alert Correlation Engine ....................................................................................................... 95
5.3.1 Generating Correlation Records by Alert Aggregation ....................................................... 96
5.3.2 Evaluation of Alert Correlation Engine............................................................................. 100
5.3.2.1 Experiment Setup ....................................................................................................... 100
5.3.2.2 Impact on Detection Model Performance ..................................................................... 100
5.4 Evaluation of long-term adaptation ....................................................................................... 102
5.4.1 Experiment Setup .............................................................................................................. 102
5.4.2 Individual Performance of Base MAWI Detectors ............................................................ 102
5.4.3 Baseline Performance of A-SVM and A-GHSOM during long term operation ............... 103
5.4.4 Impact of Dynamic Scaling on A-GHSOM and A-SVM long-term Performance ........ 105
5.4.5 Impact of On-Line Adaptation on Long-term A-SVM and A-GHSOM Performance ..... 106
5.4.6 Impact of Adaptive Alert Aggregation on long term A-SVM and A-GHSOM performance 108
5.5 Summary and Discussion ...................................................................................................... 111

Chapter 6. Self-Tuning, Self-Optimizing Anomaly Detection ......................................................... 113
6.1 Introduction ........................................................................................................................... 113
6.2 Tuning Anomaly Detection Systems ..................................................................................... 115
6.3 Optimizing Anomaly Detection Systems .............................................................................. 116
6.3.1 A Model for Optimal Objective for Intrusion Detection .................................................. 117
6.3.1.1 Anomaly Detection Performance CONFIDENCE .................................................. 118
7.3.1 Experimental Setup ........................................................................................................... 155

7.3.1.1 Packet Scheduling Algorithms ...................................................................................... 155

7.3.1.2 Performance of PID Control Buffer Management ...................................................... 157

7.4 Summary and Discussion ...................................................................................................... 159

Chapter 8. ....................................................................................................................................... 160

Conclusion ....................................................................................................................................... 160

8.1 Summary and Discussion ....................................................................................................... 160

8.2 Future Work .......................................................................................................................... 163

8.3 Publications ........................................................................................................................... 164

Bibliography ................................................................................................................................... 166
List of Figures

Figure 1-1 Basic Elements of the Anomaly Detection Process ................................................................. 3
Figure 1-2: Relationship between theoretical and technical concepts ..................................................... 11
Figure 2-1 Classification of Surveyed Detection Methods ..................................................................... 17
Figure 4-1: Example Source Host I ......................................................................................................... 46
Figure 4-2: Example Source Host II ....................................................................................................... 47
Figure 4-3: Example Source Host III ...................................................................................................... 48
Figure 4-4: Count Min Sketch Schematic .............................................................................................. 49
Figure 4-5: Cumulative Percentage of Flow Duration ............................................................................ 50
Figure 4-6: Impact of Augmentation History Duration on Basic Performance Metrics ....................... 55
Figure 4-7: Example Adapted Scaling Values ......................................................................................... 57
Figure 5-1: GHSOM After Off-Line Training ......................................................................................... 62
Figure 5-2: Example Quantization Error of two Input Vectors ............................................................... 64
Figure 5-3: Impact of Dynamic Input Normalization on the baseline A-GHSOM performance .......... 71
Figure 5-4: Impact of Adaptation on Dynamically Scaled A-GHSOM – KDD Data ............................. 73
Figure 5-5: Impact of Adaptation on Dynamically Scaled A-GHSOM – TD-SIM Data ....................... 74
Figure 5-6: Efficiency of Adding Vector to Auxiliary Set ...................................................................... 85
Figure 5-7: Efficiency of Deleting a Vector from Auxiliary Set .............................................................. 86
Figure 5-8: Impact of Depth Parameter on SVM Confidence Forwarding ............................................ 88
Figure 5-9: Impact of Dynamic Normalization on A-SVM Performance .............................................. 90
Figure 5-10: Impact of Adaptation on Dynamically Scaled A-SVM – KDD Data ................................. 92
Figure 5-11: Impact of Adaptation on Dynamically Scaled A-SVM – TD-SIM Data ............................ 93
Figure 5-12: Alert Aggregation Data Structures .................................................................................... 99
Figure 5-13: Impact of Aggregated Alert Correlation on Scaled and adapted detection engines ......... 101
Figure 5-14: Baseline Long-term performance of A-GHSOM and A-SVM models ............................. 104
Figure 5-15: Impact of Dynamic Normalization on long-term A-SVM and A-GHSOM performance ... 106
Figure 5-16: Impact of On-Line Adaptation to long-term A-SVM and A-GHSOM Performance ........ 107
Figure 5-17: Impact of Adaptation + Alert Aggregation on long-term A-SVM and A-GHSOM Performance
.....................................................................................................................................................................109

Figure 5-18: Impact of integrated model.....................................................................................................110

Figure 6-1: Classification of Events Based on Description Vectors.............................................................120

Figure 6-2: Anomalous and Benign Sets.....................................................................................................121

Figure 6-3: Normal and Malicious Sets.....................................................................................................121

Figure 6-4: Intuitive tradeoff between false positive and false negative .....................................................122

Figure 6-5: Intuitive tradeoff between false positive, false negative, and confidence forwarding ..........123

Figure 6-6: System Controller Schematic ...................................................................................................125

Figure 6-7: Impact of adjusting malicious confidence threshold using TD-SIM Dataset .........................134

Figure 6-8: Impact of adjusting benign confidence threshold using TD-SIM Dataset .........................135

Figure 6-9: Impact of adjusting malicious confidence threshold using KDD Dataset .........................136

Figure 6-10: Impact of adjusting benign confidence threshold using KDD Dataset .............................137

Figure 6-11: Performance of Statically Tuned Model on MAWI data.........................................................138

Figure 6-12: Dynamic Tuning on the TD-SIM Dataset (Tradeoff Parameter Forward – SVM) .............139

Figure 6-13: Dynamic Tuning on the TD-SIM Dataset (Tradeoff Parameter Forward - GHSOM) .........140

Figure 6-14: Dynamic Tuning on the TD-SIM Dataset (Precision / Recall Tradeoff - SVM) .....................141

Figure 6-15: Dynamic Tuning on the TD-SIM Dataset (Precision / Recall Tradeoff - GHSOM) ..........142

Figure 6-16: Dynamic Tuning on the MAWI Set (SVM) ...........................................................................143

Figure 6-17: Dynamic Tuning on the MAWI Set (GHSOM) ....................................................................145

Figure 7-1: Automated Response Approach ...............................................................................................149

Figure 7-2: The Performance of VPS when overall workload changes from 55% to 100%. ...................156

Figure 7-3: The performance of VPS when Class-1’s workload changes from 10% to 90% of the overall
workload......................................................................................................................................................156

Figure 7-4: The behaviors of VPS scheduling schemes for FBS-DD provisioning in different sampling
intervals ..........................................................................................................................................................157

Figure 7-5: The impact of the control-theoretic buffer management on FBS-DD provisioning .........158
Chapter 1.

Introduction

1.1 Network Anomaly Detection

Revolutions in communications and information technology have given birth to a virtual world. With the growth of cyberspace and its potential, there has been a subsequent change in every facet of daily life. In today’s information age, everything we do relies on access to information networks. The internet is used in the home to shop, pay bills, and stay connected via social networks. National infrastructure relies on information networks to deliver oil and gas, power and water. They support hospitals and schools, public transportation and air traffic control. Businesses are relying more and more on e-commerce. As the use of cyberspace grows, the need to protect it also grows.

Attacks against computer networks are increasing at an alarming rate. In the last five years many major institutions have experienced disruptions in service due to cyber-attacks. Facebook, Twitter, Visa, MasterCard, and Google have all been victims. Akamai, the world’s largest cloud computing company reported that they experienced more denial of service attempts in the fourth quarter of 2010 than in the previous three quarters combined. One of the most significant trends in cyber-attacks is the prevalence of zero-day attacks. A zero day attack occurs when a vulnerability is exploited before the software developer or administrator is aware the vulnerability exists. Therefore, detection systems are not programmed to identify the attack and software patches have not been developed or deployed.

Not only are attacks increasing in number, there effect is becoming potentially devastating. In 2003 worms such as SoBig and Klez caused $80 billion in cleanup costs and in 2008, cyber criminals stole over 1 trillion dollars in intellectual property. In 2009 the conficker worm infected the armed forces network in France forcing aircraft at several airbases to be grounded because their flight plans could not be downloaded. In 2007, researchers at the department of energy demonstrated the potential of cyber-attacks by hacking into a power generator and making it destroy itself. According to Senator Carl Levin the Chairman of the Senate Committee on Armed Services “cyber weapons are approaching weapons of mass destructions in their effect.”
Existing detection methods, based primarily on identifying and mitigating specific attack signatures are by themselves are inadequate. Signature based methods are ineffective against zero-day attacks. Furthermore, even when similar attack signatures are known, attackers use techniques that disguise the threat. In 2010 Google, Adobe and as many as 33 other companies were attacked using tactics that combined encryption, stealth programming and a vulnerability in Internet Explorer. Despite exploiting vulnerabilities similar to those discovered months earlier, the extremely sophisticated use of encryption prevented discovery by traditional means.

Even if attacks are identified easily, existing response methods are often too slow or impractical causing collateral damage to legitimate customers. In March 2011, CODERO, a cloud and server hosting service for small and medium business, was assaulted by a denial of service attack. Although the attack was specifically targeting a single customer, service to all their customers was disrupted. Despite quickly identifying the /24 network of the target victim, no automated means to analyze and defend affected users was available. Engineers manually redesigned routing tables attempting to isolate malicious traffic from benign, gradually restoring service to all users except those on the same /24 as the targeted system and then eventually to all users.

With the elevated damage caused by intrusions, the increase in zero-day attacks, and the growing sophistication used by attackers, anomaly based methods are well suited to the current threat environment. For Network Intrusion Detection, anomaly based techniques are particularly attractive. In a traditional signature based system, the signature for each new attack must be identified and programmed before a detection system can be sensitive to it. Anomaly based systems do not rely on signatures. They maintain models of normal behavior and heuristically flag abnormal conditions as potential attacks. In 1987 Denning proposed the hypothesis that security violations could be detected by monitoring a system's audit records for abnormal patterns of system usage [32]. Anomaly detection schemes are an important step forward in network defense. They excel at identifying zero-day attacks, previously unknown, or well-disguised attack methods. However despite the longevity of research, there are very few anomaly detectors in operational use today.

In this chapter we discuss the elements of network anomaly detection and identify technical issues and challenges involved with the process. We introduce our choices for technical approaches to mitigating
these challenges. Next we present our research focus and contributions. We conclude with a roadmap to the remainder of the thesis.

1.2 Technical Issues and Challenges

A detection system is only effective if the alerts it generates are timely, accurate, and provide useful actionable information to the security team. Unfortunately, anomaly based schemes detect anomalies, not attacks. Having detected an anomaly rather than a well understood attack signature, they are generally unable to provide security teams with information that can be used to guide response actions. Maintaining accurate systems, interpreting alerts, and responding to potential threats requires extensive expert manual intervention. The ability to capitalizing on the advantages of anomaly detection while at the same time providing a useful tool to security teams is hindered by a number of technical challenges.

Figure 1 identifies the basic steps that must be accomplished in order to use an anomaly detection system for network defense. Existing research in anomaly based network intrusion detection is not evenly distributed among each of the steps. The majority of existing work focuses on identifying attacks by detecting anomalies [42, 136, 139, 154, 156]. Very little work is dedicated to other elements of the...
process. While a body of work has been done on distinguishing between legitimate anomalies and attack anomalies, most of this work involves reducing false positive rates [90, 156]. The work in [26] examines the act of sanitizing training data used to develop base traffic models. In [7] and [124] discuss approaches to classifying attacks in anomaly based detection systems. There are also proposals for active systems that automatically respond to attacks. However these systems primarily involve disallowing anomalous requests [125, 140]. They lack specific response methods tailored to the identified threat level. Combining existing proposals into a comprehensive anomaly detection solution is hindered by the level of operator involvement required. In current systems, each of these elements requires significant human intervention delaying adequate response.

Collect Network Information: Nearly all network intrusion detection systems require some form of pre-processing data. Unless the system is going to perform its prediction by inspecting each packet in its entirety, there is some form of pre-processing or data presentation that must be performed. In an attempt to find known attacks or unusual behavior, modern intrusion detection systems traditionally inspect the contents (payload) of every packet. The problem of packet inspection, however, is that it is hard, or even impossible, to perform it at the speed of multiple Gigabits per second (Gbps). Additionally, Opaque traffic that is compressed or encrypted may make packet content inspection impossible.

Maintain current model of normal behavior: As the normal behavior of a monitored system evolves, the underlying base model used by an anomaly detection system must be updated. This is traditionally accomplished periodically off line using training data that has been either sanitized by an expert, or certified to be either clean or at least noisy (mostly clean with very few attack instances). If the data used to update the model is not sanitized, sophisticated attacks can fool anomaly detectors into accepting attack traffic as normal.

Identify normal and anomalous activity: Numerous methods have been proposed to accomplish the task of anomaly detection. In these approaches exists a variety of control parameters and settings that must be tuned by an expert familiar with both the specific approach and the operating environment of the
monitored systems. Failure to properly tune these parameters can impact the effectiveness of the approach. Because these settings are often environment specific, they must be tuned for each potential application. Additionally, as the operating environment evolves, systems can become out of tune. There is an unmet need for systems to be able to dynamically tune based on their current operating and threat environment.

*Categorize anomalous activity as attack or legitimate*: Anomaly detection systems classify events as either normal or anomalous. This is not the same thing as classifying them as malicious or benign. In regards to network anomaly detection there are four possible conditions after a detection system has made a prediction concerning a network event

1. Benign and not anomalous
2. Malicious and anomalous.
3. Malicious but not anomalous
4. Benign but anomalous

Existing systems excel at identifying events that belong to categories 1 and 2. They assume a direct relationship between anomalous and malicious. However, effectively managing events that belong to categories 3 and 4 is an ongoing challenge faced by anomaly detectors. Stealthy attacks that operate over time can gradually force anomaly detectors to accept malicious behavior as normal. On the other hand, network upgrades, legitimate use of new software, and normal evolution of user behavior are all examples of circumstances that while anomalous compared to recorded models, are nonetheless benign. By flagging legitimate traffic that appears to be anomalous as attack traffic, the security operator is forced to address excessive false positive events. Anomaly based schemes need to be able to learn how to relate anomalies with intrusion attempts.

*Correlate events to guide response*: Performing attack classification is an important step to attack mitigation and response. In order for an operator to effectively respond to a potential threat, they must have more than a generic anomaly alert. Modern networks are constantly under attack. While many of
these attacks are not relevant (i.e. an old attack targeting an already patched system), others must be addressed immediately by a variety of attack specific means (patching vulnerable systems, terminating sessions, updating routing tables, rate limiting traffic from specific sources, etc...). Performing attack correlation in signature based systems is trivial. It is merely a matter of matching current behavior with a set of clearly analyzed and defined patterns. Existing anomaly based approaches generally lack sufficient information and methods to properly perform this task. By not providing attack correlation data, security operators must manually research each alert to identify potential threat and take appropriate action.

**Take appropriate action:** The last step in the process is for the operator to take action. Most intrusion detection systems are passive in that they raise alerts only. Others are active and take predetermined steps depending on the particular attack. However these approaches generally rely on trained professionals to determine appropriate steps and in many cases are limited in the actions that they are able to take, usually resorting to blocking access to resources or terminating sessions. Due to the inherent uncertainty between an anomaly and malicious intent, blocking, terminating, or disallowing all anomalous events is too excessive. However, by waiting until operators have been able to research each alert, it may already be too late by the time responses are initiated.

### 1.3 Research Objectives

The objective of this research is to advance the field of intrusion detection by contributing algorithms and methodologies for automating the anomaly detection process. We study the applicability of machine learning methodologies to each of the elements of the detection cycle in Figure 1 and provide a novel framework that demonstrates four important features: self-adaptive, self-tuning, self-optimizing, and automated response there by significantly reducing the need for operator intervention.

#### 1.3.1 Self-Adaptive

We develop efficient methods for flow based data collection that enable detectors to identify a variety of network anomalies without the need for packet inspection. We propose methods for dynamically scaling
input data to aid engines in identifying true anomalies. We explore methods for anomaly detection models that dynamically adapt themselves on-line with little or no expert operator input. Our proposed methods adapt to changes in normal operations, changes in threat environment, and changes to appropriate response actions.

1.3.2 Self-Tuning

There are numerous approaches to identifying anomalies in network traffic. Within each of these approaches exists a platform dependent set of control parameters. Tuning these parameters require extensive knowledge of both the detection model and environment that the model will be operating in. We develop approaches for automatically tuning detection systems in any environment without requiring expert understanding of the underlying detection scheme.

1.3.3 Self-Optimizing

Optimizing a detection system involves tuning the system to achieve an optimal objective. In intrusion detection systems, the optimal objective is constantly changing in response to changes in operating environment and threat conditions. We propose and justify a model for defining optimal performance in an anomaly detection system and then devise algorithms for self-optimizing capability.

1.3.4 Automated Response

Existing, anomaly detection systems excel at detecting anomalies not attacks. Because of this condition, relative to signature based systems, there is inherent error in attack predictions made by anomaly detection systems. We discuss algorithms for calculating confidence measures in attack predictions. We provide a mechanism for automated response actions. We develop Quality of Service algorithms fed by evolving alert correlation approaches, protecting resources while limiting collateral damage to legitimate traffic.

We demonstrate the effectiveness of our approaches by proposing a prototype integrated detection and response framework. The integrated framework will apply broad identification of network events into categories of normal, attack, and confidence filtering. Attack predictions will automatically aggregated into evolving correlated alerts. We evaluate the prototypes ability to demonstrate self-adaptive, self-tuning, self-optimizing and automated response capability by conducting extensive experiments on the KDD99 Intrusion Detection Data set, an enterprise network data set developed by combining actual network trace data and simulated network attacks, and labeled backbone link data set consisting of 100% real trace data.
We propose novel Quality of Service algorithms to support automated response that can be used to apply scalable response actions to normal and attack predictions with low confidence ratings.

1.4 Technical Approaches

In this section we briefly introduce the technical approaches we have used to achieve our research objectives. We discuss algorithms for a complete detection and initial response system. We propose a data collection approach that uses statistical counting approaches and dynamic input normalization to provide detection engines with distinguishable data on-line. We propose two feedback adaptive detection engines based on machine learning methodologies, one based on an adaptive version of the Growing Hierarchical Self Organizing Map and one based on an on-line adaptive Support Vector Machine. Additionally, we propose and justify methods for implementing prediction confidence forwarding. We discuss a reinforcement learning based controller for self-tuning self-optimizing anomaly detection. We also propose a mechanism for evolving alert correlation that uses distinct value estimation to aggregate related alerts. Finally, we propose algorithms for fair bandwidth sharing and delay differentiation that can be used for scalable threat response.

1.4.1 Statistical Counting for Augmented Traffic Flows:

Several anomalies are visible by monitoring volume and distribution metrics. We propose a method of augmenting basic flow records with an array of additional information that directly provides volume metrics, but also allows detectors to infer information about distribution changes in specific fields. Each generated flow is identified by the 5-tuple \{src_addr, src_port, dst_addr, dst_port, protocol\}. Each flow record also contains metrics specific to that flow (duration, packet count, byte count, etc…). When a flow record is generated, we use statistical sketches to augment that record with estimated flow, packet, and byte information from all other related flows previously observed.

1.4.2 Dynamic Input Normalization:

We propose an adaptive input normalization approach that will bring attention to the true difference between individual input vectors. In many approaches, data is normalized to account for differences in scale between two different data points. We propose an adaptive input normalization approach that automatically tunes scaling parameters online based on the observed traffic patterns.
1.4.3 Feedback Adaptive Detection Models:

We propose and develop two adaptive detection engines to evaluate our ideas. First we develop an Adaptive Growing Hierarchical Self Organizing Map. In a GHSOM, the size and dimensionality of the map architecture are determined during the training phase. The map is grown horizontally by adding rows and columns to the map to reduce quantization error. The map is grown vertically by adding child layers to parent layers that exceed the pre-specified maximum dimensionality of each map. We propose an enhanced A-GHSOM that is able to adapt its architecture on-line as events are processed. Additionally, it uses adaptive thresholds to classify dis-similar events mapped to similar portions of the model.

We also propose an Adaptive Support Vector Machine. Support vector machines use hyper-planes to divide samples into one of two classes. In the case of anomaly detection they identify normal and abnormal network events. Traditional SVMs are trained in batch and are not well suited to on-line adaptation. We propose an SVM model that maintains a dynamic on-line training set and uses the SMO algorithm to efficiently adapt itself during live operation. We propose algorithms for heuristically adding and removing training samples reducing the number of on-line passes the training algorithm must make to adapt.

1.4.4 Confidence Forwarding:

The need for confidence filtering in an anomaly detection system comes from the fact that compared to signature based systems they are known for a high false alarm rate. Additionally, sophisticated attack methods are capable of training anomaly detectors to accept malicious activity as normal. The inherent overlap between anomalous <-> benign and normal <-> malicious traffic requires attention. We argue the need for confidence filtering/forwarding in network anomaly detection systems. We develop and integrate novel confidence calculation approaches into our A-GHSOM and A-SVM detection models and we discuss mechanisms for using confidence as a tuning and optimization parameter and as a factor in initial response actions.

1.4.5 Reinforcement Learning for Dynamic Tuning and Control

We propose a reinforcement learning approach to automated tuning and optimization in anomaly detection systems. We justify and argue an optimization scheme that incorporates prediction confidence with precision and recall metrics. Our approach allows an operator to set performance goals and priorities in any two metrics and a reinforcement learning based controller attempts to meet these goals while
simultaneously optimizing performance in the third. In reinforcement learning, the learner is a decision-making agent that takes actions in an environment and receives reward (or penalty) for its actions in trying to solve a problem. The objective is for the agent to choose actions so as to maximize the expected reward over some period of time. When the state and or action spaces are extremely large or continuous performing state-action mappings is extremely challenging. We integrate neural network function approximation methods into our state space calculation. We propose an action-space discretization / aggregation method making our controller suitable for use in anomaly detection systems.

1.4.6 Distinct Value Estimation for Evolving Alert Correlation

Many anomaly detectors generate alerts on individual connections or flows. However, network wide anomalies are rarely isolated to a single flow. Attacks such as worms, scans, probing compromise attempts, and distributed denial of service attacks, will generate numerous flows per anomaly event. Individually identifying anomalous flows does not provide operators with accurate threat assessments. On the other hand, many anomaly detectors capable of identifying network wide events fail to identify the specific flows composing the anomalous event. In these instances, operators lack sufficient information to mount precision responses. In order to maximize effectiveness, a network operator analyzing anomaly alerts needs to be provided with actionable information in order to prioritize response actions. In this thesis we propose a novel evolving alert aggregation and correlation approach that generates correlated alerts used to guide initial response actions. We use efficient linear-counting distinct value estimation techniques to dynamically aggregate alerts on individual connections giving operators and downstream response engines an evolving picture of the current threat environment.

1.4.7 Bandwidth Sharing and Delay Differentiation Mechanism for Scalable Response

Most existing detection methods are passive methods that raise alerts but do not initiate response actions. Our proposed approaches identify attacks, assert prediction confidence, and provide correlated alerts. A key element of our model is the idea of confidence filtering. Anomaly detection systems, when compared to signature based systems, are inherently over sensitive. Not all anomalies are attacks. Our methods not only make predictions, but also learn to assess confidence in those predictions. When we have low confidence in the predictions that we make, there are one of two possible situations.
- We have predicted that an event is benign. However, we reasonably believe that it may be malicious.
- We have predicted that an event is malicious. However, we reasonably believe that it may be benign.

Traditional methods for responding to malicious activity involve various ways of completely denying access to the suspected attack. In this thesis we develop bandwidth sharing and delay differentiation mechanism that can be used to provide a quality of service based set of scalable response actions to an initial response engine. Figure 1-2: Relationship between theoretical and technical concepts illustrates the relationship between the theoretical and technical concepts proposed in this thesis.

![Figure 1-2: Relationship between theoretical and technical concepts](image-url)
1.5 Research Contributions

The need to protect cyber resources is ever increasing. Existing detection methods, based primarily on identifying and mitigating specific attack signatures are by themselves inadequate. Signature based methods are ineffective against zero-day attacks, and many new attack delivery methods apply sophisticated use of encryption thus disguising themselves from signature based systems. Anomaly based systems excel at identifying zero-day attacks, previously unknown, or well-disguised attack methods. Because anomaly detection systems do not rely on pre-programmed attack signatures, they are able to identify a much broader range of network threats. However this strength is also the source of the greatest deficiency. Existing anomaly detection systems lack the ability to interpret anomalous events. While anomalies are readily identified, every step in the process of applying anomaly detection to network defense requires extensive expert intervention. This requirement significantly delays the implementation of appropriate response actions thereby reducing overall effectiveness.

This thesis integrates learning techniques with quality of service approaches to move away from existing models that rely almost entirely on operator intervention. We explore automated approaches that are able to quickly, accurately, and dynamically evolve with their environments with little or no expert input. We develop an approach that can automatically identify, assert confidence, correlate and respond to anomalous events. We advance the state of the art of intrusion detection by proposing a fully integrated detection and response framework that is self-adaptive, self-tuning, self-optimizing and automatically responsive. We specifically achieve the following contributions:

1. Develop efficient methods for flow based data collection that enable identification of network wide anomalies in near real time while also identifying individual contributing flows.
2. Automate the gathering and maintenance of base models of network behavior on-line by developing two adaptive anomaly detection mechanisms.
3. Automate the process of correlating anomaly alerts into an evolving threat status.
4. Automate tuning of control parameters eliminating the need for expert knowledge of the operating environment and the underlying detection algorithm.
5. Propose and justify a new model for optimal performance of an anomaly detection system

6. Automate the optimization process on line to achieve the goals established in the proposed performance model.

7. Develop algorithms for determining prediction confidence.

8. Develop novel Quality of Service algorithms providing for delay differentiation with fair bandwidth sharing and buffer management.

1.6 Thesis Roadmap

The remainder of this thesis is organized as follows.

Chapter 2 provides an overview of published literature related to the topics explored in this work. We discuss the state of the art of network intrusion detection and provide overview classification of modern techniques. We discuss recent approaches for flow based data collection as it relates to intrusion detection. We further discuss historical and recently proposed intrusion detection performance models.

In Chapter 3 we provide an overview of the data sets used in this thesis. We briefly describe the KDD dataset, the Trace Data + Simulated (TD-SIM) data, the process we used for developing the TD-SIM data, and the MAWI-Lab labeled backbone trace data set.

In Chapter 4 we discuss our approach for data collection and pre-processing for anomaly detection. We propose a flow based data collection approach based on the Count-Min sketch data structure. We discuss and provide a method for dynamic input normalization.

In Chapter 5 we develop two adaptive network anomaly detection engines based on machine learning techniques. One based on a Growing Hierarchical Self Organizing Map, and another based on One Class Support Vector Machines. We propose a method for aggregated alert correlation and demonstrate its ability to enhance performance of our proposed detectors. We evaluate the general effectiveness of these models and their ability to adapt over time using the data sets described in chapter 3.
In Chapter 6 we propose a self-tuning self-adapting system controller based on reinforcement learning and function approximation techniques. We compare results between manually tuned and auto-tuned approaches. We also propose a framework for defining optimal performance in a network anomaly detection system and evaluate the system controller’s ability to conform to this framework using the proposed detection engines and the identified datasets.

In Chapter 7 we discuss automated response mechanisms. We develop and discuss novel algorithms for performing fair bandwidth sharing and delay differentiation to support automated response.

Chapter 8 concludes the thesis with summarization of the proposed integrated framework and evaluation results, a discussion of identified drawbacks and directions for future work.
Chapter 2.

Related Work

2.1 Classification of Anomaly Detection Systems

The work in [32] proposed a model for intrusion detection based on the assertion that exploitation of a system’s vulnerabilities involves abnormal use of the system. Therefore, intrusions could be detected by identifying abnormal patterns of system usage. Since then, a significant amount of research has been accomplished in the area of anomaly based intrusion detection. A number of surveys have been published that characterize the features of intrusion detection systems. Some surveys have categorized a broad range of intrusion detection approaches while others have focused on anomaly based techniques. The authors of [31] classified systems according to Detection method, behavior on detection, audit source location, usage frequency, and whether systems were knowledge based or behavior based. Building on the idea of knowledge versus behavior based, the work in [3] identified two broad categories of intrusion detection systems: anomaly based and signature based. Anomaly based systems were further divided into Self-learning systems and Programmed Systems. The authors in [151] and [78] provide comprehensive reviews of Machine Learning Techniques and Artificial Intelligence techniques to intrusion detection respectively. In [110] Anomaly based Intrusion Detection classifying models are surveyed based on approach. [29] provides a survey on anomaly detection applied to a variety of disciplines including intrusion detection. Inspired by these surveys, we discuss related work according to the following distinctions: The input data examined for detection, and the method of identifying anomalies. We first characterize anomaly based methods based on the input data. Host based systems monitor data such as system call stacks, registry access and application logs, network based systems typically monitor packet header, packet content, and connection information. We further classify approaches based on the detection method. The surveys identified previously classify approaches in a variety of disciplines including statistical analysis, machine learning, data mining, and association rule discovery. Data mining and machine learning are closely
related disciplines and there is significant overlap in classifying methods in these disciplines. Within the context of network anomaly detection we use three broad classifications: statistical analysis, rule based approaches, and machine learning based approaches. The basic assumption of statistical anomaly detection techniques is that normal data instances occur in high probability regions of a stochastic model, while anomalies occur in the low probability regions of the stochastic model [29]. Rule based approaches are methods that heuristically identify a set of rules that a system can use to identify anomalous network events. Machine learning methods involve approaches that use input data to learn patterns of normal and abnormal traffic and evolve identification and or classification capability of suspicious traffic according to the algorithm used. The work in this thesis is network anomaly detection with machine learning and quality of service mitigation. As such we do not address signature based approaches. In our discussion of related work, we first briefly discuss host based approaches. We then discuss various network based approaches including statistical, rule based, and machine learning methodologies with the most emphasis on machine learning approaches.

2.1.1 Host Based Anomaly Detection

Host based systems are primarily concerned with modeling the behavior of individual users or nodes on the network. Examples of input data include system call stack, computer registry entries and executable file configurations. One of the earliest examples of a host based system was Haystack [139]. Haystack monitored user behavior by extracting event information from user session audit trails. Haystack considered a variety of session features and defined ranges that were considered normal for those features. Ranges were selected for both individual users and user groups. Probability distributions were used to determine if ranges were exceeded by too much or too often and alarms were raised to the System Security Officer for investigation when anomalous behavior was suspected. In [128] the Neural Network Intrusion Detector was proposed that used back propagation neural networks to monitor user behavior by learning what commands they typically use during a day.
Unusual patterns would be flagged as intrusion. These approaches were designed to be utilized off line. On-line systems that attempted to monitor user behavior were proposed in [30, 43]. These approaches developed models of user command history and used these models to predict future user commands. This information was used to determine if issued commands were “predictable” or anomalous. Other host based models monitor program behavior. In [42] the authors proposed a method that analyzed individual programs sequences of system calls and developed profiles of normal behavior. They proposed an analogy between the human immune system and intrusion detection arguing that within programs, the short range ordering of system calls will be consistent provide a definition of “self” for that program. Deviation from this sense of “self” or normal behavior would indicate intrusion. Additional host based systems analyzing
sequences of system calls were proposed that applied the RIPPER algorithm [28, 87] to Unix process traces, [159] modeled call traces using Hidden Markov Models, and [52, 92] applied machine learning techniques. Host based techniques designed to monitor Windows Registry access have also been proposed. [2] used a probabilistic anomaly detector and [58] examined registry access using Support Vector Machines to identify malicious behavior. Other recent approaches utilize an ensemble of techniques to monitor program behavior and identify malicious software or malware [23, 101].

2.1.2 Statistical Approaches to Network Anomaly Detection

In statistical methods for anomaly detection, the system observes the activity of subjects and generates profiles to represent their behavior. Typically, two profiles are maintained for each subject: the current profile and the stored profile. As events are processed, the system updates the current profile and compares it to the stored profile using some stochastic model. The basic assumption of statistical anomaly detection techniques is that Normal data instances occur in high probability regions of a stochastic model, while anomalies occur in the low probability regions of the stochastic model [29].

**Frequency Analysis:** A basic approach to stochastic anomaly detection is the use of frequency histograms. Statistical Packet Anomaly Detection Engine (SPADE) [144] is a statistical anomaly detection system used for automatic detection of stealthy port scans. SPADE uses a simple frequency based approach, to calculate the ‘anomaly score’ of a packet. The fewer times a given packet was seen, the higher was its anomaly score. Once the anomaly score crossed a threshold, the packets were forwarded to a subsystem designed to detect port scans. In [1] the Network Intrusion Detection System (NIDES) included a subsystem that maintained long-term statistical profiles to monitor normal behavior of a computer system [66]. Two statistics are maintained, a Q statistic and an S statistic. Q represents the long term behavior of the monitored value, S is a transformation of Q and represents the variance between short term behavior and long term behavior. Large values for S indicate abnormal behavior. The original approach monitored individual system behavior. In [113] the EMERALD system expanded the capability to include monitoring of the entire enterprise and allowed for detection of Internet worms and DDoS attacks. [130] also extended this subsystem to perform anomaly detection in link-state routing protocols. [76] used the basic histogram approach to perform service specific anomaly detection. The ASCII characters in network payloads are sorted by frequency. Anomaly scores were calculated by monitoring request type, request length, and
payload distribution. The work in [98] also used a simple technique to learn normal ranges of packet header fields. It then compared the observed frequency of header values with the expected probability distribution of header values to identify anomalous behavior. In [168] a Parzen Window probability density frequency classifier was used to not only identify anomalies, but also perform basic classification. [60] propose a dynamic model based on Sketch [34] where parameters are tuned on-line. In [16] a model based on high-order Markov chains is proposed.

**Bayesian Networks:** A Bayesian network is an approach that models statistical dependencies and causal relationships between system variables. The model is typically represented by a directed acyclic graph where each node represents a variable and each link indicates a Bayesian relationship of one variable on another. A naïve Bayes network is a restricted network that assumes independence between nodes. This restriction results in a tree structure. In Bayesian approaches conditional probabilities are used to predict the traffic class of network events. The work in [154] proposed a Naïve Bayes approach that evaluated network traffic bursts for anomalous behavior. This method was capable of detecting distributed attacks where each individual attack session was not anomalous enough to raise alerts on its own. [131] Proposed a general model for network intrusion detection based on Bayesian reasoning. It proposed general methods applicable to many different types of networks arguing that Bayesian methods lead to coherent systems that can handle the complex distributions associated with network traffic. [155] proposed a Bayesian latent class modeling approach that utilized unsupervised learning and did not require labeled training data. [18] used Bayesian Networks to detect anomalous attacks in FTP traffic.

### 2.1.3 Rule Based Approaches to Network Anomaly Detection:

In rule based approaches, an anomaly detector learns rules that describe normal behavior of the system. A network event that is not covered by the defined rules is considered anomalous. The first stage of rule based systems is to apply a rule learning algorithm.

**Rule Induction:** A common approach to induced rule generation is RIPPER [28]. In the RIPPER algorithm, a large rule set is first induced, it is then iteratively revised to increase its accuracy. [86] used RIPPER to characterize network trace data and identify anomalous network connections. [38] used artificially generated anomalous events to better train RIPPER and increases its effectiveness.
**Association rule mining:** The basic elements of association rule learning include a database D of transactions T, where each transaction T in D consists of a set of items in the database. An association rule is a rule in the form of X -> Y, where X subset of T, Y subset of T and X intersection Y = null. The rule X -> Y is said to have confidence c if c% of the transactions containing X also contain Y. The rule X -> is said to have support s if s% of the transactions in D contain X union Y. [88] applied an association rule mining algorithm to finding anomalous network connections. In [5] the ADAM system used a sliding window approach to finding frequent associations in TCP connection data and identify malicious events.

**Fuzzy Rule Generation:** In [62] it was first argued that Fuzzy Logic was particularly suited to network security problems. [12] point out that several parameters that are used in the intrusion detection methodology can be viewed as Fuzzy values. They also asserted that fuzzy approaches are well suited for network anomaly detection because they help smooth the boundary between normal and abnormal conditions. Many approaches have been based on Fuzzy Logic principles. [35] proposed the Fire Intrusion Recognition Engine (FIRE). FIRE mined TCP header information and created an aggregate data key from the mined information. The data keys were then combined with frequency information and sorted into fuzzy sets. Fuzzy rules were then manually generated. The rules were then used to identify anomalous connections. While FIRE was effective against probe events, the manual rule generation process was intensive. [64] proposed a method for automating rule generation for the FIRE system. In [50] An Evolving Fuzzy classifier was proposed that used genetic algorithms to find simple Fuzzy rules to identify abnormal connections. In [24], A Fuzzy Rough classifier was used to identify anomalous connections. The Rough classifier grouped the object space into three regions: lower, boundary, and negative. The Fuzzy classifier applied fuzziness to connections in the boundary region. The test connections were then grouped into five classes including normal and four abnormal cases.

**Genetic Programming:** Genetic algorithms are a computational way of mimicking natural selection and evolution. They are commonly used to find approximate solutions to optimization and search problems. They have also been applied to the intrusion detection problem. [89] used a genetic algorithm to identify a rule set for anomaly detection. This approach used a connection field weighted approach to determine level of suspicion. The weights used for each field are manually tuned by an operator. [112]
also proposed a genetic approach to generate rule sets. In this approach manual intervention for operator input is required during a periodic retraining process.

2.1.4 Network Anomaly Detection with Machine Learning

Machine learning is the ability of computer programs to improve performance on a set of tasks over time. Machine learning techniques are a broad range of identification and classification techniques that evolve detection models based on the input of previous data. Methods in this class include K-NN clustering techniques, Support Vector machines, and Artificial Neural networks.

**K-nearest neighbor:** K-NN is a clustering technique in which new events are classified by majority vote of their K nearest neighbors in the trained vector space. Where K is the number of neighbors that get a vote, and the concept of “nearest” is calculated by methods such as Euclidean distance or Mahalanobis distance. An advantage of the KNN approach is its flexibility. A disadvantage is the computational complexity of calculating the K nearest neighbors of each data point. In [37], the search space was first clustered into subsets to reduce the time required to identify the neighbor set. In [8] Principle Component Analysis was first applied to the input set to reduce the dimensionality of the vector space. An enhancement to the basic KNN approach was proposed in [90] where transduction techniques [44] were applied to calculate a “strangeness” measure. This approach reduced false positives over other K-NN approaches and reduced the need for labeled training datasets.

**Support Vector Machines:** Support vector machines provide a supervised learning method for performing classification of high dimension description vectors [153]. Support vector machines use hyper-planes to divide samples into one of two classes. In the case if anomaly detection they identify normal and abnormal network events. In [37] an unsupervised learning approach to SVM was used that could be trained with unlabeled data. [107] proposed a one-class SVM for intrusion detection that identified outliers among positive examples and treated them as negative examples. [170] proposed an SVM that conducted training on-line to manage concept drift. Rather than periodically training the model off line, a learning algorithm was applied where training data was fed in sequence rather than in batch optimizing the training process.

Support Vector Machines (SVM) have been applied to both signature based intrusion detection [61, 104, 111] and anomaly detection [37, 58, 63, 107]. Anomaly detection using SVM is commonly based on
one-class SVM [133]. In one-class SVM, only positive examples are used for training. The algorithm calculates a decision boundary in feature space that encompasses most of the training examples. Examples not encompassed are outliers and are considered negative examples. A root challenge of anomaly detection is concept drift. As models of normal behavior change, detection systems must be retrained. Traditional SVMs are trained in batch and are not well suited to on-line adaptation. Enhancements to SVM that attempt to overcome this liability typically involve methods based on incremental SVM [126, 147] or on-line SVM [83]. Incremental approaches converge toward optimum by training on small batches of new data combined with the output of previous partial solutions [93, 103]. Approaches using on-line SVM process training examples one at a time rather than in batch and are therefore better suited to streaming data [170]. While these approaches have shown promise in achieving on-line adaptation they still suffer from scalability issues due to the requirement to maintain kernel matrices. The SMO algorithm [115] is a method of solving the SVM without requiring matrix operations. Our approach uses the SMO approach and maintains a dynamic auxiliary training set. The training set is adapted online and the SMO algorithm is used to progress from the existing solution to the new solution based on the updated auxiliary set w/o the need for matrix operations.

Support vector machines do not innately provide confidence intervals or posterior probabilities on classifications predictions. In SVM classification, hyperplane proximity or decision function scaling methods have been proposed for determining individual prediction confidence [114, 127]. Ji et al proposed a method for assigning confidence scores less than 1 to examples most likely to be support vectors in an effort to improve classification accuracy [67, 68]. Each training example is augmented with a confidence score. In [67] A priori example confidence was based on human ability to classify examples. In [68] confidence is automatically assigned by proximity to the decision boundary. The augmented training examples are used to improve classification accuracy.

**Artificial Neural Networks:** An artificial neural network (ANN) is a computational model that is inspired by the structure and functional aspects of biological neural networks. Feed forward Back propagation networks have been used primarily in host based anomaly detection [53, 54]. In [172] several types of neural networks were applied to network anomaly detection and tested individually. The authors experimented with Perceptron, Backpropagation (BP), Perceptron-backpropagation-hybrid, Fuzzy
ARTMAP, and Radial-based Function neural networks. Their approach was particularly effective at detecting UDP flood attacks. An ANN approach that has been commonly used in network anomaly detection is based on the Self-organizing map. The Self organizing map converts non-linear relationships between data points in high-dimensional space into geometrical relationships between points in two dimensional space [75, 120] used SOMs for network anomaly detection. In this approach input vectors were mapped to regions of the map and then categorized both on which region they were assigned, and how well they fit that region. Individual specialist maps were trained to recognize anomalies in specific protocols. [117] proposed ANDSOM as an anomaly detection module for the INBOUNDS intrusion detection system [150]. ANDSOM modeled traffic in six dimensions and was particularly effective against buffer overflow attacks. [71] and [129] proposed using a fixed hierarchy of self organizing maps. In [71] three layers were employed. The first layer was associated basic TCP features. The second layer was used to classify connections clustered by the first layer. The third layer was used to further process connections for those neurons, which win for both attack and normal behaviors in the second layer. [129] proposed a fixed hierarchal model where each layer in the map was trained to cluster different features of a connection. Test connections are only fed to subsequent layers if they are classified by low accuracy clusters in parent maps. [109] proposed using a Growing Hierarchical Self Organizing Map [118] for anomaly detection. In this approach, the map size and dimensionality is not fixed before training. Rather, it is grown to fit the training space based on observed quantization error. Once training is complete, the map dimensions are fixed. Other variations on SOM approaches have been proposed in [69, 156].

2.1.5 Hybrid Techniques

It is becoming more common to combine several approaches into a single model for intrusion detection. The idea behind this approach is to combine the strengths of several different approaches to maximize accuracy and minimize false positives. One simple approach to a hybrid system is a composition where modules are deployed as part of a comprehensive system. Examples of this approach include SPADE [144] which was a plugin for SNORT and [154] that developed plugin for EMERALD [113]. Another approach to hybrid systems involves using one approach to enhance the training and configuration of another. In [50, 135, 137] Genetic algorithm approaches were used to enhance training of other machine learning models. Multiple approaches can also be combined to produce a single prediction output. Cascading systems
involve using the output of one approach as the input to another. In [72] a cascading series of Self Organizing Maps was used. In [33] a Self Organizing map was used as an anomaly detection preprocessor for a decision tree module that raised event alerts. [111] proposed a cascading Decision Tree and Support Vector Machine approach. An integrated hybrid approach directly combines multiple methodologies. In [39] an approach combining Naïve Bayes and Decision Tree methods into one classifier was proposed. [137, 156] Combined Support Vector Machines with Self Organizing Maps.

2.1.6 Dynamic Anomaly Detection

The majority of surveyed approaches are static. That is that they consist of some sort of off line training or modeling phase and then an on-line testing or running phase. During the off-line training phase, a model of normal behavior is learned. During the on-line running phase, incoming events are compared to the learned model and tested for normality. Over time, the normal behavior of the system evolves and the model must be updated. Static systems do this through periodic off-line retraining. There is growing research in performing this re-training incrementally on-line. [17] proposed an adaptive neural network that autonomously learned new attacks on-line based on feedback from protected systems. The operating state of protected systems was fed back into the detection model. When incoming traffic was predicted normal, operating state should remain normal. If the operating state degraded after receiving “normal” traffic, the model could use this feedback to learn new attack types. [121] proposed a model where previously evaluated packets that were identified as normal were periodically used to retrain the detection system on-line thus evolving the model of normal behavior. To prevent poisoning, the feedback input was selected randomly and was additionally sanitized prior to being utilized for training. [119] proposed a method of dynamically updating an intrusion detection system by using “challenges”. A challenge is a previously evaluated normal or attack pattern that is fed back into the system on-line. Based on the systems response to this challenge, the detection module can be evaluated and updated as necessary. [26] proposed a method of updating models of normal behavior by using time-delimited slices of data to represent normal models and re-calibrate training data for a method independent detection system. They argue that specific attacks will be concentrated around certain time periods affecting only a small fraction of the micro models reducing the probability of data poisoning by stealthy attacks. In [60], rather than adapting the base model
of normal behavior, the system dynamically updates the parameters used for detection. They argue that at different times of day and under different traffic conditions, a different set of parameters will provide optimal detection capability.

Recently there are a few studies on tuning specific models on-line [41, 60, 70, 122]. For instance, Himura et al. investigated the effect of on-line parameter tuning for the SKETCH algorithm [60]. A method for automatically learning optimal parameter setting and dynamically adapting the learning period was proposed. It was shown that an inappropriate parameter setting significantly degraded performance. Additionally, when selecting optimal parameter settings the same value cannot be used consistently, because the optimal parameter is not constant during on-line operation. Fontugne et al. described a Hugh transform method for anomaly detection that also automatically adjusted a parameter set in regards to the traffic fluctuations [41]. The value of the time intervals considered was automatically computed based on recent throughput and distribution in traffic feature space. The work in [122] examined the sensitivity of PCA to parameter settings and found that minor changes to the parameter settings increased false positive rates by a factor of three or more. Kanda et al. combined SKETCH with PCA and examined the impact of changing the PCA parameters based on the cumulative proportion for each randomly divided traffic sketch [70]. A method for adaptively tuning PCA parameters that outperformed similar approach using fixed parameters was discussed.

Ciocarlie et al. proposed a self-calibrating model that used unlabeled data divided into micro-models of normal behavior [26]. A voting threshold was used to determine the number of micro-models that must accept a packet before it would be considered normal. The voting threshold was then dynamically updated to improve performance.

The work in [171] presented M-AID as a middle ware to correlate the functioning and tuning of multiple atomic anomaly detection systems. They used a reinforcement learning approach to tune the system of ADS's where each individual ADS was considered part of the whole and adapted tuning based on a global reward signal. Emphasis in [171] is on coordinating the behaviors multiple individual ADS's. System state information was based on ADS action and coarsely divided into two main states Attack and Normal. This approach dynamically tuned systems for better performance in a variety of threat levels.
Operator feedback and interaction has been used to tune detection systems on line in [51, 119, 169]. Rehak et al. proposed a method of dynamically updating an intrusion detection system by using multiple individually tuned agents and a system of “challenges” [119]. A challenge is a previously evaluated normal or attack pattern that is fed back into the system on-line. They are processed and evaluated together with the rest of the traffic then used to update the anomaly detection mechanism. The work in [169] proposed a rule based automatically tuning IDS that took advantage of the analysis of alarms by the system operators and tuned the model on-line. Based on operator feedback, rule weights were adjusted to improve system performance. Gornitz et al. also used operator feedback to automatically tune a SVM based IDS [51]. They proposed a semi-supervised learning approach that used a combination of labeled and unlabeled data for training. The labeled data was generated on-line by identifying predictions that were identified in anomalous feature space, but lie close to the boundary. These events were forwarded to security operators for labeling and then used to re-train the model.

<table>
<thead>
<tr>
<th>Author</th>
<th>Year</th>
<th>Host</th>
<th>Network</th>
<th>Statistical</th>
<th>Rule Based</th>
<th>Machine Learning</th>
<th>Hybrid</th>
<th>Dynamic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fontugne et al 2011</td>
<td>2011</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Farid et al 2010</td>
<td>2010</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Himura et al 2010</td>
<td>2010</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Wang et al 2010</td>
<td>2010</td>
<td>x</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Ciocarlie et al 2009</td>
<td>2009</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Himura et al 2009</td>
<td>2009</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Jiang et al 2009</td>
<td>2009</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Menahem et al 2009</td>
<td>2009</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Rehak et al 2009</td>
<td>2009</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Callegari et al 2008</td>
<td>2008</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Rieck et al 2008</td>
<td>2008</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Cha and Lee 2007</td>
<td>2007</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dewaele et al 2007</td>
<td>2007</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Li et al 2007</td>
<td>2007</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Peddabachigari et al 2007</td>
<td>2007</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Shon and Moon 2007</td>
<td>2007</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Chimphee et al 2006</td>
<td>2006</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Shon et al 2006</td>
<td>2006</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Chen et al 2005</td>
<td>2005</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Depren et al 2005</td>
<td>2005</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Idris and Shanmugam 2005</td>
<td>2005</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sarasamma et al 2005</td>
<td>2005</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Zang and Shen 2005</td>
<td>2005</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bouzida et al 2004</td>
<td>2004</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>


<table>
<thead>
<tr>
<th>Reference</th>
<th>Year</th>
<th>Column 1</th>
<th>Column 2</th>
<th>Column 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fan et al 2004</td>
<td>2004</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Li 2004</td>
<td>2004</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Pillai et al 2004</td>
<td>2004</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Scott 2004</td>
<td>2004</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Apap et al 2002</td>
<td>2003</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Heller et al 2003</td>
<td>2003</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Kayacik et al 2003</td>
<td>2003</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ramadas et al 2003</td>
<td>2003</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Eskel 2002</td>
<td>2002</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gammerman and Vovk 2002</td>
<td>2002</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gomez and Dasgupta 2002</td>
<td>2002</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Kruegel et al 2002</td>
<td>2002</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Liao and Vermuri 2002</td>
<td>2002</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nguyen et al 2002</td>
<td>2002</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Staniford et al 2002</td>
<td>2002</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Yeung and Chow 2002</td>
<td>2002</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Zheng et al 2001</td>
<td>2001</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mahoney et al 2001</td>
<td>2001</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bridges et al 2000</td>
<td>2000</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cannady et al 2000</td>
<td>2000</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dickerson et al 2000</td>
<td>2000</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ghosh et al 2000</td>
<td>2000</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lee and Stolfo 2000</td>
<td>2000</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rhodes et al 2000</td>
<td>2000</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tjaden et al 2000</td>
<td>2000</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Valdes et al 2000</td>
<td>2000</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ghosh et al 1999</td>
<td>1999</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lee et al 1999</td>
<td>1999</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sinclair et al 1999</td>
<td>1999</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Warrender et al 1999</td>
<td>1999</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ghosh et al 1998</td>
<td>1998</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sargor 1998</td>
<td>1998</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lee et al 1997</td>
<td>1997</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Porras and Neumann 1997</td>
<td>1997</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Forrest et al 1996</td>
<td>1996</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Anderson et al 1995</td>
<td>1995</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hosmer 1993</td>
<td>1993</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Debar et al 1992</td>
<td>1992</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Javitz and Valdes 1991</td>
<td>1991</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fox et al 1990</td>
<td>1990</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ryan et al 1998</td>
<td>1998</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Smaha 1988</td>
<td>1988</td>
<td>x</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2-1: Classification of Techniques
2.2 Flow Based Data Collection

Due to increased network speeds and growing use of encrypted protocols, flow based intrusion detection is an active research topic. A variety of flow based detection methods have been recently proposed. In [73] records are augmented with additional data. Traditional flow records are augmented with traffic pattern information and a look-up table is used to identify scanning and flooding attacks. Packet and flow sampling techniques are also common. [10] compared results using flows generated via packet sampling with varying sample intervals. [173] used flow level sampling to identify DoS attacks and packet filtering to ensure no more than one packet per flow was sampled. [45] and [79] perform analysis on aggregated flow data. [46] uses 2d sketches to aggregate flow data and identify scanning and SYN flood attacks. The sketches track, for each time frame and each tuple (dest_IP, dest_port), the difference between the number of SYN packets and the number of SYN/ACKs. If the stored value deviates from the expected one, a DoS SYN Flooding attack is going on. In [79] and [81] traffic is first aggregated into Origin-Destination (OD) flows PCA analysis and eigenflows are then used to identify a variety of anomalies. An eigenflow, derived from a PCA of OD flows, is a timeseries that captures variability in the OD flows. Each OD flow is expressed as a weighted sum of eigenflows. This approach is effective at identifying anomalies in multiple sub-spaces of flow header combinations and uses flow data to identify a variety of anomaly types including denial of service attacks (single-source and distributed), flash crowds, port scanning, downstream traffic engineering, high-rate flows, worm propagation, and network outages. The methods in [70], [91] and [34] perform analysis on network wide data and then extract information related to the flows contributing to the identified anomaly. These approaches are based on random sketches of network traffic. They are similar in that network data is randomly aggregated into sketches and statistical analysis approaches are used to identify anomalies. They differ in their analysis approach. [70] used PCA, [91] uses sub-space clustering, and [34] uses non-gaussian multiresolution statistical detection techniques. If an anomaly is detected the sketches are deconstructed to identify the flows that contributed to the anomaly. Evaluation in [91] is performed off-line, in [70] and [34], evaluation is performed in 15 minute time buckets.

2.3 Intrusion Detection Performance Models
Effective tuning is a two-step process consisting of identifying performance objectives and then selecting values for tuning parameters to meet them. Systems are tuned to achieve optimal operation which is generally considered finding the optimal tradeoff between false alarm rate and the detection rate. This tradeoff is commonly analyzed using receiver operating characteristic (ROC) curves [37, 76, 86]. However, it has been identified that using ROC alone has many limitations [4, 19, 55]. In order to address the limitations of simple ROC analysis, many alternative IDS evaluation methods have been proposed [4, 19, 48, 55, 145]. Axelson pointed out the base rate fallacy, rooted in Bayesian statistics and states that because overall attack rates are low, that when a detection system raises an alarm, the likelihood that there is actually an attack is also low [4]. The work in [4] defined the Bayesian detection rate as the probability of an attack given that an alarm has been raised. This metric requires a-prior knowledge of the expected attack rate before an acceptable ROC operating point can be selected. Stolfo et al. discussed work in fraud detection and expanded that work to intrusion detection [145]. They proposed a cost based model with three types of costs, Damage cost, Challenge cost, Operational cost. Using this method an optimal ROC operating point becomes the point where total cost is minimized. Likewise, Gaffney et al. proposed a decision theory method that extended ROC analysis by using expected cost and attack probability to identify the optimal operating point [48]. Gu et al. argued that cost based methods are confusing and ineffective when the cost factors are not carefully selected [55]. In [55] an objective metric, $C_{ID}$ (Intrusion Detection Capability), motivated by information theory was proposed. Cardenas et al. point out that a drawback of this method is that it is difficult for an operator to tune $C_{ID}$ to local requirements. This is because the notion of reducing the uncertainty of an attack is difficult to quantify in practical values of interest such as false alarms or detection rates [19]. In [19] the Intrusion Detection Operating Curve (IDOC) is proposed. The IDOC combines ROC analysis with the use of isolines to capture the impact of uncertain environmental variables such as likelihood of attack and operational costs. The optimal operation point is then found by identifying the point on the ROC curve that intercepts the optimal isoline of the uncertain metric.

In all of these methods, traditional false positive and false negative analysis is extended by also considering environmentally specific variables such as cost and attack rate. Tuning becomes the process of
predicting the variables for an operating environment and then adjusting system parameters to approach optimal operation. The tuning goal consists of finding a locally defined optimal tradeoff between false negative and false positive rates. The inherent limitation of accomplishing this off line is that even if a reasonable range for the environment variables is known a-priori, their exact values are highly dynamic and even small changes in these values can shift the operating point on the ROC curve to an unacceptable tradeoff.

2.4 Open Issues in Network Anomaly Detection

Despite extensive research and a large solution set in the realm of network anomaly detection, the majority of operational systems in use today are misuse detectors, most commonly signature systems that scan traffic for byte sequences [142]. This is due to a number of open issues that remain in the field of anomaly detection.

2.4.1 False Alarm Rate

Proposed anomaly detection solutions predominantly suffer from high false alarm rates. Many approaches report false positive rates between 10% and 1% [37, 72, 169]. [149] surveyed 276 anomaly detection approaches proposed in the last 8 years and reported that most achieve 98% accuracy with false positive rates at 1%. While this seems promising, [4] points out that in operational scenarios 1% false positive rate may be unacceptably high. When the overall attack rate is very low, even with very high accuracy false positive alerts will outnumber true positive alerts in the alert log.

2.4.2 Relationship between anomaly and intrusion

Network anomaly detection is based on the premise that anomalous behavior is malicious behavior [32]. This core assumption leads to two possible categories of behavior in a system.

1. Not intrusive and not anomalous
2. Intrusive and anomalous.

However, [77] propose that anomalous behavior does not always mean intrusive activity. They argue that there are four possibilities:
1. Intrusive but not anomalous
2. Not intrusive but anomalous
3. Not intrusive and not anomalous
4. Intrusive and anomalous.

The assumption that attacks are anomalous is challenged in [49], [57], and [148]. It is argued that stealthy attacks present as normal traffic despite being malicious. [9], [81], and [97], identify several anomalous circumstances that although unusual, are nevertheless legal operations of the system.

2.4.3 Correlation of attacks
Performing attack correlation is an important step to mitigation and response [108, 152]. Determining the attack correlation in a signature based system is trivial. The attack type is manually assigned during the signature development phase [7, 125]. However, in an anomaly detection system, the detection model is identifying an anomaly, not a clearly defined attack pattern. Frequently there is not enough information to accurately correlate related malicious traffic. Most approaches only report “normal” or “anomalous” events [6, 24, 23]. Few approaches actively study the ability of an anomaly detection system to accurately classify activity [7, 124]

2.4.4 Tuning and optimization
Network operations are constantly evolving. This constant change is directly related to the core effectiveness of an anomaly detection system. As normal network usage evolves, the model of “normality” maintained by the detection system must be updated. While there is some recent research in on-line adaptation of anomaly based systems [26, 119, 121] effective schemes remains an open research issue. Further, there are very few examples of dynamic tuning and optimization of anomaly based systems [60].

2.4.5 Evaluation of approaches
The KDD and DARPA datasets have been criticized [13, 14, 96, 99]. However in [149] of the 276 approaches surveyed, 70% were evaluated using publically available datasets with the majority of these using either the KDD data set or DARPA set. The other 30% of the studies utilized datasets that they
generated themselves with either simulated or recorded trace data. [123] argue for the need for carefully constructed evaluation datasets consisting of a combination of simulated and true trace data.

2.5 Attack mitigation through quality of service allocation.

Many systems have been proposed that attempts to prevent or mitigate cyber attacks by applying quality of service principles. Techniques involving applying varying delay and filtering mechanisms to both packet and connection level communications have recently been studied. [95] proposed a generic system for aggregate based congestion control. During periods of sustained congestion such as those caused by DoS attacks, the model identified congestion signatures using dropped packet histories. It used a destination based detection method in which high bandwidth destinations are selected, then suspected aggregate source streams are filtered according to the detected signature. The model also proposed “pushback” where downstream routers requested upstream routers implement QoS mechanisms against individual streams in the congestion cluster. While this approach is effective it is inherently unfair to legitimate traffic destined for congested areas. [47] Proposed a QoS mitigation strategy that combined rate based regulation strategies with a resource windowing strategy that controlled resource consumption of aggregated flows. [167] Proposed a method of installing rate throttles in upstream routers. Congested servers could then request incoming traffic be limited according to a min-max fair feedback control algorithm. [164] proposed a system for controlling the spread of viruses in enterprise networks by limiting connection rates to new hosts at the network layer. Suspected malicious connections were delayed rather than dropped gaining the network operator time to perform more serious mitigation strategies while limiting collateral damage to legitimate traffic. [134] and [143] proposed rate limiting connection requests from suspected infected clients or subnets. [22] proposed a method of restricting work propagation by dynamically dropping connection requests from sources based on their connection failure rate. As the connection failure rate of a source increased beyond a threshold, random connections from that source were dropped in order to keep the failure rate below a predefined limit.
Chapter 3.

Data Sets for System Evaluation

3.1 Introduction

In this chapter we discuss the datasets we use for discussion and evaluation throughout the remainder of this thesis. We use three data sets in this work: The KDD99 dataset, the TD-SIM dataset, and the MAWI Lab dataset. The first is the KDD99 dataset. Until recently, this dataset was considered the gold-standard for evaluating new approaches to intrusion detection. However, the dataset is now 14 years old and has not been updated. Additionally, there are a number of criticisms that have recently been identified that we will discuss in the following sections. We still use the KDD set as a point of reference to compare our ideas to existing approaches. To avoid drawing research results and conclusions solely based on experiments with the KDD dataset, we use two additional data sets to validate our conclusions. First we have built a dataset combining trace data and simulated data (TD-SIM), which consists of a mixture of live trace data from the Lawrence Berkeley National Laboratory Enterprise network and simulated network traffic.

The TD-SIM data set is a combination of live Trace Data taken from the LBNL enterprise trace project, and Simulated network events. This data set enables us to evaluate our methods on a variety of anomalies on traffic that resembles real world traffic. However while the TD-SIM dataset models a true enterprise, the base traces are taken at random intervals and are not completely adequate for evaluating long term behavior. Additionally, the process of introducing simulated traffic in itself can introduce anomalies. For this reason, we also use one additional dataset, the MAWI Lab trace files.

Second we use labeled live traces from a traffic data repository maintained by the MAWI Working Group of the WIDE Project. The traces are examined and labeled by the MAWILab project to identify normal and anomalous traffic. MAWI Lab set is a labeled trace set taken from an internet backbone link. The dataset is 100% live trace data. In the remainder of this chapter, we discuss and describe the KDD99 data set and the criticisms of its use. We also describe the process we used to create the TD-SIMM dataset. Finally, we discuss and describe the MAWI Lab trace files.
3.2 Knowledge Discovery and Data Mining 1999 Dataset

The 1998 DARPA Intrusion Detection Evaluation Program was prepared and managed by MIT Lincoln Labs. The objective was to survey and evaluate research in intrusion detection. A wide variety of intrusions simulated in a military network environment, was provided. The KDD99 intrusion detection contest used a version of this dataset. This data set is commonly known as the KDD99 set.

To create the set, Lincoln Labs set up an environment to acquire raw TCP dump data for a local-area network (LAN) simulating a typical U.S. Air Force LAN. The network was repeatedly attacked in a simulated environment.

The raw traces contained four gigabytes of compressed binary TCP dump data from seven weeks of network traffic. This was processed into just over 4 million connection records. In addition to TCP trace statistics, the connection records are augmented with data from several other system logs.

Each connection is labeled as either normal, or as an attack, with exactly one specific attack type. Each connection record consists of about 100 bytes. Attacks in the KDD set fall into four main categories:

- **DOS**: denial-of-service, e.g. syn flood;
- **R2L**: unauthorized access from a remote machine, e.g. guessing password;
- **U2R**: unauthorized access to local superuser (root) privileges, e.g., various “buffer overflow” attacks;
- **probing**: surveillance and other probing, e.g., port scanning.

The datasets contain a total of 24 training attack types, with an additional 14 types in the test data only.

Until recently, the KDD set has been considered the gold standard for evaluating intrusion detection systems. In [149] of the 276 approaches surveyed, 70% were evaluated using publically available datasets with the majority of these using either the KDD data set or DARPA set. However, the KDD set is now 14 years old and has not been updated to reflect current threats. Additionally, there have been several criticisms of this set that have been made over the years [13, 14, 96, 99]. Specifically, no validation has been done to ensure the KDD set is in fact similar to live network traffic. Additionally, the distribution of attacks in the
KDD set is said to be unrealistic. Finally, it has been noted that due to the way the traffic in the KDD set was generated, some of the differences between attack traffic and normal traffic is statistically obvious.

Despite numerous criticisms, researchers today continue to use this dataset [15, 138]. Therefore we use this data set in this thesis to compare to existing works and to use as a baseline performance measurement for our work. However, we also use additional sets to perform more thorough evaluation.

3.3 TD-SIM: an Integrated Dataset of Live Traces and Simulated Data

There have been many critiques or arguments made against the use of the KDD dataset [13, 14, 96, 99]. One major issue is that no validation has been done to ensure the KDD dataset is in fact similar to real network traffic [13, 14]. Additionally, the distribution of attacks in the KDD dataset is said to be unrealistic. Due to the way the traffic in the dataset was generated, some of the differences between attack traffic and normal traffic are statistically obvious.

To avoid drawing research results and conclusions solely based on experiments with the KDD dataset, we have built a dataset combining trace data and simulated data, which consists of a mixture of live trace data from the Lawrence Berkeley National Laboratory Enterprise network and simulated network traffic.

TD-SIM was constructed in three steps. We first addressed the issue of similarity between the dataset and live data by using live trace data as the foundation. We started with a publicly available trace dataset and performed preprocessing to correct header/payload discrepancies created during packet anonymization. We then processed the dataset through a commercial intrusion detection system to categorize attack vs. normal traffic. Finally, we inserted simulated traffic to ensure adequate coverage of a variety of network attacks.

The packet trace dataset we used as a base for TD-SIM is from the Lawrence Berkeley National Laboratory (LBNL)\(^1\). The data was collected from Oct 2004 through Jan 2005. The database includes 11GB of packet header traces in 131 files. Each day trace covers a range of ten minutes to one hour. In this dataset, the payload field is deleted and the IP address field is anonymized. Because the payload field was deleted, calculations in the packet headers were not consistent with the actual packet sizes. We used the

commercial tool WireShark\(^2\) to pad the payload fields to match packet header calculations. We chose this method over recalculation in order to preserve the original packet size and data transfer information. As the baseline for identifying malicious activities, we processed the trace dataset through a commercial intrusion detection system (IDS). We selected Snort [125] IDS because it is publicly available and widely used.

In order to ensure coverage of a variety of attack types, we inserted both simulated attacks and isolated live attacks into TD-SIM. Simulated attacks were generated on a test bed network utilizing the MetaSploit [102] and NMAP [106] toolkits. Isolated attacks were downloaded from the OpenPacket and EvilFingers PCAP repositories. We then randomly salted the LBNL pcap files with simulated/isolated pcap files. We call the modified LBNL traces “salted files” and the simulated/isolated files used to salt them the “salting files”. Because the live traces and the simulated/isolated traces were created in different environments, some anomalies could be created inherently by this process. For example, the address space, and timing of the salting files was obviously different than the live data. We attempted to compensate for this during the salting process.

We first made a list of all the endpoints generating traffic in the LBNL trace files. We made a list of unique records capturing address, port, and protocol information for every endpoint. We then modified the simulated/isolated files re-writing the packet header information using identical address space and timing as the original traces. We did this by substituting addresses in the salting files with addresses from the LBNL traces and adjusting the trace times to include similar timeframes as the LBNL traces. We maintained the integrity of all conversations by ensuring a one for one address substitution. Port and protocol information was not adjusted. In order to ensure that our salting process not blatantly use the address space in an anomalous way (i.e. performing and succeeding in an HTTP attack against a target that does not host HTTP), we only substituted addresses in the salted files with addresses from the LBNL traces that had utilized similar ports and protocols within the salting timeframe. That is, if the conversation in the salting file contained packets destined for port 53 utilizing UDP, only addresses in the LBNL traces that had received live UDP packets on port 53 were considered as substitution addresses. Once the salting files were modified, they were merged into the LBNL trace files using the `Mergecap` tool set. The pcap files

were then processed by the flow generator as discussed in section 1.2.3.4. Table 3-1 lists the attack types that were salted into the TD-SIM data set.

<table>
<thead>
<tr>
<th>Attack Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>UDP Scan</td>
</tr>
<tr>
<td>IP Fragment Attack</td>
</tr>
<tr>
<td>JRE Remote Denial of Service</td>
</tr>
<tr>
<td>Active X buffer overflow</td>
</tr>
<tr>
<td>LiveUpdate remote code execution</td>
</tr>
<tr>
<td>Black Ice Software arbitrary file download</td>
</tr>
<tr>
<td>Black Ice remote buffer overflow</td>
</tr>
<tr>
<td>Active X arbitrary file creation/execution</td>
</tr>
<tr>
<td>FTP Arbitrary data write</td>
</tr>
<tr>
<td>IMAP File execution</td>
</tr>
<tr>
<td>Internet Explorer Remote Denial of Service</td>
</tr>
<tr>
<td>D-Link MPEG 4 remote buffer overflow</td>
</tr>
<tr>
<td>Microsoft XML cross domain scripting exploit</td>
</tr>
<tr>
<td>Microsoft Internet Explorer cross zone scripting</td>
</tr>
<tr>
<td>Adobe Flash remote code execution</td>
</tr>
<tr>
<td>SMB Man in the Middle Attack</td>
</tr>
<tr>
<td>TCP-Scan</td>
</tr>
<tr>
<td>Telnet Denial of Service</td>
</tr>
<tr>
<td>ICMP Echo fingerprinting</td>
</tr>
<tr>
<td>Zeus bot-net infection</td>
</tr>
</tbody>
</table>

Table 3-1: Attack Traffic Salted into TD-SIM Dataset

### 3.4 MAWI Lab

While the TD-SIM Dataset more closely resembles live network activity than the KDD data set, it is still partially simulated and therefore not 100% realistic. Additionally, the traffic collected only comprises a total of 100 hours. The collection times are random and intermittent. This makes it difficult to test a detection systems ability to adapt over longer periods of time. The third data set we use helps us examine our model in the context of both short and long term adaptation. We perform experiments on data from MAWI (Measurement and Analysis on the WIDE Internet) [25]. This archive contains daily traces captured from a trans-Pacific link between Japan and the United States. The data is publically available; packet payloads are omitted and IP addresses are anonymized. MAWI data set consists of traces from 2001
through 2013 with traces collected at the same times and durations every day. Each trace consists of 15 minutes of traffic from 1400 - 1415 GMT. The core MAWI data set is unlabeled data.

In Fontugne et al provide a complimentary set of xml files that identify anomalies in the MAWI data from 2001 through 2010 [40]. They use four different and diverse anomaly detectors combining the results to provide a labeled dataset called MAWILab. The used anomaly detection algorithms are: a PCA based detector that uses random projection techniques [70], a detector relying on sketching and multi-resolution gamma modeling [34], a Hough Transform-based detector [41] and a detector applying the Kullback-Leibler divergence to several histograms monitoring distinct features [10]. The results from these detectors are heuristically combined to provide a “ground-truth” labeled set consisting entirely of live trace data. The output of their method is composed of four levels: anomalous, suspicious, notice and benign. Anomalous is traffic that is positively identified as abnormal. Traffic labeled as suspicious is suspected to be abnormal and should be identified by a reasonable anomaly detection scheme. Traffic labeled notice was identified by at least one detector but is not considered abnormal, and benign is traffic that was not identified by any of the detectors. Anomalies in the MAWILab labeled files are categorized first by specific attack type when specific signatures are identified (i.e. SYN Flood attack, Ping Flood, RST Scan, etc…). They are also categorized by port/protocol for anomalies found on common protocols such as HTTP, FTP, Telnet, etc… Finally, the anomalies are considered unknown if they are clearly anomalous but do not match common attack profiles or common port/protocol pairs. Table 3-2 lists the anomaly types identified the MAWI Dataset.

<table>
<thead>
<tr>
<th>Scan FTP</th>
<th>Unknown</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scan SSH</td>
<td>Sassr</td>
</tr>
<tr>
<td>Scan HTTP</td>
<td>NetBios</td>
</tr>
<tr>
<td>Scan HTTPS</td>
<td>RPC</td>
</tr>
<tr>
<td>FTP Attack</td>
<td>SMB</td>
</tr>
<tr>
<td>SSH Attack</td>
<td>SYN Scan</td>
</tr>
<tr>
<td>HTTP Attack</td>
<td>RST Scan</td>
</tr>
<tr>
<td>HTTPS Attack</td>
<td>FIN Scan</td>
</tr>
<tr>
<td>Other</td>
<td>Ping Flood</td>
</tr>
</tbody>
</table>

Table 3-2: Anomaly types identified the MAWI Dataset

In our experiments we consider Anomalous/Suspicious traffic malicious and all other traffic benign. The MAWI-Lab files include two xml files detailing the resulting label. One file identifies anomalies; the other
identifies “noticed” streams. Each file identifies a slice of traffic and the detectors that alerted on the flow in question. The slice of traffic is identified by two temporal fields \{start\_time, stop\_time\} and five traffic fields \{source\_address, source\_port, destination\_address, destination\_port, protocol\}. One or more of the traffic fields may be considered a wildcard. (That is all values match the field). For example, a slice that is identified by the traffic fields \{131.7.135.10, *, *, *, HTTP\} will match all traffic from source address 131.7.135.10, from all source ports to all destination addresses, all destination ports using HTTP protocol.
Chapter 4.

Network Data Collection and Pre-processing

4.1 Introduction

Nearly all network intrusion detection systems require some form of pre-processing data. Unless the system is going to perform its prediction by inspecting each packet in its entirety, there is some form of pre-processing or data presentation that must be performed. In this chapter we present our ideas for data representation. We first propose an approach for flow based data collection that uses statistical counting methods to augment traditional flow data with additional information that can be used to infer network wide status relating to the flow under consideration. This approach is extremely efficient, and enhances single flow anomaly detection. However it also allows network wide anomaly detection while also tracking contributing flows in near real time. We then discuss a process for dynamically normalizing the generated data records on line. By constantly updating data scale interpretation on line, detection engines are better able to detect anomalies based on recent models of behavior without the need for consistent off-line training.

4.2 Feature Selection and Flow Record Generation

In an attempt to find known attacks or unusual behavior, modern intrusion detection systems traditionally inspect the contents (payload) of every packet [11, 125]. The problem of packet inspection, however, is that it is hard, or even impossible, to perform it at the speed of multiple Gigabits per second (Gbps) [46, 82]. Additionally, Opaque traffic that is compressed or encrypted may make packed content inspection impossible or at the very least, increase CPU requirements by several orders of magnitude [20]. It has been shown that, opaque traffic can account for up to 89% of TCP packets, and 86% of traffic bytes [163]. It is therefore important to investigate alternatives to packet inspection. One option that currently attracts the attention of researchers and operators is flow-based intrusion detection.
The IP Flow Information Export (IPFIX) working group within IETF defines a flow as “a set of IP packets passing an observation point in the network during a certain time interval. All packets belonging to a particular flow have a set of common properties.” The common properties are typically source and destination addresses, source and destination port numbers and IP protocol: \((ip\_src, ip\_dst, src\_port, dest\_port, proto)\).

There is a difference between a network flow and a network connection such as in the context of TCP. A flow can exist in a situation where there is no connection. An example of a connectionless flow is a UDP flow where a set of packets has been sent from a certain source address/port to a certain destination address/port. These packets will be grouped into a single flow despite the fact that no connection exists. A flow can also consist of multiple connections. If several short connections occur within the flow collection timeframe with identical properties, it is possible for them to be aggregated into a single flow. Additionally, a flow does not have minimum size restrictions: each communication between source and destination hosts will generate a flow, even if only single packet has been exchanged.

Creating flow data is a three step process: monitoring, exporting, collecting. These tasks are performed by three components: flow monitor, flow exporter and flow collector. The flow monitor or observation point is responsible for the metering process. The flow monitor extracts the packet header information from each packet seen on the monitored interface and updates the corresponding flow record in the flow cache. If no matching flow record is found, a new record is created. The flow-exporter monitors the flow-cache for expired records and forwards them to the flow collector. In case of Cisco NetFlow [27] and similarly in IPFIX [116], a flow is considered expired when:

- The flow exceeds inactive timeout. (The flow is idle for more than an established threshold). The default value for the inactive timeout for Cisco Netflow is 15 seconds.

- The flow exceeds its active timeout. (The flow has been active for more than an established threshold). When this happens, its corresponding flow record is exported to the collector and, if necessary, a new flow record is created for that flow. For Cisco Netflow, the default active timeout is 30 minutes. However, shorter timeouts are commonly used [141].
• The flow-cache memory gets full. In this case, certain flow records are marked as expired and exported to the collector. Least Recently Used (LRU) algorithms may be used to free the flow-cache memory, as well as heuristic algorithms [141].

The flow collector retrieves the flow records created by the flow exporter and stores them for further monitoring or analysis. As an example of information available in a flow record, a Cisco NetFlow version 5 flow record contains the following information:

NetFlow version 5 contains the following:

- Input interface index used by SNMP (ifIndex in IF-MIB).
- Output interface index or zero if the packet is dropped.
- Timestamps for the flow start and finish time, in milliseconds since the last boot.
- Number of bytes and packets observed in the flow.
- Layer 3 headers:
  - Source & destination IP addresses
  - Source and destination port numbers for TCP, UDP, SCTP
  - ICMP Type and Code.
  - IP protocol
  - Type of Service (ToS) value
- For TCP flows, the union of all TCP flags observed over the life of the flow.
- Layer 3 Routing information:
  - IP address of the immediate next-hop (not the BGP nexthop) along the route to the destination
  - Source & destination IP masks (prefix lengths in the CIDR notation)
- For ICMP flows, the Source Port is zero, and the Destination Port number field codes ICMP message Type and Code (port = ICMP-Type * 256 + ICMP-Code).

Flow-collection is a data reduction technique. As such, there are challenges associated with using this method for intrusion detection. Packet content information is lost and there is little or no network wide traffic state information. An individual, un-augmented, flow record does not contain enough information to detect a variety of attacks. Some single-source/single-destination attacks may be identifiable in a single flow record (i.e. Smurf, Land, Ping-Pong). However, in order to detect a wide range of or more sophisticated attack types (i.e. distributed DoS, worm propagation), multiple flow records or additional network traffic information is required. Advanced flow based techniques must overcome challenges of prediction scope, prediction timeliness, and resource/complexity scalability.
4.2.1 Analyzing Flow Record Data for Anomaly Detection:

Analyzing flow data for anomaly detection typically involves monitoring specific time intervals and seeking anomalies in flow record data field volume and flow record data field distributions over those time intervals. Several types of network attacks/anomalies can be detected via this method. Table 4-1: Anomalies Visible by Change in Distribution lists several examples of identifiable attacks based on distributions. Table 4-2 lists several examples of identifiable attacks based on volume metrics. [73, 81].

<table>
<thead>
<tr>
<th>Anomaly Category</th>
<th>Description</th>
<th>Affected Distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>DoS / DDoS</td>
<td>Denial of Service</td>
<td>Volume to Destination from unusual distribution of Source</td>
</tr>
<tr>
<td>Flash Crowd</td>
<td>Unusual Burst of traffic to Destination</td>
<td>Source Address, Destination Address, Destination Port</td>
</tr>
<tr>
<td>Port Scan</td>
<td>Probes to many destination ports on a small set of destination addresses</td>
<td>Destination Address, Destination Port</td>
</tr>
<tr>
<td>Network Scan</td>
<td>Probes to many destination addresses on a small set of destination ports</td>
<td>Destination Address, Destination Port</td>
</tr>
<tr>
<td>Outages</td>
<td>Traffic shifts due to equipment failures or maintenance</td>
<td>Source Address, Destination Address</td>
</tr>
<tr>
<td>Worms</td>
<td>Scanning by worms for vulnerable hosts</td>
<td>Source Address, Destination Address, Source Port, Destination Port</td>
</tr>
</tbody>
</table>

Table 4-1: Anomalies Visible by Change in Distribution

<table>
<thead>
<tr>
<th>Anomaly Category</th>
<th>Description</th>
<th>Affected Volume Metric</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alpha Flows</td>
<td>Unusually Large point to point flow</td>
<td>Low flow count, high packet or byte count</td>
</tr>
<tr>
<td>Port Scan</td>
<td>Probes to many destination ports on a small set of destination addresses</td>
<td>High flow count, low packet count, High port volume, low destination volume</td>
</tr>
<tr>
<td>Network Scan</td>
<td>Probes to many destination addresses on a small set of destination ports</td>
<td>High flow count, low packet count, High destination volume low port volume</td>
</tr>
<tr>
<td>TCP-SYN Flood</td>
<td>High volume SYN packets w/o SYN-ACK packets</td>
<td>High flow count, low packet count</td>
</tr>
<tr>
<td>Smurf</td>
<td>High volume of ICMP packets from spoofed victim to ICMP Broadcast</td>
<td>High flow count, high source volume, single protocol</td>
</tr>
<tr>
<td>Fraggle</td>
<td>High volume of UDP echo packets from spoofed victim to UDP Broadcast</td>
<td>High flow count, high source volume, single protocol</td>
</tr>
</tbody>
</table>

Table 4-2: Anomalies visible by change in volume metrics

Approaches to flow based anomaly detection are varied. Methods using augmented flow records [73], aggregated records [79, 80, 81], flow-sampling [173], and random network wide traffic data [34, 70] have
been proposed. However, these approaches suffer from deficiencies. Kim et al use flow records that have been augmented with additional network traffic data indicating high or low values for traffic statistics such as bandwidth usage, packets/flow, and flow count. While this approach is not computationally complex and is able to make predictions on-line, it is a signature based approach and is only able to identify scanning or flooding type attacks. Lakhina et al use aggregated flow data and more advanced statistical analysis techniques enabling them to perform detection in near real time on a wide variety of attack types. However, their approach maintains an analysis matrix for each OD combination. In their experiments, their data was limited to < 500 OD combinations. In network backbone traffic, OD combinations will grow too large to scale well. [173] propose methods that are suitable for backbone links. However, their approach is a targeted detection method toward high fan-in/fan-out patterns. While this is well suited for DDoS detection, it is not versatile enough to detect a variety of anomalies. [34] and [70] have proposed approaches that scales well to high-speed networks and are able to identify a variety of anomalies. However, their experiments were forensic in nature performed in batch on trace data.

4.2.2 Augmented flow records

Our goal is to provide detectors with network wide information collected at the same time that flow records are generated in order to minimize detection delay. We also wish to provide detectors with a combination of both volume and distribution information so that detectors can be generalized rather than specialized for specific attack signatures. Finally, the collection method must be computationally efficient to support on-line/near real time detection and must be saleable to support high speed/backbone links. We propose a flow augmentation method based on the count-min sketch that meets these requirements.

Several anomalies are visible by monitoring volume and distribution metrics. We propose a method of augmenting basic flow records with an array of additional information that directly provides volume metrics, but also allows detectors to infer information about distribution changes in specific fields.
Each flow is defined by the 5-tuple \{src_addr, src_port, dst_addr, dst_port, protocol\}. We call individual elements members of this 5-tuple key elements, and we call combinations of key elements consisting of subsets of all five elements flow-keys. There are 31 possible key element sub sets or flow-keys. (i.e. \{src_addr\}, \{src_addr, dst_addr\}, \{src_port, dst_port, protocol\}, etc...) We select all flow keys that include at least src_addr or dst_addr and augment flow records with network wide volume information matching these flow keys. Table 4-3 identifies the flow keys used for augmenting flow records.

When a flow record is generated, we augment that record with flow, packet, and byte counts from the previous \(w\) seconds where \(U\) is a configurable parameter. As an example, if \(U = 30\) and key = 1, the flow record is augmented with an estimate of the total number of flows, packets, and bytes processed over the previous 30 seconds that have the same src_addr as the generated flow. If key = 5, the flow record is augmented with an estimate of the total number of flows, packets, and bytes processed over the previous 30 seconds that have the same dst_addr and src_port as the generated flow. For each flow record we use all 23 keys. It is obvious that volume information is directly available to detectors using this method. However, this approach also allows detectors to infer some limited general connection distribution information as well.

<table>
<thead>
<tr>
<th>Key#</th>
<th>Key Elements included in key</th>
</tr>
</thead>
<tbody>
<tr>
<td>src_addr</td>
<td>dst_addr</td>
</tr>
<tr>
<td>1</td>
<td>X</td>
</tr>
<tr>
<td>2</td>
<td>X</td>
</tr>
<tr>
<td>3</td>
<td>X</td>
</tr>
<tr>
<td>4</td>
<td>X</td>
</tr>
<tr>
<td>5</td>
<td>X</td>
</tr>
<tr>
<td>6</td>
<td>X</td>
</tr>
<tr>
<td>7</td>
<td>X</td>
</tr>
<tr>
<td>8</td>
<td>X</td>
</tr>
<tr>
<td>9</td>
<td>X</td>
</tr>
<tr>
<td>10</td>
<td>X</td>
</tr>
<tr>
<td>11</td>
<td>X</td>
</tr>
<tr>
<td>12</td>
<td>X</td>
</tr>
<tr>
<td>13</td>
<td>X</td>
</tr>
<tr>
<td>14</td>
<td>X</td>
</tr>
<tr>
<td>15</td>
<td>X</td>
</tr>
<tr>
<td>16</td>
<td>X</td>
</tr>
<tr>
<td>17</td>
<td>X</td>
</tr>
<tr>
<td>18</td>
<td>X</td>
</tr>
<tr>
<td>19</td>
<td>X</td>
</tr>
<tr>
<td>20</td>
<td>X</td>
</tr>
<tr>
<td>21</td>
<td>X</td>
</tr>
<tr>
<td>22</td>
<td>X</td>
</tr>
<tr>
<td>23</td>
<td>X</td>
</tr>
</tbody>
</table>
Recall that there is a difference between a flow and a connection. If several connections involving the same key elements occur within the active timeout of a flow, they will be aggregated into a single flow record. Several connections from a single source to several destinations on similar ports occurring within the active timeout window will generate distinct flow records. Including key 1, a flow record will include the total number of flows previously processed from the same source to all destinations, all ports, all protocols. By including key 2, it will include the total number of flows from the same source to the same destination on all ports, all protocols, and so on. Combining these keys can allow the detectors to infer some connections distribution information.

Consider Error! Reference source not found.. In this figure the source host generates 100 connections to each of the three destinations in the active timeout window using similar ports and protocols. (i.e. short web requests, database queries, etc…) The destination shaded blue is the destination used generating a flow record. In this case, key 3 will reflect 1 flow from this source and key 22 will reflect 3 flows from this source because all connections to each destination are aggregated to a single flow record due to similar ports/protocols. The distribution of the 300 connections is very tight.
Now consider Figure 4-2 where the same 300 connections are distributed to 15 different destinations but still use similar ports/protocols (or 150 or 1500 destinations as in a network scan).

In this case, key 3 will still reflect 1 as the number of flows. However, key 22 will reflect 15 flows indicating a more expended connection destination distribution. Likewise key 1 will reflect 15 because the connections to each destination are not distributed across ports or protocols.

Now, consider Figure 4-3 where again, a single source connects to three destinations 100 times each in the active timeout window. Only this time each connection is to a distinct port (i.e. port scan).

In this case, key 3 will reflect 100 as the connections are distributed across ports thereby generating distinct flows. Key 22 will indicate 3 as the number of flows distributed by destination is low. However, key 1 will indicate a high number as the connections are distributed across ports and/or protocols. By combining all 23 keys, a limited estimation of the various levels of distributions of network activity can be achieved.

In order to accomplish this data collection in an efficient manner, we use an array of count-min sketch structures modified to manage streaming data.
4.2.3 Statistical Counting for Flow Augmentation

Calculation of flow-level statistics, such as, sizes and identities of large flows, per-flow traffic, and flow size distribution are essential for network management and security. Measuring such information on high-speed links is challenging since the standard method of maintaining precise per-flow state (e.g., using a hash table) for tracking various flow statistics is prohibitively expensive [173]. The amount of memory used by the data structure can become very large over time as more items are added. Because of its size is large, accessing such a data structure can be quite slow, since it resides in slow memory or in virtual memory. Further, since its size grows over time, periodically the data structure has to be resized, which makes it unsuitable for real-time processing in high speed networks.

We overcome these obstacles by using count-min sketches to estimate flow level traffic statistics [29]. The count-min data structure consists of a fixed \( w \times d \) array of counters. Each row of counters is associated with a different member of a universal family of hash functions. The hash function maps items uniformly on to the range \( \{1, 2, \ldots, w\} \). When a new key \( k \) is added, the hash function for each row is applied and the value in the corresponding entry in that row is incremented. When a count estimate is required for key \( k \), the hash function for each row is applied to look up the corresponding counters for each row. The minimum value out of all the counters is used as the estimate.
Count min data structures are designed to manage counts on a finite data streams. Because they are based on hash tables, over time, they will become saturated. Additionally, they to not maintain temporal state information making them not suited for infinite streaming data. We overcome this limitation by using a small array of structures and a sliding window approach to ensure suitability for our application as seen in Figure 4-4.

![Figure 4-4: Count Min Sketch Schematic](image)

For each window w of U seconds, we divide the window into 10 intervals and use a 2 dimensional i x j array of count min structures. Where i = 1 and represents one U/10 second sub-window and j = 23, one for each flow key estimation. We use a rotating index k indicating the active column to apply updates to. Every U/10 seconds, k is incremented by 1, the new column is reset and becomes the active column. When updates are applied, all 23 structures in the active column are updated, one for each flow key. When an estimate is requested, the values from all 11 columns are aggregated into the estimation. With this arrangement, the granularity of the sliding window is fairly coarse. We compensate for this by discounting the contribution of the oldest column by a ratio equal to the remaining number of seconds in the current sub-window. For example, if each sub-window is 10 seconds long, and the current sub-window has been active for 2 seconds, then the oldest window is discounted 20%. Thus with a total duration request of 100
seconds, the estimate will consist of an aggregation of nine 10 second complete sub-windows, the first 2 seconds of the current sub-window, and 80% of the oldest sub-window.

4.2.4 Flow Creation Timeout values:

There are three temporal parameters to set, the inactive flow timeout, the active flow timeout, and the window size for the count-min array. 15 Seconds is the default inactive timeout in Cisco Netflow and IPFIX and it is the value we use in all our data collection. Default active timeout is 30 minutes, but smaller times are common (cite survey paper).

![Figure 4-5: Cumulative Percentage of Flow Duration](image)

Figure 4-5 details the cumulative flow duration percentages for both the TD-SIM and MAWI datasets. In both datasets, more than 99% of all flows are less than 60 seconds in duration. For the remainder of this thesis we use 60 seconds as the active timeout during flow creation.

In the foundational work behind the KDD dataset, Lee & Stolfo performed anomaly detection using some “time based” features similar to our augmented features including connection counts between src_addr.dst_addr, src_addr.src_port, etc… They demonstrated that detection accuracy could be increased by using a history of as little as 5 seconds. In their work accuracy vs. false positive stabilized at 90 seconds. In section 4.4.2 we empirically test the effect of flow augmentation history duration on accuracy, precision and recall using two detection engines and both the TD-SIM and MAWI datasets.
4.3 Input Normalization and the Effect of Scale

Once the flow records are generated, the data is normalized. We propose an adaptive input normalization approach that will bring attention to the true difference between individual input vectors. In many approaches, data is normalized to account for differences in scale between two different data points. As an example consider that the two data points are height and weight, height being 5~6 feet and weight being 100~200 pounds. Weight would always take precedence over height if the data is not normalized. To capture relative distance between two different values of the same feature, we adjust the input pattern so that all values are in the range of 0 to 1 using simple linear scaling for the normalization. We consider the “distance” to be how different these two values are from one another. The smaller the distance, the more similar the values are considered. The scale used to normalize input data has an effect on relative distance.

In a simple example, Table 4-4, the reference range used to normalize input values varies from 1~16 to 1~256. For each reference range, two values 8 and 14 are normalized based on the range and their normalized distance is calculated. When the reference range is 1~16, two values are considered very different from each other as the normalized distance is 0.4. When the reference range is 1~256, they are considered very similar as the normalized distance is 0.023.

<table>
<thead>
<tr>
<th>Minimum</th>
<th>Maximum</th>
<th>Value-1</th>
<th>Value-2</th>
<th>Normalized Distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>16</td>
<td>8</td>
<td>14</td>
<td>0.4</td>
</tr>
<tr>
<td>1</td>
<td>32</td>
<td>8</td>
<td>14</td>
<td>0.194</td>
</tr>
<tr>
<td>1</td>
<td>64</td>
<td>8</td>
<td>14</td>
<td>0.095</td>
</tr>
<tr>
<td>1</td>
<td>128</td>
<td>8</td>
<td>14</td>
<td>0.047</td>
</tr>
<tr>
<td>1</td>
<td>256</td>
<td>8</td>
<td>14</td>
<td>0.023</td>
</tr>
</tbody>
</table>

Table 4-4: Effect of Scale on Normalized Distance

In a network intrusion detection problem, there are several data points that the scale will be known ahead of time. Some points are in the scale of 0~1, while some others are in 0~255. However, as an example, in the KDD’99 dataset which is commonly used to evaluate anomaly detection approaches, 17 of the 42 data points have an undefined scale. Furthermore, the scale in the training data is not the same as the scale in the
live data. Therefore, scaling done during the training process will not be accurate when applied to the live
data. We will propose an adaptive input normalization approach that will automatically tune scaling
parameters online based on the observed traffic patterns.

4.3.1 Dynamic Input Normalization Calculation

Over a scaling interval \( t-1..t \), we maintain a running mean and running standard deviation for each
feature in the pre-normalized input space [74]. In the interval \( t..t+1 \) we maintain bins, one bin for each
feature mean \( \pm 10 \) standard deviations. As new features are observed, for each bin, we keep a count of the
number of values observed in that bin as well as the min/max values for the bins. For bins below the mean
we track minimum value for bins above the mean we track maximum value. At the end of interval \( t+1 \),
upper and lower bins are selected as the first bin above and below the mean that cumulatively account for
the upper and lower 99 percentiles. \textit{Observed\_min} and \textit{Observed\_max} are set to the minimum value in
the lower bin and the maximum value in the upper bin respectively. After each scaling interval, scaling
adjustments are made according to equations:

\[
max(t + n) = min(t) + R(t) \cdot \left(1 - e^{-\frac{1}{\alpha}}\right) + \beta_{\text{max}}(t). \tag{4-1}
\]

\[
min(t + n) = max(t) - R(t) \cdot \left(1 - e^{-\frac{1}{\alpha}}\right) - \beta_{\text{min}}(t). \tag{4-2}
\]

In equations (4-1) and (4-2), we have

\[
max_{\Delta}(t) = \begin{cases} 
obs_{\text{max}}(t) - max(t) & obs_{\text{max}}(t) > max(t) \\
0 & obs_{\text{max}}(t) \leq max(t)
\end{cases} \tag{4-3}
\]

\[
min_{\Delta}(t) = \begin{cases} 
min(t) - obs_{\text{min}}(t) & obs_{\text{min}}(t) \leq min(t) \\
0 & obs_{\text{min}}(t) > min(t)
\end{cases} \tag{4-4}
\]

\[
R(t) = max(t) - min(t). \tag{4-5}
\]
Control parameters $\alpha$ and $\beta$ have values $> 0$. $\alpha$ controls the exponential growth/decay rate. $\beta$ controls how differences between the current used min/max and the observed min/max is used.

4.4 Performance Evaluation

4.4.1 Experimental Setup

To evaluate the effect of our data collection techniques and the behavior of our preprocessing methods we developed two simulators using the C++ programming language and the winpcap API [165]. The two simulators model a Flow Creation module and an Intrusion Detection module. The Flow Creation Module accepts raw pcap formatted trace files and processes them into datafiles consisting of augmented flow records using the methods described in this chapter. It is parameterized with three values: inactive timeout, active timeout and augmentation history duration. Inactive timeout is set to 15 seconds, active timeout is set to 60 seconds, and changes to augmentation history duration are evaluated. We process the raw trace files from both the TD-SIM and MAWI datasets and generate corresponding datafiles consisting of augmented flow records. If a flow contains packets belonging to an attack, the flow is labeled “malicious” otherwise it is labeled “benign”. Malicious flows are also labeled with information pertaining to the nature of the attack.

The Intrusion Detection Module models two intrusion detection systems: one based on an Adaptive Support Vector Machine, and the other based on an Adaptive Growing Hierarchical Self Organizing Map. These algorithms are discussed in detail in Chapter 5. The detection module accepts a datafile of augmented flow records as input and preprocesses the data according to the methods described in section 4.3.1. After preprocessing, for each flow record in the data set, the module makes a malicious/benign prediction based on one of the two detection algorithms. The confusion matrix is detailed in Table 4-5.
After processing all flow records three metrics are reported: Accuracy, Precision, and Recall. The three metrics are calculated according to:

\[
\text{Accuracy} = \frac{TP + TN}{TP + FP + TN + FN}
\]

\[
\text{Precision} = \frac{TP}{TP + FP}
\]

\[
\text{Recall} = \frac{TP}{TP + FN}
\]

### 4.4.2 Impact of Augmentation History Duration

We start our evaluation by examining the impact of Augmentation History duration on basic performance metrics. The purpose of this evaluation is to demonstrate the impact of Augmentation History duration on Accuracy, Precision, and Recall and to empirically determine appropriate settings for future extensive evaluations of our proposed detection algorithms. In these experiments, inactive time out is fixed at 15 seconds and active timeout is fixed at 60 seconds. Augmentation History Duration is varied from 30 seconds to 300 seconds. Count-min data-structures are fixed at \(\{w = 2^{15}, d = 5\}\). Figure 4-6(a)(b)(c)(d) show the impact that changing the duration of the augmentation history has on basic performance metrics. In Figure 4-6(a)(b) we apply the A-GHSOM and A-SVM detection engines to the TD-SIM Dataset. In Figure 4-6(c)(d) we apply the A-GHSOM and A-SVM detection engines to the MAWI Dataset. In Chapter 5 we propose and discuss the basic A-GHSOM and A-SVM detection algorithms as well as several proposed enhancements to the basic models. In these tests we do not use any of the proposed enhancements. The results in Figure 4-6 are obtained using only the baseline models. In all cases we see

<table>
<thead>
<tr>
<th>Prediction</th>
<th>Flow Label</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Malicious</td>
<td>Malicious</td>
<td>True Positive (TP)</td>
</tr>
<tr>
<td>Malicious</td>
<td>Benign</td>
<td>False Positive (FP)</td>
</tr>
<tr>
<td>Benign</td>
<td>Malicious</td>
<td>False Negative (FN)</td>
</tr>
<tr>
<td>Benign</td>
<td>Benign</td>
<td>True Negative (TN)</td>
</tr>
</tbody>
</table>

Table 4-5: Basic Prediction Confusion Matrix
that while there are variations in corresponding Precision and Recall metrics, overall accuracy is generally increased as the history duration is increased.

Figure 4-6: Impact of Augmentation History Duration on Basic Performance Metrics

In all four figures, performance stabilizes at between 210 and 240 seconds. Maximum overall performance is reached when history duration is 300 seconds. However, we also observe diminishing returns. Performance increases after 210 seconds are negligible. In Table 2-1 we see that when history duration increases from 30 seconds to 240 seconds
accuracy increases are between 2.2 and 4.4% while increasing history duration from 240 to 300 seconds only achieves gains <1%.

Recall that the values used to augment the flow records are estimates calculated using count min data-structures. As the size of the history increases, we must also increase the size of the estimating data structures in order to maintain similar levels of estimation accuracy. Failure to grow the count-min structures adequately will reduce estimation accuracy and contribute to the diminishing returns ultimately leading to decreased performance. Larger data-structures mean more memory is required. In order to achieve reasonable accuracy within resource constraints we cap the history duration size at 300 seconds for the remainder of the experiments in this thesis.

In all cases, with essentially no significant modifications made to proposed basic detection approaches we are able to achieve baseline performance results of >90% accuracy. Precision performance ranges from 59% to 99% and Recall performance ranges from 75% to 95% In chapters 5, 6, and 7 we propose and discuss enhanced detection methods that consistently achieve >95% Precision and >99% Recall on augmented flow records generated from both TD-SIM and MAWI test data when combined into an integrated model.

4.4.3 Behavior of Dynamic Input Normalization

In this section we examine the behavior of dynamic input normalization. Figure 4-7 demonstrates the relationship between the observed min/max values and the adapted min/max values. They represent a subset of the “test-bytes” feature of the KDD dataset. The x-axis represents the time interval that the measurement was recorded. The y-axis represents the min or max value in bytes. The plotted series represent the relationship between the observed values and the adapted values with $\alpha = 0.05$ and $\beta = 1.0$. The adapted values are used for scaling during the normalization process.
(a) Alpha = 0.2 Beta = 1.0

(b) Alpha = 0.2 Beta = 0.2

(c) Alpha = 0.5, Beta = 0.5

(d) Alpha = 0.5, Beta = 1.0

(e) Alpha = 0.2, Beta = 1.0

(f) Alpha = 0.2, Beta = 0.2

(g) Alpha = 0.5, Beta = 0.2

(h) Alpha = 0.5, Beta = 1.0

Figure 4-7: Example Adapted Scaling Values
When $\alpha$ and $\beta$ values are set to > 0.2, growth and decay rates are very abrupt. In these circumstances, single outliers can cause significant changes to pre-processing normalization effects. When $\alpha$ and $\beta$ values are set to mild values, adaptations are more subtle and normalization values are effected less by outlier values. In subsequent sections we will examine in detail the effect these parameters have on prediction performance.

4.5 Summary and Discussion

In this chapter we have presented our ideas for preprocessing raw network traffic for anomaly detection. First we present a method for generating flow records that augments traditional flow data with additional information. This information is both flow specific and network wide. We use an array of Count-Min data structures to collect and process estimates in a way that is suitable for real-time data collection. Our method provides both volume metric and limited data distribution information for presentation and analysis. We perform initial empirical evaluation of temporal time out values. We find that 99% of all flows worked with are less than 60 seconds in duration. We also find stable performance and limited increase in basic performance after 210 seconds. The count-min sketch is able to perform and estimates in fixed time and space. However, because it is an estimation technique, as history duration increases, accuracy bounds decrease. We therefore limit history duration to 300 seconds. For the remainder of this dissertation, we choose temporal values for flow creation as 15 second in-active timeout, 60 second active-timeout and 300 second augmentation history window.

In this chapter we also present a mechanism for dynamic input normalization. Using this process we adjust the scale by which incoming data is normalized before being presented to the detection engines. We examine the behavior of the control parameters and see that both mild and aggressive updates can be achieved. As input normalization in the context of this dissertation will impact the relative difference between individual data points in an input vector we expect dynamic normalization to have a significant impact to predicting normal or anomalous traffic.

In the remainder of this dissertation we will develop two detection engines capable of making accurate predictions using the augmented flow records created in accordance with the methods outlined in this
chapter. We will also examine in detail the impact of dynamic input normalization under both mild and aggressive conditions on precision and recall rates.
Chapter 5.

**Adaptive Anomaly Detection Models**

In this chapter we develop two adaptive detection models. The first is based on an Adaptive Growing Hierarchical Self Organizing Map. The Self Organizing Map is a popular anomaly detection approach. One variant of the SOM is the GHSOM. In a traditional SOM, the map size and dimensionality is fixed. In a GHSOM, the size and dimensionality of the map architecture are determined during the training phase. We further modify this approach by developing algorithms for adjusting the map on-line.

In order to test the elements of this thesis on multiple detection engines, we propose a second model based on Support Vector Machines. While SVM application to network anomaly detection has been an active research topic for some time, current proposed models still suffer from several common deficiencies such as sensitivity to tuning parameters, high false positive rate and inability to adapt to drifting concepts without off-line retraining. Our model is based on one-class SVM that is solved by a modified Sequential Minimal Optimization algorithm. We enhance traditional incremental SVM by maintaining an on-line feedback based auxiliary training set. The set is adapted based on operator feedback and enables the system to respond to concept drift and significantly enhances predictive capability while at the same time remaining efficient enough for real-time adaptation.

In this chapter we also propose a novel evolving alert aggregation mechanism. The detection engines proposed make predictions on individual augmented flow records. However, many network events consist of multiple flows. We propose a simple yet novel method based on discrete value estimation that correlates individual alerts and uses this information to identify network events. This approach significantly increases the overall capability of the intrusion detection scheme.

The remainder of this chapter is organized as follows. First in section 5.1 we present algorithms for an Adaptive Growing Hierarchical Self Organizing Map detection engine. We evaluate the model on the KDD and TD-SIM data sets. Next in section 5.2 we present algorithms for an Adaptive Support Vector Machine detection engine. We also perform evaluation against the KDD and TD-SIM data sets. Then in section 5.3 propose and develop an evolving alert aggregation module and examine its ability to enhance
performance of the two detection engines. Finally, in section 5.4 we examine the ability of the detection models and the alert correlation module to perform and adapt on-line over extended periods by conducting evaluation on the MAWILab dataset.

5.1 Adaptive Growing Hierarchal Self Organizing Map (A-GHSOM)

In this section we propose and develop an adaptive detection engine based on the Growing Hierarchical Self Organizing Map. The Self Organizing Map is a popular anomaly detection approach [69, 71, 109, 117, 120, 129, 156]. One variant of the SOM is the GHSOM [109]. GHSOMs have been proposed and used effectively to classify high dimensional data [118]. In a GHSOM, the size and dimensionality of the map architecture are determined during the training phase. The initial map size is very small, usually a single layer 2x2. During the training process, the map is grown both vertically and horizontally until the training process is complete. After each training iteration, the deviation of the input data (quantization error) is computed. The map is grown horizontally by adding rows and columns to the map to reduce quantization error. The map is grown vertically by adding child layers to parent layers that exceed the pre specified maximum dimensionality of each map. This process continues until the quantization error of the map is below an established threshold. The purpose of growing the map in this manner is to ensure it is well suited to the input set. After training, regions of the map are labeled as either normal or attack regions. It is important to note that at the end of the training process, each layer and sub-layer can have a different number of maps and sub-maps of varying dimensionality as illustrated in Figure 5-1.
During live operation input vectors representing network events are presented for evaluation and classification. Using some vector distance formula, they are matched to the best matching unit (BMU) in the map. The quantization error (QE) between the input vector and the BMU is observed. If the BMU is in an attack region the connection is considered attack. If the BMU is in a normal region and the QE is below some threshold, the connection is considered normal. If the QE exceeds some threshold, the connection is considered anomalous and treated as an attack.

**Dynamic Size and Dimensionality:** After offline training, the GHSOM will have a finite structure consisting of a hierarchy of fixed size SOMs. Each region in each map will be labeled to either predict benign or malicious network activity. This structure will be well suited to the training set and subsequent connections that match patterns in the training set. Existing GHSOM approaches to anomaly detection do not adapt the map dimensionality after offline training. However, over time, the normal behavior of the network will evolve. As new patterns are consistently observed that the model is not well suited to predict by other adaptations, there will be a need to adapt the size and or dimensionality of the hierarchy itself. For example when there is new legitimate behavior that is consistently flagged as anomalous or new attacks that are matched to normal regions of the map. We propose methods to grow the underlying structure on-
line to account for these new patterns. The larger the map, the higher the resolution and greater the number of distinct patterns it can identify. However, do to resource constraints we cannot simply grow the model indefinitely. We must strike a balance between performance and accuracy. We propose methods for identifying under-utilized regions of the map and devise self-pruning algorithms. We will apply the algorithms we develop for Dynamic Input Normalization. In Chapter 6, we develop a performance model that includes a mechanism for confidence forwarding. To support that model we include confidence estimation algorithms in the A-GHSOM.

5.1.1 AGHSOM Model

Our A-GHSOM approach develops an online learning process using feedback-based quantization error thresholds to adapt the system over time to changes in the input data. It also uses confidence estimation to identify traffic with low prediction confidence. The following subsections give the design details for important components of the A-GHSOM detection framework.

5.1.1.1 Adaptive Thresholds

We propose methods for adaptive thresholds to enable and adjust anomaly detection capability. In addition we propose methods for identifying anomalies in individual data points within a vector. Existing techniques use parameters such as the mean quantization error of the vector, or Euclidean distance between vectors [109, 118]. However, these approaches do not always identify anomaly conditions because they distribute the anomaly across the vector.

For example, consider three vectors representing two connections and one weight vector. Connection-1 and Connection-2 are each compared to the weight vector. Figure 5-2 shows the individual quantization errors for each data point in the vectors. If we were to use mean quantization error or Euclidian distance to evaluate Connection-1 and Connection-2, we would find that the mean quantization error and Euclidian distance are higher for Connection-1. However, it is obvious that Connection-2 has an anomalous value in the vector. For effective anomaly detection, we are interested in not only the aggregate small discrepancies, but also identifying large single discrepancies. We develop threshold formulas that are able to identify both small aggregate anomalies and single point anomalies. We then adaptively adjust these thresholds to account for concept drift.
5.1.1.2 Feedback Based Threshold Adaptation:

A-GHSOM is further enhanced by use of feedback-based quantization error threshold adaptation. It adaptively adjusts thresholds for each node as input patterns are applied and add new nodes when appropriate. Each node is assigned two initial threshold parameters. $\tau_1$ is used to calculate the threshold error value. The threshold error value is calculated using the quantization error vector as follows:

$$TEV_j = \sum_{k=0}^{n} f(k)$$

$$f(k) = \begin{cases} \frac{qte_j}{\tau_2} & \text{if } k > \tau_1 \\ 0 & \text{if } k \leq \tau_1 \end{cases}$$

Here, $\tau_1$ is a threshold that controls how closely an element of an input vector must match an element of a weight vector before it is no longer factored into the threshold error value. $\tau_2$ is used as an upper limit on the acceptable total quantization error and used to calculate the quantization error boundary (QEB) for a selected node as follows.

$$QEB_j = \begin{cases} 0 & \text{if } tqe_j < \tau_2 \\ 1 & \text{if } tqe_j \geq \tau_2 \end{cases}$$

Figure 5-2: Example Quantization Error of two Input Vectors
Where $t_q e_j = \sum_{k=1}^{n} q e_j^k$. During live intrusion detection, input patterns are applied to the A-GHSOM and the best matching unit is selected. However, because of the size of the problem space, there is no guarantee that the best matching unit is a good match to the input pattern. The only guarantee is that it is a better match than all the other nodes in the neural network. The threshold error value and quantization error boundary are used to determine if the connection being processed is within thresholds. If and only if they are identically equal to 0, is the pattern considered within thresholds.

Each node in the final A-GHSOM is marked either “normal”, “unmarked”, or “anomalous”. As patterns are examined, they are mapped to one of the three node types and considered within thresholds or not. Results are then used to make predictions to identify the suspected connection type, either “normal” or “anomalous”. Any pattern that is not within thresholds is identified as a suspected attack.

Even though its best matching unit is labeled “normal”, the fact that its threshold error value or its quantization error boundary is greater than 0 means this connection is anomalous in nature and it is assumed to be an attack. Likewise, any patterns mapped to a best matching unit of type “unmarked” are patterns that have not been previously identified and are suspected attack.

The actual connection type is then compared to the suspected connection type and a result of “correct” or “incorrect” is identified. The distinction of correct or incorrect is made based on operator feedback. In the absence of feedback, the system assumes that its prediction was correct. Table 3 details the adaptation rules based on perceived prediction result.

<table>
<thead>
<tr>
<th>Best Matching Unit Type</th>
<th>Within Thresholds</th>
<th>Prediction</th>
<th>Result</th>
<th>Adaptation Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attack</td>
<td>Yes</td>
<td>Attack</td>
<td>Correct</td>
<td>No Action</td>
</tr>
<tr>
<td>Attack</td>
<td>No</td>
<td>Attack</td>
<td>Correct</td>
<td>No Action</td>
</tr>
<tr>
<td>Attack</td>
<td>Yes</td>
<td>Attack</td>
<td>Incorrect</td>
<td>Grow Network</td>
</tr>
<tr>
<td>Attack</td>
<td>No</td>
<td>Attack</td>
<td>Incorrect</td>
<td>Grow Network</td>
</tr>
<tr>
<td>Normal</td>
<td>Yes</td>
<td>Normal</td>
<td>Correct</td>
<td>No Action</td>
</tr>
<tr>
<td>Normal</td>
<td>No</td>
<td>Attack</td>
<td>Correct</td>
<td>No Action</td>
</tr>
<tr>
<td>Normal</td>
<td>Yes</td>
<td>Normal</td>
<td>Incorrect</td>
<td>Lower $\tau_\nu, \tau_2$</td>
</tr>
<tr>
<td>Normal</td>
<td>No</td>
<td>Attack</td>
<td>Incorrect</td>
<td>Raise $\tau_\nu, \tau_2$</td>
</tr>
</tbody>
</table>
Table 5-1: Feedback Based Adaptation Rules

<table>
<thead>
<tr>
<th>Unmarked</th>
<th>Yes</th>
<th>Attack</th>
<th>Correct</th>
<th>No Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unmarked</td>
<td>No</td>
<td>Attack</td>
<td>Correct</td>
<td>Label Node</td>
</tr>
<tr>
<td>Unmarked</td>
<td>Yes</td>
<td>Attack</td>
<td>Incorrect</td>
<td>Label Node</td>
</tr>
<tr>
<td>Unmarked</td>
<td>No</td>
<td>Attack</td>
<td>Incorrect</td>
<td>Label Node Raise $\tau_1, \tau_2$</td>
</tr>
</tbody>
</table>

5.1.1.3 Dynamic Pruning

In the adaptation rules detailed in Table 5-1: Feedback Based Adaptation Rules make the A-GHSOM dynamic in one direction only. That is over time, the network grows to accommodate concept drift. Given an infinite amount of time, the model will grow continuously until resource utilization can get out of control. The size of the A-GHSOM directly affects its performance. The larger the network, the more diverse the event set it is able to accurately classify. However, the larger the network becomes, the more resources it consumes. In order to limit resource consumption we establish a maximum size $\varrho$. We define the current size of the A-GHSOM as the total number of nodes in the hierarchy designated $\sigma$. Every time an event is processed, the parent SOM of the best matching node and all of that SOM’s ancestors are marked with a freshness value equal to the current time stamp. When the A-GHSOM size exceeds $\varrho$, stale branches of the hierarchy are pruned so that at all times, size $\sigma \leq \varrho$. Algorithm 5-1 summarizes the process.
By pruning the network, we increase efficiency however, we sacrifice classification resolution. The existing nodes are forced to generalize more and more, and prediction errors may occur.

### 5.1.1.4 A-GHSOM Prediction Confidence

In Chapter 6 we define a performance model that requires detection engines to be able to report a confidence rating in their predictions. In this section we develop that mechanism for the A-GHSOM method. It monitors the neuron consistency and accuracy and uses those measures to develop a neuron confidence rating.

In a traditional GHSOM, individual neurons and regions of the map are organized to classify connections. All similar connections matched to identical regions of the map will be classified as the same. However, in A-GHSOM, due to the addition of dynamic input normalization and feedback based threshold adaptation, this is not the case. It is possible for two connections to be matched in an identical region on the map, but be placed into different classification categories depending on their threshold error value and quantization error boundary value. Thus, under varying conditions, agitation of an individual neuron will

---

**Algorithm 5-1: GHSOM Pruning**

```plaintext
1:    evaluate $\sigma$
2:    if ($\sigma > q$)
3:        repeat
4:            using breadth first search, identify map $\Gamma$ with oldest time stamp.
5:        if $\Gamma$ has a child
6:            repeat
7:                $\Gamma \leftarrow$ stalest child of $\Gamma$
8:            until $\Gamma$ has no children
9:        end if
10:    delete $\Gamma$
11: until ($\sigma \leq q$)
12: end if
```

prompt varying predictions. Neurons that are essentially trained to agitate on normal connections can indicate attack and vice versa.

We monitor the frequency that each neuron predicts each connection class and use this condition to calculate a consistency rating. After each connection is processed at time $t$ and a prediction made by neuron $j$, consistency for the predicting node is calculated according to

\[
\text{Attack}_t^j = \frac{\lambda (\text{Attack}_{t-1}^j + \text{Predict}_{\text{Attack}})}{\lambda + 1}
\]

\[
\text{Normal}_t^j = \frac{\lambda (\text{Normal}_{t-1}^j + \text{Predict}_{\text{Normal}})}{\lambda + 1}
\]

\[
\text{Consistency}_t^j = \frac{\max(\text{Attack}_t^j, \text{Normal}_t^j)}{\text{Attack}_t^j + \text{Normal}_t^j}
\]

where $\lambda$ is the size of the history, $\text{Predict}_{\text{Attack}} = 1$ if an attack is predicted and 0 otherwise, and $\text{Predict}_{\text{Normal}} = 1 - \text{Predict}_{\text{Attack}}$.

Because A-GHSOM also expects limited feedback from a network operator, we are able to estimate an accuracy rating for each node. As the amount of feedback received by the system is limited, the system assumes that its predictions are correct in the absence of feedback. $\text{Predict}_{\text{Accurate}}$ is equal to 0 if corrective feedback is received and 1 otherwise. Accuracy for predicting node $j$ is calculated according to.

\[
\text{Acc}_t^j = \frac{\lambda (\text{Acc}_{t-1}^j) + \text{Predict}_{\text{Accurate}}}{\lambda + 1}
\]

We use a combination of consistency and accuracy to calculate a confidence rating as follows.

\[
\text{Confidence}_t^j = \frac{\lambda (\text{Confidence}_{t-1}^j) + \frac{1}{2} (\text{Acc}_t^j + \text{Consistency}_t^j)}{\lambda + 1}
\]

As the number of predictions by a node increases, three possible consistency scenarios can occur.

1) A node trained “attack” consistently makes “attack” predictions.

2) The node trained “normal” consistently makes “normal” predictions.
3) A node trained “normal” makes a significant number of “attack” predictions.

Situations 1 and 2 are not a concern if the accuracy is also high. If an “attack” neuron consistently makes inaccurate “attack” predictions, new neurons will be added to that region trained to predict “normal” and correct the deficiency. Likewise, if a “normal” neuron consistently makes inaccurate “normal” predictions, new neurons will be added to that region trained to predict “attack” and again, the deficiency is corrected. The third situation, however, is a concern.

According to the adaptation rules of A-GHSOM model, all neurons should eventually stabilize to a consistent state. However, it assumes that normal connections and connections of attack types are differentiable based on the available data. If there exist normal connections and attack connections with identical or extremely similar connection vectors, the adaptation process will be stalled and inconsistent predictions will be made. We identify nodes that have low confidence and flag them for special handling by subsequent processors.

5.1.2 A-GHSOM Performance Evaluation

5.1.2.1 Experimental Setup

The experimental setup for the A-GHSOM model is similar to the set up for evaluation in chapter 4. We use a simulator we have developed using the c++ programing language. The simulator accepts a sequence of vectored data as input and calculates performance metrics as output. The input vectors are either individual connection records from the KDD dataset, or augmented flow records generated from the TD-SIM and MAWI datasets according to the methods described in Chapter 4. When Dynamic Input Normalization is applied, the data are preprocessed according to the methods in section 4.3.1. The model is trained on a small random sample of records consisting of a small percent of the testing data. Records selected for training are not re-used for testing. We again use the performance metrics of Precision, Recall, and Accuracy according to the confusion matrix in Table 4-5. We evaluate the impact of Dynamic Input Normalization, the effectiveness of adaptation.

5.1.2.2 Baseline Performance

The purpose of this experiment is to evaluate the baseline performance of the A-GHSOM without any enhancements. In this trial, no dynamic normalization is used, no feedback is provided and no adaptation is
implemented. We conduct evaluation on the KDD, TD-SIM, and MAWI datasets. For the KDD trial, the model is trained on a random sample of records consisting of 5 percent of the “10 Percent training set”. For the TD-SIM trial, the model is trained on 0.01 percent of the records. Training records are not reused as test records. For the MAWI trial, the model is trained on 0.05% of the records from the 1 January, 2010 data file. For all three trials, normalization scale is based on the min/max values found in the training records. \( \tau_1 \) and \( \tau_2 \) are set to 0.1 and 0.5 respectively and \( q \) is fixed at 6000 nodes. Table 4 gives the baseline performance of the A-GHSOM on the three datasets.

<table>
<thead>
<tr>
<th>Data</th>
<th>Precision</th>
<th>Recall</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>KDD</td>
<td>0.991394</td>
<td>0.915859</td>
<td>0.922150</td>
</tr>
<tr>
<td>TD-SIM</td>
<td>0.840150</td>
<td>0.926699</td>
<td>0.968148</td>
</tr>
<tr>
<td>MAWI</td>
<td>0.883175</td>
<td>0.959432</td>
<td>0.924512</td>
</tr>
</tbody>
</table>

Table 5-2: Baseline A-GHSOM Performance

5.1.2.3 Impact of Dynamic Input Normalization

The purpose of this experiment is to evaluate the impact of dynamic input normalization on A-GHSOM performance. In this experiment, no on-line adaptation utilized. \( \tau_1 \) and \( \tau_2 \) are set to 0.1 and 0.5 respectively and are not adapted. \( q \) is again fixed at 6000 nodes. The model is trained with initial normalization scaling conducted using min/max from the training records. We then apply dynamic input normalization adjusting the \( \alpha \) and \( \beta \) parameters from 0.1 to 1.0 and evaluate on all three data sets.
Figure 5-3: Impact of Dynamic Input Normalization on the baseline A-GHSOM performance

Figure 5-3 shows the impact of Dynamic Input Normalization on the baseline A-GHSOM performance. In this figure, color shifts from red to green as the values are increased. We note that specific impact is different for each data set. However a common pattern is that increased values of $\alpha$ will make the system more aggressive increasing Recall, but at the same time decreasing Precision. This is because the system is more sensitive to new measurements for min/max values and scales aggressively. Increased values of $\beta$ are able to temper this behavior to some degree making the system retreat scale when larger max values or small min values are observed. However, false positive quickly gets out of control when $\alpha$ is aggressive. Without additional adaptation, careful tuning of these parameters is required to keep overall performance under control.
Table 5-3: Summary of A-SVM Performance after Dynamic Scaling

Table 5-5 summarizes performance changes caused by dynamic scaling. We observe that significant increases to Recall can be achieved by scaling alone. However, Precision can be drastically reduced. In the case of TD-SIM Aggressive, while Recall increased from 77% to 96%, the system is so sensitive that Precision is less than 30%. While TD-SIM Mild Scaling sees a mild increase to Recall with only minor decrease to Precision. It is clear that minor changes to these parameters can have significant effects. Tuning them is an important task. In additional experiments in this chapter we use adaptation maintain high Recall while keeping Precision under control. In chapter 6 we propose and methods for improving performance further by including confidence forwarding and propose a method for tuning model parameters.

5.1.2.4 Impact of On-Line Adaptation

The purpose if this experiment is to evaluate the impact of on-line adaptation. In the previous experiment we observed that while Dynamic input normalization was able to influence the sensitivity of the model, the model could be too sensitive even under mild scaling changes. In this experiment we add on-line adaptation to the Dynamically Scaled trials. We perform tests on the KDD and the TD-SIM data sets only.

In this experiment we train the model as we have in previous experiments. We utilize Dynamic Input Normalization consistent with the “Mild scaling” and “Aggressive Scaling” discussed previously. We adapt the model using simulated operator feedback. The feedback rate is adjusted from 0.01 to 0.1 That is, the chance that the operator will identify incorrect predictions and respond to the model is between 1% and 10%. Figure 5-10 and Figure 5-11 show the impact on trials conducted with the KDD data and TD-SIM data respectively.
Figure 5-4: Impact of Adaptation on Dynamically Scaled A-GHSOM – KDD Data
We observe that on both datasets, on-line adaptation is able to help keep precision under control while at the same time increasing Recall over statically scaled non-adapted performance. In the KDD set, Recall is...
increased up to 97.7% while Precision is maintained up to ~96% in the mild case and ~94% in the aggressive case. While precision is still impacted in a negative way due to dynamic input normalization, it is kept under control by adaptation. We are still able to achieve an increase in Recall performance.

Likewise in the TD-SIM trials, we observe increases to Recall rates up to ~94% and ~96% with mild and aggressive dynamic input normalization. Again, on-line adaptation is able to keep Precision under control. In the aggressive case, adaptation is able to raise Precision from less than 30% to up to over 70%.

5.1.2.5 Summary and Discussion

In this section we have proposed algorithms for an A-GHSOM detection module. We provide a mechanism to respond to operator feedback and adapt on-line. We demonstrated the initial impact that Dynamic Input Normalization can have on this detection approach. While Recall is enhanced, this enhancement is at the expense of Precision. We demonstrated that on-line adaptation can mitigate this to some degree. However, without additional modifications precision is still not consistently at a very high rate. In section 5.3 of this chapter we will discuss an alert correlation method than will enhance performance further. We also proposed a method for calculating confidence estimates in A-GHSOM predictions. In Chapter 6 we will propose a performance model that will capitalize on these calculations. In section 5.4 of this chapter we will further examine the effectiveness of on-line adaptation of the A-GHSOM model along with the A-SVM model we develop in the next section.

5.2 Adaptive Support Vector Machine

In this section we propose and develop ideas for a network intrusion detection model based on Adaptive Support Vector Machines. Support Vector Machines have been applied to both signature based intrusion detection [61, 111] and anomaly detection [37, 107]. While SVM application to network anomaly detection has been an active research topic for some time, current proposed models still suffer from several common deficiencies such as sensitivity to tuning parameters, high false positive rate and inability to adapt to drifting concepts without off line retraining.

In this section, we propose an Adaptive SVM for network anomaly detection. Our model is based on one-class SVM that is solved by a modified Sequential Minimal Optimization algorithm, [115]. We address the identified issues by proposing 3 significant enhancements.
First we enhance traditional incremental SVM by maintaining an on-line feedback based auxiliary training set. The set is adapted based on operator feedback and enables the system to respond to concept drift and significantly enhances predictive capability while at the same time remaining efficient enough for real-time adaptation. Second, we enable the use of dynamic input normalization process on-line. Third, we propose a novel method for calculating prediction confidence in one-class SVMs to identify traffic patterns that are beyond the capability of a content oblivious anomaly based system to correctly classify. While traditional SVM confidence approaches use proximity to the hyperplane as the primary indicator, our approach identifies portions of the hyperplane that are most and least likely to correctly correlate anomalies with malicious intent.

5.2.1 Background and overview

5.2.1.1 One-class support vector machines

A Support vector machine performs classification by mapping input space to high-dimensional feature space and then finding a hyperplane that separates input vectors with maximal margin. In one-class SVMs, rather than separating the input data into classes, the goal is to identify the support of a distribution by identifying a small region in feature space that contains most of the examples and have that region be as far from the origin as possible [132]. The hyperplane is calculated from unlabeled training data and then future input vectors are classified based on their relationship to the hyperplane. Vectors separated from the origin are considered normal; vectors on the same side of the hyperplane as the origin are considered anomalous. Let the training set \( \{x_1, x_2, \ldots, x_l\} \in \chi \) be a compact subset of \( \mathbb{R}^n \) where \( l \in \mathbb{N} \) is the number of observations. Let \( \Phi \) be a feature map \( \chi \rightarrow F \) such that inner product can be computed by kernel evaluation \( K(x, y) = (\Phi(x) \cdot \Phi(y)) \) where \( K(x, y) \) is some kernel. In our model we use an RBF kernel where the Kernel Function is

\[
K(x, y) = \exp\left(-\frac{\|x - y\|^2}{2\sigma^2}\right)
\]

Calculating the hyperplane specified by the hyperplane's normal vector in feature space \( \mathbf{w} \) and offset from the origin \( \rho \) involves solving the quadratic programming problem:
\[ \min \quad \frac{1}{2}\|w\|^2 + \frac{1}{vl}\sum_i \xi_i - \rho \]

\[ w \in F, \xi \in \mathbb{R}^l, \rho \in \mathbb{R} \] (5-6)

subject to \((w \cdot \Phi(x_i)) \geq \rho - \xi_i, \xi_i \geq 0\)

Where \(\xi\) are slack variables, and \(v\) is a parameter that controls the tradeoff between maximizing the distance from the origin and the amount of vectors contained in the region created by the hyperplane. Once solved, the decision function for each \(x\) is:

\[ f(x) = \text{sgn}((w \cdot \phi(x)) - \rho) \] (5-7)

Introducing a Lagrangian and rewriting the optimization in terms of multipliers \(\lambda\) the optimization becomes:

\[ \min \quad \frac{1}{2} \sum_{ij} \lambda_i \lambda_j K(x_i, x_j) \]

\[ \lambda \] (5-8)

subject to \(0 \leq \lambda_i \leq \frac{1}{vl}, \sum_i \lambda_i = 1\) (5-9)

Once optimized, \(\rho\) can be recovered from the multipliers. For any \(\lambda_i\) where \(0 < \lambda_i < \frac{1}{vl}\)

\[ \rho = (w \cdot \phi(x_i)) = \sum_{j} \lambda_j K(x_j, x_i) \] (5-10)

In terms of the multipliers the objective function \(O(x)\) and decision function \(f(x)\) become:

\[ O(x) = \sum_{j} \lambda_j K(x, x_j) \] (5-11)

\[ f(x) = \text{sgn}(O(x) - \rho) \] (5-12)
5.2.1.2 Sequential Minimal Optimization

We solve the optimization problem with a variation of the SMO algorithm [115, 132, 133]. The strategy of the SMO algorithm is break the very large quadratic programming problem into a series of smallest possible quadratic programming problems that can be individually solved analytically. The memory required for SMO algorithm is linear in the size of the training set and does not require matrix computation.

The Karush-Kuhn-Tucker (KKT) conditions are necessary and sufficient conditions for an optimal point of a positive definite QP problem. QP problem in Eq 5-6 is solved when, for all \(i\) when the objective function \(O\) in Eq 5-11 and \(\lambda\) meet the following KTT conditions:

\[
\begin{align*}
(1) & \quad \lambda_i = 0 \quad \Leftrightarrow \quad O(x_i) - \rho > 0 \\
(2) & \quad 0 < \lambda_i < \frac{1}{vl} \quad \Leftrightarrow \quad O(x_i) - \rho = 0 \\
(3) & \quad \lambda_i = \frac{1}{vl} \quad \Leftrightarrow \quad O(x_i) - \rho < 0
\end{align*}
\]

The SMO algorithm works by scanning the training set until a vector violating the KTT conditions \(x_i\) is found. A corresponding candidate \(x_2\) is then selected by a series of heuristics detailed in [115]. The two vector QP is solved by optimizing \(x_i\) and \(x_2\) over \(\lambda_i\) and \(\lambda_2\) with all other variables fixed. Let \(\lambda_i^*\) and \(\lambda_2^*\) = the multipliers before the update step. New \(\lambda_i\) and \(\lambda_2\) are calculated according to the update steps:

\[
\lambda_2 = \lambda_2^* + \frac{O(x_i) - O(x_2)}{K(x_i, x_i) + K(x_2, x_2) - 2K(x_i, x_2)}
\]

\[
\lambda_1 = \lambda_1^* + \lambda_2^* - \lambda_2
\]

If the new multipliers are outside the constraint \([0..1/vl]\), \(\lambda_2\) is projected onto the region allowed and \(\lambda_1\) is recalculated. \(\rho\) is recalculated after every update.

5.2.2 Adaptive SVM Auxiliary Training Set

Traditionally, in an optimized SVM, most of the multipliers are = 0 and the corresponding training vectors have no effect on the objective function and are discarded. In our model, we do not completely discard the
training vectors. Rather we maintain an auxiliary set and use it to dynamically update the hyperplane on-line as the model responds to operator feedback. The initial auxiliary set is defined by the tuple \((\chi, L, T, A)\) where \(\chi\) is the set of example vectors, \(L\) is the set of lagrange multipliers, \(T\) is the age of each vector and \(A\) is an operator defined parameter limiting the maximum size of the auxiliary set. As the model processes traffic and makes predictions, the auxiliary set is updated according to operator feedback. When false positives are reported vectors are removed from the auxiliary set. When missed anomalies are reported, new vectors are added to the auxiliary set. After each change to the auxiliary set, the model is re-optimized on-line. Because each change to the auxiliary set causes only minor changes to the optimized solution, use of the SMO algorithm enables us to re-optimize on-line. The auxiliary set is updated through individual additions and deletions. When the auxiliary set is modified, the constraints \(\sum_{i} \lambda_i = 1\) and \(0 < \lambda < \frac{1}{v_l}\) must be taken into consideration.

5.2.2.1 Adding a new vector to the auxiliary set:

A vector \(x_k\) is added by setting \(\chi = \chi \cup x_k\), \(\lambda_k = 0\), and \(l = l + 1\). After addition, \(\sum \lambda\) is unaffected and \(\frac{1}{v_l} \text{new} < \frac{1}{v_l} \text{old}\). If the new vector is inside the region created by the hyperplane, its addition will have minimal effect on the overall solution as it does not violate the KTT conditions. If the new vector is outside the region created by the hyperplane it violates KTT condition (1) and must be re-optimized. Reducing the number of vectors in the auxiliary set reduces the constraint \(\frac{1}{v_l}\). Any multiplier that was bounded by the \(\frac{1}{v_l}\) constraint prior to vector removal will now violate the constraint, as will any lambda that was sufficiently close to the boundary. We attempt to minimize the number of passes required by the SMO algorithm by spreading the excess lambda values heuristically prior to calling the SMO algorithm. The process is summarized in algorithm Algorithm 5-2:
Algorithm 5-2: Heuristic Approach for correcting newly bound vectors and distributing excess lambda

1: \( \text{Set } v := v + 1 \)
2: \( \text{For every } k \text{ such that } \lambda_k > \frac{1}{vl} \)
3: \( \text{Set } \lambda_k^\Delta := \lambda_k - \frac{1}{vl} \)
4: \( \text{While } \lambda_k^\Delta > 0 \)
5: \( \text{Select } j \text{ from } L \text{ where } \lambda_j < \frac{1}{vl} \text{ and } K(j,k) \text{ is minimized} \)
6: \( \text{Set } \lambda_j^\Delta := \min\left(\lambda_k^\Delta + \frac{1}{vl}\right) \)
7: \( \text{Set } \lambda_k^\Delta := \lambda_k^\Delta - \lambda_j^\Delta \)
8: \( \text{End while} \)
9: \( \text{Set } \lambda_k := \frac{1}{vl} \)
10: \( \text{End For} \)

Algorithm 5-3 is a random approach that does not attempt to minimize passes of the SMO algorithm. We will use this algorithm for comparison to our approach only.

Algorithm 5-3: Random Approach for correcting newly bound vectors

1: \( \text{Set } l := l + 1 \)
2: \( \text{For every } k \text{ such that } \lambda_k > \frac{1}{vl} \)
3: \( \text{Set } \lambda_k^\Delta := \lambda_k - \frac{1}{vl} \)
4: \( \text{Randomly Select } j \text{ from } L \text{ where } \lambda_k^\Delta + \lambda_j \leq \frac{1}{vl} \)
5: \( \text{Set } \lambda_j := \lambda_k^\Delta + \lambda_j \)
6: \( \text{Set } \lambda_k := \frac{1}{vl} \)
7: \( \text{End For} \)

If \( l > A \), then the existing vector with \( \max(t_i) \) is selected for removal from the auxiliary set.
5.2.2.2 Removing a vector from the auxiliary set

A vector \( x_k \) is removed from the auxiliary by setting \( \chi = \chi - x_k \) and setting \( l = l - 1 \). If \( \lambda_k \neq 0 \) then its removal will affect \( \sum \lambda \). In order to maintain the constraint \( \sum \lambda = 1 \), \( \lambda_k \) must be redistributed to \( L \).

Additionally, any \( \lambda \) that was previously bounded by \( \frac{1}{vl} \) constraint is no longer bounded and must be re-optimized. Our goal when redistributing the excess lambda is to minimize the amount of times the SMO algorithm will have to cycle over the auxiliary set in order to re-optimize the model. We heuristically do this by spreading the excess caused by the removal of a vector to the remaining vectors closest to the removed vector in the feature space. This process is summarized in algorithm 5-4:

<table>
<thead>
<tr>
<th><strong>Algorithm 5-4</strong></th>
<th>Heuristic Approach for deleting vector and distributing excess Lambda value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1:</td>
<td>Set ( \lambda_k^\Delta := \lambda_k )</td>
</tr>
<tr>
<td>2:</td>
<td>While ( \lambda_k^\Delta &gt; 0 )</td>
</tr>
<tr>
<td>3:</td>
<td>Select ( j ) from ( L ) where ( \lambda_j &lt; \frac{1}{vl} ) and ( K(j,k) ) is minimized</td>
</tr>
<tr>
<td>4:</td>
<td>Set ( \lambda_j^\Delta := \min \left( \lambda_k^\Delta + \lambda_j, \frac{1}{vl} \right) )</td>
</tr>
<tr>
<td>5:</td>
<td>Set ( \lambda_k^\Delta := \lambda_k^\Delta - \lambda_j^\Delta )</td>
</tr>
<tr>
<td>6:</td>
<td>End while</td>
</tr>
<tr>
<td>7:</td>
<td>Remove ( k ) from ( L )</td>
</tr>
</tbody>
</table>

Algorithm 5-5 is a random approach for removing a vector and is used for comparison to our approach only.

<table>
<thead>
<tr>
<th><strong>Algorithm 5-5:</strong></th>
<th>Random Approach for deleting vector and distributing excess Lambda value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1:</td>
<td>Set ( \lambda_k^\Delta := \lambda_k )</td>
</tr>
<tr>
<td>2:</td>
<td>While ( \lambda_k^\Delta &gt; 0 )</td>
</tr>
<tr>
<td>3:</td>
<td>Randomly Select ( j ) from ( L ) where ( \lambda_j &lt; \frac{1}{vl} )</td>
</tr>
<tr>
<td>4:</td>
<td>Set ( \lambda_j^\Delta := \min \left( \lambda_k^\Delta + \lambda_j, \frac{1}{vl} \right) )</td>
</tr>
<tr>
<td>5:</td>
<td>Set ( \lambda_k^\Delta := \lambda_k^\Delta - \lambda_j^\Delta )</td>
</tr>
<tr>
<td>6:</td>
<td>End while</td>
</tr>
<tr>
<td>7:</td>
<td>Remove ( k ) from ( L )</td>
</tr>
</tbody>
</table>
5.2.3 Support Vector Machine Confidence Rating for Anomaly Detection

Support vector machines do not innately provide confidence intervals or posterior probabilities on classifications predictions. In SVM classification, hyperplane proximity or decision function scaling methods have been proposed for determining individual prediction confidence [114, 127]. Ji et al proposed a method for assigning confidence scores less than 1 to examples most likely to be support vectors in an effort to improve classification accuracy [67, 68]. Each training example is augmented with a confidence score. In [67] A priori example confidence was based on human ability to classify examples. In [68] confidence is automatically assigned by proximity to the decision boundary. The augmented training examples are used to improve classification accuracy. We propose an approach to SVM confidence that also uses examples augmented with confidence scores. We update confidence scores based on operator feedback on-line and only use decision boundary proximity to determine the relative impact on overall confidence of individual feedback events. Support vectors closest to the example receiving feedback are affected the most by the event.

In SVMs, confidence is commonly calculated as some function of the distance to the hyperplane. In one class SVM, the core idea is to identify the level of “support” a distribution has. In [132, 133] it was shown that the parameter $v$ is both an upper bound on the fraction of anomalies (training vectors outside the estimated region) and a lower bound on the fraction of support vectors (training vectors with corresponding non-zero multipliers). When confidence measures based on margin or distance to hyperplane are used, a direct relationship between degree of normality and degree of accuracy is assumed. In network anomaly detection, there is an implied assumption that anomalous events are also malicious. This is not always the case. Concept drift, network changes, changes to the user behavior, hardware upgrades, routing changes, etc..., can all cause events that while anomalous are not necessary malicious. If we were to design a confidence calculation based primarily on degree of normality we would not be able to differentiate between benign/malicious anomalous events or between benign/malicious normal events. Instead, we propose a confidence calculation that is designed to identify the portions of the hyperplane that are most responsible for prediction errors. In our model, we make the assumption that the operator has the ability to provide feedback to the model. We do not assume a particular approach, only that a mechanism exists. For specific methods, refer to [51, 119, 169].
When a connection record is evaluated, we maintain an index of support vectors sorted according to the input records kernel distance from each support vector. If corrective feedback is received, each vector is assigned “blame” for the error according to this index. Support vectors closest to the misclassified input vector in kernel space are assigned the most blame. This blame is used to calculate each support vectors level responsibility for previous prediction errors and individual support vector confidence. When new predictions are made, prediction confidence is calculated as a weighted combination of individual support vector confidence with SVs closest to the input vector receiving the most weight.

Let $\Psi$ be the set of support vectors. For each support vector $x_i, i \in \mathbb{N}, i < |\Psi|$, a history queue $q_i$ and a responsibility score $r_i$ is maintained. We also maintain $d$ which is defined and controls depth into $\Psi$ to use to calculate blame respectively. When feedback is received, the values are updated according to equation (5-15) and algorithms Algorithm 5-4 and Algorithm 5-5.

$$f(x_i) = \begin{cases} \frac{d - i}{d} & i \leq d \\ 0 & i > d \end{cases}$$ (5-15)

**Algorithm 5-4: Adjust History Queue and Responsibility Score based on Positive Feedback**

1: Calculate $f(x_i)$ according to (5-15)
2: push($q_i, f(x_i)$)
3: $r_i := r_i - pop(q_i)$

**Algorithm 5-5: Adjust History Queue and Responsibility Score based on Negative Feedback**

1: Calculate $f(x_i)$ according to (5-15)
2: $r_i = r_i + f(x_i)$
3: push($q_i, f(x_i)$)
4: $r_i := r_i - pop(q_i)$
Confidence in a single support vector is calculated \( SV_i^{con} = \frac{r_i}{m} \). When a prediction on input vector \( x \) is made, overall prediction confidence is calculated:

\[
\text{confidence}(x) = \sum_{j=0}^{d-1} \frac{(d - j)SV_j^{con}}{\frac{1}{2}d(d+1)}
\]  

(5-16)

5.2.4 A-SVM Performance Evaluation

5.2.4.1 Experimental Setup

The experimental setup for the A-SVM model is similar to the set up for evaluation of the A-GHSOM. We use a simulator we have developed using the C++ programing language. The simulator accepts a sequence of vectored data as input and calculates performance metrics as output. The input vectors are either individual connection records from the KDD dataset, or augmented flow records generated from the TD-SIM and MAWI datasets. As in the A-GHSOM evaluation, the model is trained on a small random sample of records consisting from the testing data. Records selected for training are not re-used for testing. The metrics calculated are the same as used in evaluating the A-GHSOM model. We evaluate the efficiency of adapting the base auxiliary set, the impact of depth on confidence calculation, as well as baseline and adaptive performance.
5.2.4.2 Efficiency of maintaining Auxiliary Set

The purpose of this experiment is to evaluate the efficiency of the heuristic methods of adding or removing vectors from the auxiliary set on-line. In this experiment we examine the ability of Algorithm 5-2 and Algorithm 5-4 to minimize the number of passes the SMO algorithm must make to optimize the detection engine after additions and deletions.

We perform these tests on the KDD, TD-SIM, and MAWI datasets with no dynamic scaling and no alert correlation. Recall that additions and deletions are only made when feedback is received. We set

![Graphs showing efficiency of adding vector to auxiliary set](image-url)
feedback probability to 5%. That is, 5% of the incorrect predictions made are reported to the detection engine on line and the axillary data set is updated. Figure 5-6 shows the average number of cycles the SMO algorithm must accomplish to re-optimize the model after adding a vector. Results from using our proposed heuristic method are compared to random selection. In Figure 5-6(a)(c) the $v$-parameter is fixed at 0.01 and the Kernel Constant is adjusted from 0.5 to 1.4. In Figure 5-6(b)(d) the Kernel Constant is fixed and $v$ is adjusted from 0.01 to 0.1. By using our heuristic approach, we distribute the $\lambda$ adjustments required due to changes in the axillary set to the example vectors closest to the vector causing the change. This has the effect of approximating the new boundary in a single pass through the auxiliary set. Because the new boundary is already approximated, the optimization process is significantly reduced. On average
we are able to increase efficiency of the on-line additions by approximately 20% in the TD-SIM, and MAWI data sets.

Figure 5-7 shows the average number of cycles the SMO algorithm must accomplish to re-optimize the model after deleting a vector. Results from using our proposed heuristic method are compared to random selection. In Figure 5-7(a)(c) the $v$ parameter is fixed at 0.01 and the Kernel Constant is adjusted from 0.5 to 1.4. In Figure 5-7(b)(d) the Kernel Constant is fixed and $v$ is adjusted from 0.01 to 0.1. As in the case of additions, we are also able to improve efficiency when deleting vectors from the auxiliary set. The new boundary is again approximated in the initial pass through the auxiliary set and the subsequent SMO optimization is reduced. On average we are able to increase efficiency of the on-line deletions by up to 23% TD-SIM, and MAWI data sets.

5.2.5 Impact of Depth Parameter on SVM Confidence

The purpose of this experiment is to evaluate the impact of the depth parameter on A-SVM confidence calculation. Predictions are forwarded when their confidence falls below operator defined threshold. That is, as thresholds are raised, less traffic is forwarded, as thresholds are lowered, more traffic is forwarded. The objective of forwarding is to identify a small percentage of overall traffic for additional processing thereby maximizing precision and recall metrics on the majority of traffic that is processed normally. In this experiment we examine the impact on confidence forwarding of selecting various values for $d$, the confidence depth. Recall that $d$ controls the number of support vectors that are used when assessing responsibility for negative feedback, and the number of support vectors aggregated to assess confidence in each prediction. We experiment using the TD-SIM and MAWI data. Feedback probability is fixed at 5%. Kernel constant is fixed at 1.5, $v$ parameter is fixed at 0.01. We adjust the depth parameter from 1 to 12. We also include $d = \frac{1}{2}|\Psi|$ and $d = |\Psi|$.
Figure 5-8 shows the comparison of the percentage of correct and incorrect predictions with similar confidence. In Chapter 6 we will develop a performance model that utilizes this relationship. We observe that in both cases of malicious predictions and in the case of benign predictions the value chosen for $d$ has an impact on the relationship between correct predictions and incorrect predictions. When $d = 1$, generally more correct predictions have similar confidence to incorrect predictions. In the case of “benign” predictions, as $d$ is initially increased through $d = 4$, the ratio of incorrect to correct predictions also increased. However, for values of $d > 4$, the effectiveness begins to decrease and eventually becomes less effective than $d = 1$. As $d$ approaches $|\Psi|$ the effectiveness is reduced to almost linear. We attribute this
to the fact that “benign” predictions make up the vast majority of predictions. Diffusing confidence over the entire set of support vectors dilutes the rating too much to be effective. Limiting confidence calculations to the few support vectors closest to the flow under consideration enhances the system’s ability to isolate low confidence portions of the decision boundary. In Chapter 6 we will show how this relationship can be used to support our confidence forwarding ideas.

5.2.6 Baseline Performance

The purpose of this experiment is to establish a baseline for A-SVM performance without using any enhancements. In this experiment, no dynamic normalization is used, no feedback is provided and no adaptation is implemented. We conduct evaluation on the KDD, TD-SIM, and MAWI datasets. For the KDD trial, the model is trained on a random sample of records consisting of 0.01 percent of the “10 Percent training set”. For the TD-SIM trial, the model is trained on 0.01 percent of the normal records. Training records are not reused as test records. For the MAWI trial, the model is trained on 0.05% of the normal records from the 1 January, 2010 data file. For all three trials, normalization scale is based on the min/max values found in the training records. We conduct a basic grid search to identify appropriate starting values for the \( v \)-parameter and Kernel Constant.

<table>
<thead>
<tr>
<th>Data Set</th>
<th>V param</th>
<th>Kernel Constant</th>
<th>Precision</th>
<th>Recall</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>KDD</td>
<td>0.01</td>
<td>0.5</td>
<td>98.40%</td>
<td>95.50%</td>
<td>95.10%</td>
</tr>
<tr>
<td>TD-SIM</td>
<td>0.01</td>
<td>1.1</td>
<td>77.80%</td>
<td>77.70%</td>
<td>94.30%</td>
</tr>
<tr>
<td>MAWI</td>
<td>0.04</td>
<td>0.7</td>
<td>99.10%</td>
<td>80.90%</td>
<td>91.10%</td>
</tr>
</tbody>
</table>

Table 5-4: A-SVM Baseline Performance

Table 5-4 shows the base performance of the A-SVM on the three data sets. We observe that while overall accuracy is > 91% in all trials, Recall for the TD-SIM and MAWI trials is only ~78% and ~81% respectively and Precision in the TD-SIM trial is only ~78%. In the following experiments we will evaluate the effect of dynamic normalization, adaptation, and confidence forwarding on performance.
5.2.7 Impact of Dynamic Input Normalization

The purpose of this experiment is to evaluate the impact of dynamic input normalization on A-SVM performance. In this experiment, no on-line adaptation or confidence forwarding is utilized. The \( \alpha \)-parameter and Kernel Constant are set to 0.1 and 0.5 respectively. The model is trained as in previous experiments, with initial normalization scaling conducted using min/max from the training records. We then apply dynamic input normalization adjusting the \( \alpha \) and \( \beta \) parameters from 0.1 to 1.0 and evaluate on all three data sets. Figure 5-9 shows the impact of Dynamic Input Normalization on the baseline A-SVM performance. In this figure, color shifts from red to green as the values are increased. We note that specific impact is different for each data set. However a common pattern is that increased values of \( \alpha \) will make the system more aggressive increasing Recall, but at the same time decreasing Precision. This is because the system is more sensitive to new measurements for min/max values and scales aggressively.
Increased values of $\beta$ are able to temper this behavior to some degree making the system retreat scale when larger max values or small min values are observed. However, false positive quickly gets out of control when $\alpha$ is aggressive. Without additional adaptation, careful tuning of these parameters is required to keep overall performance under control.

<table>
<thead>
<tr>
<th>Data Set</th>
<th>$\alpha$</th>
<th>$\beta$</th>
<th>Precision</th>
<th>Recall</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>KDD (No Scale)</td>
<td>N/A</td>
<td>N/A</td>
<td>98.40%</td>
<td>95.50%</td>
<td>95.10%</td>
</tr>
<tr>
<td>KDD (Mild Scale)</td>
<td>0.1</td>
<td>0.3</td>
<td>88.10%</td>
<td>96.10%</td>
<td>94.00%</td>
</tr>
<tr>
<td>KDD (Aggressive)</td>
<td>1.0</td>
<td>1.0</td>
<td>89.40%</td>
<td>98.30%</td>
<td>89.20%</td>
</tr>
<tr>
<td>TD-SIM (No Scale)</td>
<td>N/A</td>
<td>N/A</td>
<td>77.80%</td>
<td>77.70%</td>
<td>94.30%</td>
</tr>
<tr>
<td>TD-SIM (Mild Scale)</td>
<td>0.1</td>
<td>0.1</td>
<td>73.80%</td>
<td>90.80%</td>
<td>94.70%</td>
</tr>
<tr>
<td>TD-SIM (Aggressive)</td>
<td>1.0</td>
<td>1.0</td>
<td>48.50%</td>
<td>92.10%</td>
<td>86.40%</td>
</tr>
<tr>
<td>MAWI (No Scale)</td>
<td>N/A</td>
<td>N/A</td>
<td>99.10%</td>
<td>80.90%</td>
<td>91.10%</td>
</tr>
<tr>
<td>MAWI (Mild Scale)</td>
<td>0.1</td>
<td>0.5</td>
<td>87.70%</td>
<td>80.70%</td>
<td>86.20%</td>
</tr>
<tr>
<td>MAWI (Aggressive)</td>
<td>1.0</td>
<td>1.0</td>
<td>84.30%</td>
<td>80.30%</td>
<td>84.40%</td>
</tr>
</tbody>
</table>

Table 5-5: Summary of A-SVM Performance after Dynamic Scaling

Table 5-5 summarizes performance changes caused by dynamic scaling. We include trials with greatest overall Precision as “mild scaling” and trials with greatest overall Recall as “Aggressive”. We observe that significant increases to Recall can be achieved by scaling alone. However, Precision can be drastically reduced. In the case of TD-SIM Aggressive, while Recall increased from 78% to 93%, the system is so sensitive that Precision is only 40%. While TD-SIM Mild Scaling sees a substantial increase to Recall with only minor decrease to Precision. It is clear that minor changes to these parameters can have significant effects. Tuning them is an important task. In additional experiments in this chapter we use adaptation and confidence forwarding to maintain high Recall while keeping Precision under control. In chapter 6 we propose and methods for tuning model parameters.

5.2.8 Impact of Online Adaptation

The purpose if this experiment is to evaluate the impact of on-line adaptation. In the previous experiment we observed that while Dynamic input normalization was able to influence the sensitivity of the model, the model was too sensitive even under mild scaling changes. In this experiment we add on-line adaptation to the Dynamically Scaled trials. We perform tests on the KDD and the TD-SIM data sets only. In section 5.4 we will examine the impact of adaptation in the long term on the MAWI set using trace files ranging over a year’s time.
(a) Mild Scaling / Precision

(b) Aggressive Scaling / Precision

(c) Mild Scaling / Recall

(d) Aggressive Scaling / Recall

(e) Mild Scaling / Accuracy

(f) Aggressive Scaling / Accuracy

Figure 5-10: Impact of Adaptation on Dynamically Scaled A-SVM – KDD Data
Figure 5-11: Impact of Adaptation on Dynamically Scaled A-SVM – TD-SIM Data
In this experiment we train the model as we have in previous experiments. We utilize Dynamic Input Normalization consistent with the “mild scaling” and “Aggressive Scaling” discussed previously. As we have done with the A-GHSOM trials we adapt the model using simulated operator feedback. The feedback rate is adjusted from 0.01 to 0.1. That is, the chance that the operator will identify incorrect predictions and respond to the model is between 1% and 10%. Figure 5-10 and Figure 5-11 show the impact on trials conducted with the KDD data and TD-SIM data respectively. We observe that in the KDD trials, the combined impact of Dynamic Input Normalization and on-line adaptation is negligible. In both the “mild” and “aggressive” cases the overall impact to Precision, Recall, and Accuracy is not significant compared to the baseline performance. However, we note that baseline model performance on the KDD set is very high. Additionally, the KDD set only consists of approximately 300K records. While we do see some improvement from scaling and adaptation with the SVM model, the trial is not long enough to adapt in a significant way.

In the TD-SIM trials, improvement is substantial. We observe that Precision is markedly improved over both trials with and without Dynamic Normalization. While the increase in Precision is accompanied by a decrease in recall compared to trials run with Dynamic Scaling alone, we see that with the combination of scaling and adaptation, Recall is still improved over baseline trials. In the “mild” case we see improvements to Recall from 2.3% at 10% feedback to 6.8% at 1% feedback. In the “Aggressive” case, we observe improvements to Recall from 4.5% at 10% feedback to 10.2% at 1% feedback. In both “Mild” and “Aggressive” cases we also see improvements to Precision: 6.8% and 2.3% respectively. Overall accuracy is also slightly increased in both cases. While results are improved, they are still not high enough to be acceptable on their own. We also observe that performance is still a tradeoff between Precision and Recall. While one is increased, the other is decreased. In the next section we will propose an alert correlation method that will enhance overall performance. In Chapter 6 we will discuss a performance model based on confidence forwarding that will help move beyond the direct tradeoff between Precision and Recall.

5.2.9 Summary and Discussion

In this section we have proposed algorithms for an A-SVM detection module. We provide a mechanism to respond to operator feedback and adapt on-line. We maintain an auxiliary data set that we are able to
update on-line. We proposed a heuristic method for adding and removing vectors from the dataset using the SMO optimization algorithm. Our approach is approximately 20% more efficient than a random approach. We demonstrated the initial impact that Dynamic Input Normalization can have on this detection approach. While Recall is enhanced, this enhancement is at the expense of Precision. We demonstrated that on-line adaptation can mitigate this to some degree. However, without additional modifications precision is still not consistently at a very high rate. In section 5.3 of this chapter we will discuss an alert correlation method than will enhance performance further. We also proposed a novel method for calculating confidence estimates in A-SVM predictions. We demonstrated that low confidence predictions are primarily incorrect decisions. In Chapter 6 we will propose a performance model that will capitalize on this feature to select a small percentage of calculations for additional processing. In section 5.4 of this chapter we will further examine the effectiveness of on-line adaptation of the A-SVM model.

5.3 Alert Correlation Engine

Performing event correlation is an important step to attack mitigation and response. In order for an operator to effectively respond to a potential threat, they must have more than a generic anomaly alert. Modern networks are constantly under attack. While many of these attacks are not relevant (i.e. an old attack targeting an already patched system), others must be addressed immediately by a variety of attack specific means (patching vulnerable systems, terminating sessions, updating routing tables, rate limiting traffic from specific sources, etc…). Performing this task in signature based systems is trivial. It is merely a matter of matching current behavior with a set of clearly analyzed and defined patterns. Existing anomaly based approaches generally lack sufficient information and methods to properly correlate events. By not providing this information, security operators must manually research each alert to identify potential threat and take appropriate action.

In previous sections of this chapter we proposed two models to perform anomaly detection on individual flows or in the case of the KDD data, individual connection records. However network wide anomalies are rarely isolated to a single flow. Attacks such as worms, scans, and DDoS attacks may generate numerous flows per anomaly event. One scan, while a single event, could generate thousands of flows. In the models previously discussed, in order for detection to be complete, the detection engine
would have to alert on every individual flow in order to achieve 100% coverage of the anomalous event. And even if 100% detection were achieved, there is no innate method to perform event correlation. In order to maximize effectiveness, a network operator analyzing anomaly alerts needs to prioritize. In this section we provide an efficient method for performing evolving anomaly correlation that provides both the operator and downstream automated response engines with information that can be used to prioritize and shape response actions. The correlation engine provides pre-screening of traffic flows, and attempts to correlate/aggregate individual alerts into correlated alerts that can be used by operators or upstream/downstream response measures.

5.3.1 Generating Correlation Records by Alert Aggregation

The correlation engine maintains a windowed history of all alerts $A$ generated by the detection engine. And a set of active correlated alerts $C$ that evolve as traffic is processed and operators respond to network events. We perform event correlation by aggregating similar alerts based on the level of distinct values of the values in the flow header fields of all anomalies detected. Anomalies are categorized by their 5 tuple headers ($SRC\_ADDR$, $DST\_ADDR$, $SRC\_PORT$, $DST\_PORT$, $PROTOCOL$). We correlate/filter alerts by creating correlation records consisting of five fields that correspond to the five header fields and either contain specific values or wild-card characters representing ANY value. The lowest level of correlation is to only aggregate alerts with identical values in all five fields. The highest level of correlation is to aggregate all alerts with matching values in only one field (i.e. all flows with the same destination address).

All correlation records have specific values in either $src\_addr$ or $dst\_addr$ fields. Not including this restriction leads to over filtering and extremely high false positives.

Every time the base detection engine identifies a new alert, the alert is added to $A$, and the correlation engine considers a new correlation record. A windowed history of previous alerts is maintained and the header information in the flow generating the new alert is compared to the header information of previous alerts. All new correlation records begin with all fields set to ANY. The candidate record is then refined based on a comparison of the new alert to previous alerts. Refining the correlation is a two-step process.

First, the base aggregation of $src\_addr$ or $dst\_addr$ is determined. When comparing a new alert $a_{\text{new}}$ to the previous recorded alerts $\{A\}$, a tally of how many occurrences of $a_{\text{src\_addr}}$ and $a_{\text{dst\_addr}}$ are found in $\{A\}$. The field with the greater tally is selected as the initial correlation base.
Second, fields are updated in the candidate record based on the level of distinct value estimation of the remaining header fields found in \{A\}. Fields with a low level of distinct values are added to the correlation record. Fields with a high level of distinct values are not added. When the number of distinct values is high, discrete variance is estimated high. When the number of distinct values is low, discrete variance is estimated low. When variance is high in a particular field, we do not consider that field an identifying feature of the aggregated alerts and therefore do not add it to the correlation. However, when variance is low, the field is considered a significant feature and is added to the correlation record. The correlation record is added to C. Duplicate entries are not allowed. This process is summarized in Algorithm 5-7.

---

**Algorithm 5-6: Create New Correlation Record**

Maintain set \( A \) of the previous N alerts identified by detection engine \( D \)

Receive alert \( a \) on flow \( F \) identified by \( \{a_{src\_addr}, a_{dst\_addr}, a_{src\_port}, a_{dst\_port}, a_{protocol}\} \)

Set \( A := A \cup a \)

Create candidate record \( c = \{\text{ANY, ANY, ANY, ANY}\} \)

If \( \text{count}(a_{src\_addr} \in A_{src\_addr}) \geq \text{count}(a_{dst\_addr} \in A_{dst\_addr}) \)

Set \( c_{src\_addr} := a_{src\_addr} \)

Set \( A' := \{a' \mid a' \in A, a'_{src\_addr} = a_{src\_addr}\} \)

Else

Set \( c_{dst\_addr} := a_{dst\_addr} \)

Set \( A' := \{a' \mid a' \in A, a'_{dst\_addr} = a_{dst\_addr}\} \)

End If

For every header field \( S \in \{src\_addr, dst\_addr, src\_port, dst\_port, protocol\} \)

If \( \text{distinct\_values}(A') > 5 \)

\( c_s = a_s \)

End If

End For

Set \( C := C \cup c \)
Like our previous models, this approach also responds to user feedback. When false positives are identified, matching alerts are removed from the history and the correlated alerts are adjusted according to Algorithm ALGORITHM 5-7. Likewise, when missed anomalies are recorded, alerts are generated based on the flows identified as false negatives. New correlated records are then generated according to Algorithm ALGORITHM 5-6.

This approach is inherently evolving. The first time a flow is recorded the unique values for header fields in $A$ all 1. They will all be selected as specific values for $c$ making $c$ match only alerts that have equal values in all fields. As additional alerts are identified, alerts that are identical to the first will generate no additional correlations. However, as alerts are identified which are only partially similar to previous alerts, the variance of individual fields will grow and will ultimately generate new correlated alerts identifying events that are beyond single flows, but consist of several flows (i.e. scans, worms, DoS attacks, etc…)

Aggregation decisions are made based on total count estimates and distinct value estimates. We perform our estimations by using two main data structures. First, a history queue capturing the header fields of the last $N$ anomalous flows provides a windowed approach when creating new correlated alerts. Second, we use two multi-dimensional hash structures to estimate both the count of similar source and destination addresses as the flow under consideration as well as the count of unique values of the correlation sub fields. We use one structure for source addresses and one for destination addresses. The first layer of the hash counts src/dst addresses via a 32-bit counter incremented and decremented as

<table>
<thead>
<tr>
<th>ALGORITHM 5-7: Responding to False Positive Feedback</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maintain set $A$ of the previous $N$ alerts identified by detection engine $D$</td>
</tr>
<tr>
<td>Receive False Positive Feedback on alert $a$ identified by ${a_{src_addr}, a_{dst_addr}, a_{src_port}, a_{dst_port}, a_{protocol}}$</td>
</tr>
<tr>
<td>Set $A := A - a$</td>
</tr>
<tr>
<td>For each ${c</td>
</tr>
<tr>
<td>Set $C := C - c$</td>
</tr>
<tr>
<td>End for</td>
</tr>
</tbody>
</table>
anomalous flows are processed through the history queue. The second dimension is an array of four bit-vectors attached to each non-empty cell in the parent hash. Each bit-vector represents an individual sub-field. These bit-vectors perform a simple distinct value estimation similar to the approach in [162]. Figure 5-12 shows a schematic of these structures.

When a new anomalous flow is identified, its header tuple is first added to the history queue and the values in the source-correlation hash and destination-correlation hash are incremented. Collisions in the first layer of the correlation structure are handled by maintaining an ordered list in any cell with more than one active value. Once the first layer is incremented, the bit vector for each sub field is updated. The value of sub field $i$ is hashed and $h(i)$ is set to 1. The goal is to provide an estimate of high or low distinct value count. Thus extremely low error rates are not required and collisions are not significant.

The history queue is also fixed size. If adding a new flow causes an overflow, the oldest flow record is popped from the queue, and the data structures are decremented in reverse of the add sequence. If a src/dst
address cell in the first layer of the correlation structure is decremented to zero, it is removed from the cell. Once the data structures are updated, the correlated alert is calculated according to Algorithm 4

Once correlation records are established, new flows are compared to these records and identified as matching a correlated anomaly by logical AND operations. A flow $f$ matches a correlation header field in $c$ matches every field in $c$. When a field in $c = \text{ANY}$, all values for that field in $f$ will match that field in $c$.

5.3.2 Evaluation of Alert Correlation Engine

5.3.2.1 Experimet Setup
The experimental setup for the alert correlation model is similar to the set up for evaluation of the A-GHSOM and A-SVM detection models. We use the same simulator we have developed using the C++ programming language. The simulator accepts a sequence of vectored data as input and calculates performance metrics as output. The input vectors are augmented flow records generated from the TD-SIM and MAWI datasets. The tests performed on the MAWI dataset were performed on the 1 Jan 2010 data file only. Long term results will be examined in section 5.4. The models are trained as in the previous experiments. We examine the impact using alert correlation as a pre-screening method has on overall performance. We process records from the TD-SIM and MAWI datasets and generate correlated alerts according to the methods described in this section. Once a correlated alert is generated, subsequent flow records that match the alert are blocked. Any flow blocked by the alert engine is considered “Predict Attack.” The same confusion matrix from Table 4-5 is used to determine overall Precision and Recall rates of the augmented detection engines.

5.3.2.2 Impact on Detection Model Performance
In our first experiment we examine the impact using aggregated alerts to pre-screen traffic has on Precision and Recall rates. In Figure 5-13 we consider the results using the TD-SIM and MAWI datasets evaluated by the A-SVM and A-GHSOM models. In each sub-figure, feedback rate is displayed on the x-axis. Aggregation threshold is displayed along the y-axis. The Aggregation threshold is calculated as $2^n$ with $n$ displayed on the axis. As in previous figures, the color is changed from red to green as values increase. We observe a substantial increase in performance in both TD-SIM and MAWI data sets. Recall that the aggregation threshold determines how many distinct values must be observed in a flow header field
The precision can actually be decreased when the aggregation threshold is increased. This is mostly true in the GHSOM model. However, we observe in the SVM model that the feedback rate increases as the aggregation threshold was increased due to the fact that it takes more individual alerts to cause a wildcard field in the correlated alerts. Correlated alerts with wildcards will cast a broader net and will therefore identify/block more traffic. We observe that when the aggregation threshold is low, Recall is increased more than when the threshold is low. This is intuitive. We would likewise expect Precision to increase as the aggregation threshold was increased due to the fact that it takes more individual alerts to generate wildcard fields. This is true in the GHSOM model. However, we observe in the SVM model precision can actually be decreased when the aggregation threshold is increased. This is mostly true in the

- **Precision (GHSOM)**

  - **Aggregation Threshold**
    - 1
    - 2
    - 3
    - 4
    - 5
    - 6
    - 7
    - 8
    - 9
    - 10

  - **Feedback Rate**
    - 1
    - 2
    - 3
    - 4
    - 5
    - 6
    - 7
    - 8
    - 9
    - 10

- **Recall (GHSOM)**

  - **Aggregation Threshold**
    - 1
    - 2
    - 3
    - 4
    - 5
    - 6
    - 7
    - 8
    - 9
    - 10

  - **Feedback Rate**
    - 1
    - 2
    - 3
    - 4
    - 5
    - 6
    - 7
    - 8
    - 9
    - 10
MAWI dataset where there are many more network-wide events to identify (probes, scans, etc…). However, this behavior is also seen in the TD-SIM dataset but is more pronounced when feedback is increased. We attribute this to the fact that in the SVM model, there is a single decision boundary. Both false positive and false negative feedback affect the same boundary. When the aggregation threshold is low, many more attacks are identified before reaching the detection engine. This reduced the amount of false negative feedback the model receives. More of the feedback that the model receives is able to affect the decision boundary in a way that reduced false positives, thus increasing precision. This is more pronounced when more feedback is available. In the GHSOM model, the underlying network is able to grow when appropriate so false negative and false positive feedback examples have a more independent affect on the underlying model. However, as we will see in the following section, in the long term, this effect is also evident in the GHSOM model when correlated alerts are applied. The increase in Recall with only minor increase in actual FP rates actually increases Precision in the long term. In these trials we observe Recall rates using the GHSOM model 97.7% and 99.6% in the TD-SIM and MAWI datasets and Precision rates as high as 87.5% and 93%. Using then SVM model We are able to achieve Recall as high as 99.3% while also achieving Precision of 97.2% in the MAWI dataset. While TD-SIM recall rates reach 96.7% with precision of 88.6%. In chapter 6 we will explore methods to increase this performance further.

5.4 Evaluation of long-term adaptation

5.4.1 Experiment Setup

In these experiments we evaluate the ability of both the A-GHSOM and A-SVM models to adapt over the long term. We experiment using trace files from the MAWI dataset captured in 2010. In order to evaluate the long-term ability to adapt on-line we train our models using data from 1 Jan 2010. We then experiment with tracefiles taken from the remainder of 2010 without any off-line adaptation. We examine results from Jan, Feb, Mar, …, Dec 2010. We compare results from both the A-SVM model and the A-GHSOM model to baseline results captured from 4 other recent methods used to generate the MAWI-Lab labeled data set. We also examine the impact of Dynamic Normalization, On-line Adaptation, Adaptive Alert Aggregation, and Confidence Forwarding on both A-GHSOM, and A-SVM in the long term.

5.4.2 Individual Performance of Base MAWI Detectors
In this trial we establish the baseline for the four individual detectors used to label the MAWI Lab dataset. Recall from section 3.4 that the MAWI-Lab labeled dataset is labeled based on aggregated results from 4 different detectors (PCA, Sketch-Gamma, Hough Transform and Kullback-Leibler). The MAWI-Lab files include two xml files detailing the resulting label. One file identifies anomalies; the other identifies “noticed” streams. Each file identifies the detectors that alerted on the slice in question. Because we use the labeled dataset as “ground-truth” identification of malicious/benign traffic, the aggregated performance of all four detectors is 1.0 recall, 1.0 Precision and 1.0 Accuracy. However, the individual performance of each detector varies. We calculate the performance of individual detectors by parsing the “anomalous_suspicious” and “notice” files and extract individual detector predictions on trace-slice from the MAWI trace files. We then compare each augmented flow generated by our methods from chapter 4. For each detector we record a prediction result on each augmented flow. The confusion matrix for this process is in Table 5-6:

<table>
<thead>
<tr>
<th>Augmented Flow Matched In MAWI Lab File</th>
<th>Detector Identified Flow</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anomalous_Suspicious</td>
<td>Y</td>
<td>TP</td>
</tr>
<tr>
<td>Anomalous_Suspicious</td>
<td>N</td>
<td>FN</td>
</tr>
<tr>
<td>Notice</td>
<td>Y</td>
<td>FP</td>
</tr>
<tr>
<td>Notice</td>
<td>N</td>
<td>TN</td>
</tr>
<tr>
<td>Not Matched</td>
<td>N/A</td>
<td>TN</td>
</tr>
</tbody>
</table>

Table 5-6: Confusion Matrix for Individual MAWI-Lab detectors

For example, if a flow is marked as “anomalous” in the MAWI-Lab files, and detectors 1, 2, 3 identified the flow, but detector 4 did not. We record the individual results as TP for detectors 1, 2, and 3, and as FN for detector 4. Likewise, if the flow is not identified as anomalous in the MAWI-Lab files, but is included in the notice files because only detector 2 identified it, we record individual performance as TN for detectors 1, 3, and 4, and as FP for detector 2. We then calculate overall performance using the metrics defined in section 4.4.1, namely Precision, Recall, and Accuracy.

5.4.3 Baseline Performance of A-SVM and A-GHSOM during long term operation

The purpose of this experiment is to evaluate the baseline long-term performance of our proposed A-GHSOM and A-SVM detectors. In this experiment we do not apply any of the proposed enhancements to either model. The models are trained as before on a small sample of training records. The training records
are taken from the 1 Jan 2010 tracefile. The models are then tested on tracefiles from Jan 2010 through Dec 2010 without off-line or on-line adaptation. We compare A-SVM and A-GHSOM performance to the performance of the four MAWI detectors. We observe that while aggregate performance is authoritative and therefore perfect, individual performance of the published detectors varies. In the following experiments we will demonstrate that A-GHSOM and A-SVM is able to outperform all of the individual detectors and is able to approach aggregated performance. Additionally, our methods are able to achieve

![Baseline Precision of Compared Models](image)

(a) Baseline Precision of Compared Models

![Baseline Recall of Compared Models](image)

(b) Baseline Recall of Compared Models

Figure 5-14: Baseline Long-term performance of A-GHSOM and A-SVM models this performance in near real-time rather than forensically.
We observe that results from our models are comparable to the four MAWI models. A-SVM exceeds all but KL in long-term precision while it is the median performer in Recall and Accuracy. Without adaptation, A-GHSOM model tends to be more aggressive with Recall only slightly less than PCA and KL approaches, but with substantially lower Precision ratings. Without adaptation performance rates quickly deteriorate after only being trained on 1 day of traffic. In subsequent experiments we will evaluate the impact of our proposed enhancements on performance.

5.4.4 Impact of Dynamic Scaling on A-GHSOM and A-SVM long-term Performance

The purpose of this experiment is to evaluate the impact of Dynamic Input Normalization on the long-term performance of A-SVM and A-GHSOM models. In these trials we adjust the normalization scale on-line, but do not perform any additional adaptation or alert aggregation. As before, models are trained as before on a small sample of training records. The training records are taken from the 1 Jan 2010 tracefile. The models are then tested on tracefiles from Jan 2010 through Dec 2010 without off-line or on-line adaptation. Figure 5-15 details the impact dynamic scaling has in the long term. As we observed with the TD-SIM and KDD data sets, dynamic normalization increases sensitivity of the detection models. In both the A-SVM and A-GHSOM trials, Recall is increased at the expense of Precision and overall Accuracy. We also observe that with more aggressive $\alpha$ and $\beta$ parameters, this effect is increased. In order to maintain high recall and also increase precision, we rely on adaptation.
5.4.5 Impact of On-Line Adaptation on Long-term A-SVM and A-GHSOM Performance

The purpose if this experiment is to evaluate the impact of adding on-line adaptation to dynamic input normalization on long-term performance of A-SVM and AGHSOM models. In this experiment we perform
dynamic input normalization in both mild and aggressive settings. We vary feedback probability from 1% to 10%. We do not use adaptive alert aggregation.

Figure 5.16: Impact of On-Line Adaptation to long-term A-SVM and A-GHSOM Performance
In this trial the error-lines indicate the result of feedback rates. The reported line is from 5% feedback. High and low error lines indicate 10% and 1% feedback rates. We observe the initial ability of our proposed detection models to adapt in the long term. With only training on 1 day of traffic and then responding to limited operator feedback, the models are able to increase Recall rates for the duration of the test while keeping Precision more under control. We now examine the combined ability of Dynamic Input Normalization, on-line adaptation, and aggregated alert correlation to maintain performance over extended periods.

5.4.6 Impact of Adaptive Alert Aggregation on long term A-SVM and A-GHSOM performance

The purpose of these experiments is to evaluate the impact of adaptive alert aggregation to the long term performance of A-SVM and A-GHSOM models. In these experiments, we add adaptive alert aggregation to dynamic input normalization and on-line adaptation in the A-SVM and A-GHSOM models. We evaluate the impact the additional blocking has on long-term performance. As in previous trials, models are trained as before on a small sample of training records. The training records are taken from the 1 Jan 2010 tracefile. The models are then tested on tracefiles from Jan 2010 through Dec 2010. We evaluate against both mild and aggressive dynamic scaling. Feedback probability is varied from 1% to 10%, and alerts are aggregated according to the methods discussed in section 5.3.

We observe consistent improvement when the complete model is applied. When Dynamic Input Normalization, On-line adaptation, and Aggregated Alert Correlation are applied together in the long term, we are consistently able to achieve Recall rates above 97% and in many cases exceed 99% for extended periods. We are also able to achieve much higher Precision rates. Even with only 1% feedback we achieve >88% Precision. When feedback rates are 10% both SVM and GHSOM models are able to achieve Precision of nearly 95%.
Figure 5-17: Impact of Adaptation + Alert Aggregation on long-term A-SVM and A-GHSOM Performance
Figure 5-18: Impact of integrated model
5.5 Summary and Discussion

In this chapter we have proposed two adaptive network anomaly detection models and developed a mechanism for evolving alert correlation.

We first proposed a model based on A-GHSOM. In a traditional GHSOM, the size and dimensionality of the map architecture are determined during the training phase. We proposed and developed a method to grow the map on-line. We also proposed methods for adaptive thresholds to enable and adjust anomaly detection capability. In addition we proposed methods for identifying anomalies in individual data points within a vector. Our A-GHSOM is further enhanced by the use of feedback-based quantization error threshold adaptation. It adaptively adjusts thresholds for each node as input patterns are applied and adds new nodes when appropriate. We also develop a confidence estimation for the A-GHSOM method. It monitors the neuron consistency and accuracy and uses those measures to develop a neuron confidence rating.

We also developed a detection model based on A-SVM. Our model is based on one-class SVM that is solved by a modified Sequential Minimal Optimization algorithm. We enhance traditional incremental SVM by maintaining an on-line feedback-based auxiliary training set. The set is adapted based on operator feedback and enables the system to respond to concept drift and significantly enhances predictive capability while at the same time remaining efficient enough for real-time adaptation. We developed a heuristic approach for adding and removing vectors to the auxiliary set that is approximately 20% more efficient than a random approach. Second, we enable the use of dynamic input normalization process on-line. Third, we propose a novel method for calculating prediction confidence in one-class SVMs to identify traffic patterns that are beyond the capability of a content oblivious anomaly-based system to correctly classify. While traditional SVM confidence approaches use proximity to the hyperplane as the primary indicator, our approach identifies portions of the hyperplane that are most and least likely to correctly correlate anomalies with malicious intent. We demonstrated that with this confidence rating a majority of low confidence predictions are also incorrect predictions.

Finally, we proposed and developed an evolving alert aggregation method based on efficient distinct value estimation. We examine patterns of generated alert headers and aggregate individual alerts into correlated events. We tested the combined model on the KDD and TD-SIM dataset to demonstrate
representative behavior of our proposed methods. We also conducted tests on the MAWILab labeled dataset and compared initial results to four other methods. In Figure 5-18 we see the layered results of our proposed methods. We see that as we add Dynamic Input Normalization, On-line adaptation, and Evolving Alert Correlation, each enhancement increases performance. When we compare our method to the four authoritative methods of the MAWILab data, we find that we collectively outperform the individual models. Only the Hough-Transform and KL models maintain precision rates higher or comparable to our methods. However, the KL approach is only able to maintain Recall rates between 70% and 80% while the Hough-Transform method is only able to maintain recall rates at ~50%. Additionally, compared to the four models, our method is able to detect in near real time. The MAWI data files consist of 15 minute traces. The four detectors used to label the MAWILab data and discussed for comparison here examine each data file in its entirety. In real application this means that they must wait for the 15 minute to pass, collect the trace files, and then process them offline. In contradiction our method examines flow files as they are created. With an active timeout window of 60 seconds, our method is able to begin examination in near real time. NOT forensically. The addition of correlated alerts means that we can also respond to traffic immediately rather than off-line after the fact.

The work in this chapter provides the framework for the work in chapter 6. In the following chapter we will provide a framework for performance that will include confidence filtering and forwarding that will further increase performance. Additionally, we propose and develop methods for tuning detection engines on-line.
Chapter 6.

Self-Tuning, Self-Optimizing Anomaly Detection

6.1 Introduction

In this chapter we first introduce and justify the need for confidence forwarding. The need for confidence filtering in an anomaly detection system comes from the fact that compared to signature based systems, while anomaly based systems are able to identify a much larger number of attacks, they are also known for a high false alarm rate. Additionally, sophisticated attack methods are capable to training anomaly detectors to accept malicious activity as normal. The inherent overlap between anomalous <-> benign and normal <-> malicious traffic requires attention. We argue the need for confidence filtering/forwarding in network anomaly detection systems. We develop and integrate novel confidence calculation approaches into our detection models.

Numerous methods have been proposed to accomplish the task of anomaly detection. However despite many years of active research, there are very few anomaly detectors in operational use today. One reason for this is because while published methods have shown promising experimental results, existing anomaly based systems have certain characteristics that make operational deployment challenging. One such characteristic is the sensitivity of detection models to the accurate setting of tuning parameters. Generally, anomaly detection algorithms include a set configurable control parameters. The values assigned to these parameters have a significant effect on the behavior of the model in its current operating environment. In existing methods, these parameters are generally set by researchers directly familiar with the underlying algorithm and are tuned off-line in an ad-hoc fashion, often by trial and error after several experiments in the same environment. It has been found that operational environments have a significant impact on detection results [159, 171].

Once developed, in-order for systems to be suitable for operational deployment, they must be able to operate in a wide variety of highly dynamic environments. Control parameters must be tuned for the target
environment and then re-tuned as the environment changes. It is unrealistic to expect network operators to have detailed knowledge of every detection method. Effective systems must encapsulate the underlying approach such that tuning does not require specific algorithmic knowledge. While we do not expect operators to have detailed knowledge of the detection algorithms, we do expect that they will have knowledge of their network and will be able to monitor the detection system in a general way.

Anomaly detection systems also suffer from inconsistent definition of effective operation. Common performance metrics chosen for model evaluation are base rate accuracy and false positive [149]. Existing methods attempt to maximize the former while minimizing the latter. These two metrics are generally considered a tradeoff and models are tuned to balance this trade off in the particular environment they are evaluated in. Arbitrary goals such as 98% accuracy and 1% false positive rates are commonly achieved. However, [4] points out that even in cases of high accuracy, false positive rates below 1% may still be too high. When overall attack rates are low, even 1% false positive rates can flood the alert log with false alarms thus making the detection system useless to operators who quickly become trained to ignore the alerts. Some work has been done to identify more effective evaluation criteria for detection systems [4, 19, 48, 55, 145]. These works incorporate environment specific information such as cost of failure or likelihood of attack and identify environmentally specific optimal tradeoff between false positive and false negative metrics. While these metrics can present a more meaningful performance criteria implementing them creates a new tuning problem. Optimal operation is more sensitive to changes in the environment. Maintaining performance requires adaptive on-line tuning.

Further, because anomaly based systems compare baseline models to current network behavior, they are subject to concept drift. As normal network behavior evolves, underlying base models must be updated. Recent approaches have been proposed that accomplish this adaptively on-line [39, 161]. However, the ability of these adaptive algorithms to correctly distinguish between evolutions in normal behavior and subtle, well disguised attack patterns is also dependent on effective tuning.

In this chapter we propose and develop a reinforcement learning based controller to act as an interface between the operator and the underlying detection algorithm. We define performance as a tradeoff between precision, recall, and confidence forwarding. When the detection engine has low confidence in its predictions, the subject events are forwarded for additional processing, operator review, or quarantine. In
our approach, operators establish performance objectives and priorities for stated metrics. Feedback is provided to the controller concerning its performance in meeting those objectives. The controller then combines this feedback with observations of the current network state and dynamically updates tuning parameters to maximize performance.

Precision and recall are calculated based on the events that are not forwarded, but are classified by the primary detection engine. By using the idea of confidence forwarding we are able to give operators an extremely flexible tool suitable for operational deployment. Our method is generally independent of the underlying detection method as long as that method meets two requirements: 1) It must be tunable via a finite set of control parameters. 2) It must provide some mechanism for determining confidence in its predictions.

In the reinforcement learning controller, the state space is a combination of the current micro performance objectives and the current network traffic status. It is potentially high dimensional and continuous. We expect the parameter set to be low dimensional with most detection systems tunable by a small number of control parameters. However, those parameters are often tuned to continuous ranges.

Our controller must be capable of handling both high-dimensional continuous state and low dimensional continuous action spaces. We accomplish this by using neural-network function approximation to approximate the state-action-value function. We discretize the action space into a computationally manageable set of tuning directions and combine them with a momentum factor to approximate the action space. The overall function is then trained via an on-line Levenberg-Marquardt backpropagation algorithm.

We test our proposed controller using the A-GHSOM and A-SVM detection methods discussed in chapter Chapter 5. We evaluate performance of our proposed controller using the datasets discussed in chapter Chapter 3. The remainder of this chapter is as follows. First we briefly discuss the challenges of tuning and optimizing anomaly detection systems and propose a model of optimal performance. We evaluate the A-GHSOM and A-SVM detectors under this proposed performance model. Next we develop and discuss our approach for a reinforcement learning system controller. We conclude the chapter with discussion and analysis of the proposed methods.

6.2 Tuning Anomaly Detection Systems
Parameter tuning is a difficult task and little attention has been paid to it [122]. There is a need to obtain appropriate parameter tuning dynamically and automatically in order to accomplish real-time anomaly intrusion detection [60]. In the tuning process we have a set of control parameters $CP$, appropriate for a specific operating environment $OE$, a set of performance metrics $PM$, and a set of performance goals $PG$. Where $CP$ is detection approach specific and includes parameters in the approach that are independent of the data being evaluated and can be set by a controller (i.e. size and dimensionality of a SOM, thresholds in statistical approaches, K in K-NN approaches, etc…). $OE$ is the operating environment the approach is being tuned for (i.e. traffic statistics, attack rate, attack types, micro variance, macro variance, etc…). $PM$ is the set of performance metrics (i.e. detection accuracy, false positive rate, classification accuracy, etc…). $PG$ is the established acceptable values for the performance metrics. In a perfect system, accuracy would always be 100%, false positive rate would always be 0%. In practice this is not the case. $PG$ represents the values that the researcher or operator considers acceptable. Typical values include 95% accuracy with 1% false positive. In our discussion, the primary difference between tuning and optimizing is that tuning focuses on establishing optimal values for the control parameters to meet $PG$ while optimizing focuses on identifying optimal values for $PG$ to reach an overall performance objective. We discuss a model for defining optimal performance in section 6.3.1.

In the tuning process, when given $OE$, we attempt to identify $CP$ such that all objectives of $PG$ are satisfied. In most approaches today, $CP$ is established manually utilizing expert knowledge of both $OE$ and the underlying detection approach. There are two main problems with this tuning method. First, by requiring expert knowledge of the detection method a burden is placed on network operators to learn the detection algorithm for every system they deploy. This is impractical for operational deployment. Second, presetting parameters in $CP$ assumes that $OE$ is stable. This is simply not the case. Network traffic is not stable in time or space. Macro operating conditions are constantly evolving. Workloads increase, attacks grow more sophisticated, and traffic patterns change. As these changes occur, changes to values for $CP$ required to meet $PG$ also occur. Micro changes must also be accounted for. Anomalies that would easily be detected during off hours when background traffic is light may be missed during peak hours when background traffic is very high. Tuning must account for both micro and macro environment changes.

6.3 Optimizing Anomaly Detection Systems
A self-optimizing control problem is one which involves both the explicit determination of the optimal objective and the control actions necessary to achieve that objective. In anomaly detection the traditional optimal objective is maximizing accuracy while minimizing false positive rates. Control actions involve tuning parameters to adjust the sensitivity of the anomaly detector to achieve this objective.

6.3.1 A Model for Optimal Objective for Intrusion Detection

Anomaly detectors model network behavior and flag behavior that exceeds some notion of normal. Typically, an alert is generated when an observed behavior exceeds a predefined threshold or tolerance range. This creates the necessity for threshold selection. In general, a larger threshold may cause false negatives, and a smaller threshold may lead to a false positives. The selection of threshold has a direct impact on a system’s detection coverage [171].

Typically, the tradeoff between false negative and false positive rate is used for evaluating the performance of an anomaly detector as its detection threshold is varied. In many works it is common place to attempt to achieve static and arbitrary performance goals for both false positive rates and false negative rates [149]. A common goal is < 1% false positive rate while maintaining 95% accuracy. However, this has been shown to be an ineffective evaluation approach in some situations. Axelson argued that effective intrusion detection systems will maximize both positive predictive value (PPV) and Negative Predictive Value (NPV) [4]. However, when the probability of attack is low, the low attack rate will dominate NPV calculation such that NPV will be high even in cases of low accuracy. Likewise, even low false alarm rates will dominate PPV calculations so that alert logs are flooded with false alarms. We attempt to overcome this situation by using the metrics of precision, recall, and confidence forwarding metrics that are defined as follows.
Precision and recall are commonly used in the field of document retrieval. Maximizing their value allows us to maintain high PPV and NPV while at the same time ensuring we also maintain high overall detection accuracy. Confidence forwarding is the process of identifying predictions with low confidence and forwarding them for additional processing [51]. Utilizing confidence forwarding allows us to maintain high PPV, NPV, and detection accuracy in any operating environment while simultaneously focusing overhead cost to a small subset of predictions.

6.3.1.1 Anomaly Detection Performance CONFIDENCE

We explore the concept of confidence forwarding further by first presenting an intuitive discussion of a threshold based anomaly detector. The model considers network events that are observed and represented by description vectors and attempts to sort them into a set of event classes according to a well-defined policy. We use this model to further demonstrate the requirement of confidence forwarding to achieve performance flexibility. The detector is represented by the 4 tuple \( \{ X, V, C, D \} \).

\[
Precision = \frac{TruePositives}{TruePositives + FalsePositives} \quad (6-1)
\]

\[
Recall = \frac{TruePositives}{TruePositives + FalseNegatives} \quad (6-2)
\]

\[
Forward = \frac{ConfidenceForwardedEvents}{AllObservedEvents} \quad (6-3)
\]

Precision and recall are commonly used in the field of document retrieval. Maximizing their value allows us to maintain high PPV and NPV while at the same time ensuring we also maintain high overall detection accuracy. Confidence forwarding is the process of identifying predictions with low confidence and forwarding them for additional processing [51]. Utilizing confidence forwarding allows us to maintain high PPV, NPV, and detection accuracy in any operating environment while simultaneously focusing overhead cost to a small subset of predictions.

**6.3.1.1 Anomaly Detection Performance CONFIDENCE**

We explore the concept of confidence forwarding further by first presenting an intuitive discussion of a threshold based anomaly detector. The model considers network events that are observed and represented by description vectors and attempts to sort them into a set of event classes according to a well-defined policy. We use this model to further demonstrate the requirement of confidence forwarding to achieve performance flexibility. The detector is represented by the 4 tuple \( \{ X, V, C, D \} \).

\[
X = \{ x_1, x_2, \ldots, x_n \} \text{ is the set of all possible network events. A network event is the action, state or behavior that we are evaluating to determine if it is malicious or benign. In practice, network events could represent individual packets, packet streams, connections, sessions, etc. Here we use the generic term event.}
\]
Let \( C \) be the set of possible event classes. It is composed of four subsets, \( C_m, C_b, C_a, \) and \( C_n \). 

\( C_m \) is the set of all events constituting malicious intent. \( C_b \) is the set of all events constituting benign intent. \( C_a \) is the set of all events considered anomalous. And \( C_n \) is the set of all events considered normal.

We have \( C_m \cap C_b = \emptyset \) and \( C_a \cap C_n = \emptyset \).

\( V \) is the set of all possible description vectors \( \{v_1, v_2, \ldots, v_k\} \). Where \( v_i \) is a vector of observable values related to \( x_i \) and it represents the subset of the global state that the detection system can observe and is aware of. Vector \( v_i \) is model dependent and is the input to the core algorithm. In practice this typically includes information collected and aggregated from network logs, packet headers, system logs, statistical analysis, etc. It can be raw data or processed data. Any information which the classifier has access to and can use to make predictions about \( x_i \) is contained in \( v_i \).

\( D \) is a classification algorithm. \( D \) accepts \( v_i \) describing event \( x_i \) as input and produces a prediction of class membership as output.

When a network event occurs \( D \) is presented with \( v_i \) and based on some pre-defined criteria, makes a prediction as to the class that the event belongs to. The pre-defined criteria is established either through training or programming. During training or programming, the classifier learns the relationships between \( X, V, \) and \( C \). It is assumed that every event belongs to an event class which results in

\[
\sum_{j=1}^{k} P(x_i \in c_j \mid v_i) = 1 \]

In the best case scenario, the relationship between \( v_i \) and \( C \) is one to one, that is, \( P(x_i \in c_k \mid v_i) = 1 \) where \( c_k \) is some \( c \) in \( C \). However this is not always the case.

Due to generalization during vector construction \( |X| > |V| \). As a simple example, a system that considers network level events, but is unable to consider packet payload during vector construction due to channel encryption, may construct identical vectors for individual events with identical network data but distinct content. It is then possible for a unique vector to describe events belonging to multiple distinct classes. That is \( 0 < P(x_i \in C^M \mid v_i) < 1 \) and \( 0 < P(x_i \in C^B \mid v_i) < 1 \).
Figure 6-1 is a Venn diagram where $M$ represents the set of all possible events that are described by description vectors where given that description vector the probability the event is malicious is greater than 0. $B$ represents the set of all possible events that are described by description vectors where given that description vector the probability that the event is benign is greater than 0. $E$ represents the set of all events that are described by description vectors where given that description vector, sometimes the described event is malicious and sometimes it is benign. Assuming a perfectly trained and tuned system, all vectors describing events in $M-B$ will be correctly predicted as attacks and all vectors describing events in $B-M$ will be correctly predicted as benign. However vectors describing events in $E$ can produce errors.

![Figure 6-1: Classification of Events Based on Description Vectors](image)

Consider a vector $v_i$ describing an event $x_i$ in $E$ where $P(x_i \in C^M | v_i) = 0.99$ and $P(x_i \in C^B | v_i) = 0.01$. In this example, a reasonably trained or programmed classifier will make prediction $p_i \rightarrow x_i \in C^M$. However, 1% of the events described by vectors identical to $v_i$ indicate $x_i \in C^B$ and will generate false positive errors.

At the heart of anomaly based network intrusion detection is the premise that $C_m = C_a$ and $C_b = C_n$. Anomaly detectors do not directly identify membership in $C_m$ or $C_b$. Instead, they identify membership in $C_a$ or $C_n$ and use this infer membership in $C_m$ or $C_b$. However, some prior studies found [49, 57, 148] that many events identified normal are in fact malicious and many events identified anomalous are in fact benign.

Anomaly detectors detect anomalies not necessarily attacks. In Figure 6-2 and Figure 6-3: $A = \text{set of all anomalous events}, N = \text{set of all normal events}, B = \text{set of all benign events}, M = \text{set of all malicious events}$. 

...
events, and \( FP, FN \) = set of all events that can potentially cause false positive and false negative errors respectively. Figure 6-2 and Figure 6-3 illustrate that there is overlap between anomalous behavior and benign intent. There is also overlap between normal behavior and malicious intent. When anomaly detectors raise alerts on every anomaly, false positive errors may occur. When attackers train anomaly detectors to recognize malicious activity as normal, false negatives may occur.

Due to the events that potentially belong to sets \( E, FP, \) and \( FN \), we know in advance that there will be prediction errors. The need for confidence filtering comes from this certainty.

Anomaly detectors use a variety of means to identify how anomalous an event is with the assumption that the more normal it is, the more likely it is to be benign, and the more anomalous it is, the more likely it is to be malicious. It is typical then to adjust decision boundaries such that false positives stay within a reasonable rate while attempting to maximize overall accuracy.
Figure 6-4 illustrates an intuitive tradeoff between false positive and false negative. The X axis represents the degree of normality of a description vector $v_i$. The two curves represent the probability of class membership of $x_i$ given $v_i$. The boundary between the two shaded regions represents the decision boundary of the anomaly detector. Events more anomalous than boundary are flagged as alerts. The shaded area represents events that are normal but malicious and therefore generate missed attacks or events that are anomalous but benign and therefore represent false positives. The exact shape of curves and the size of the shaded regions are dependent on the individual detection model, operating environment, traffic patterns, etc. However, this intuitive illustration is based on core anomaly detection assumptions. Generally, researchers develop detection algorithms to minimize the size of the shaded regions and then choose a decision boundary to meet their particular performance objective. While an effective algorithm may be able to reduce the total error region, it is clear to see that neither sub-region can be further reduced within the total region without increasing its counterpart.
Confidence forwarding expands this model to include three possibilities. With confidence forwarding we add a second decision boundary and divide the error region into three sub-regions. The added region represents events where we have low confidence in the prediction accuracy of the classifier. We expect events in this region to exceed some function of confidence threshold and acknowledge that they require additional processing. We can now consider performance as a tradeoff of three regions and as such are able to make any two regions arbitrarily small at the expense of the third. This model increases the systems prediction flexibility. Operators can now set specific performance goals for both precision and recall that can be achieved by adapting the confidence region of the prediction model as illustrated in Figure 6-5.

In order to properly tune and optimize system performance we need to be able to not only make predictions, but also be able to report what the systems confidence in those predictions is. Our proposed model identifies three possible broad classifications for events, attack, benign, or confidence forwarding. Events in the confidence forwarding category are events that we suspect are either malicious or benign but the model has low confidence in that prediction. These events include normal predictions that are...
anomalous enough that it would be imprudent to let them go unmolested and anomalous events that are normal enough that it would be counterproductive to treat as full attacks.

Confidence forwarding allows us to take additional action on a small subset of events increasing overall system effectiveness while reducing the cost of additional processing to just a small subset of events. The forwarded events can be forwarded to a tier-2 classifier, can be mitigated through quality-of-service mechanisms, or can be completely quarantined until their threat level can be ascertained. The concept of forwarding also gives us an additional level of control while tuning systems. In our model, we allow operators to set performance goals on two metrics and then dynamically tune to ensure the trade-off burden is absorbed by the third.

6.4 Reinforcement Learning for Automated Tuning

We propose a reinforcement learning approach to automated tuning. The reinforcement learning process adapts how a controller makes decisions in an environment in order to maximize cumulative reward. The process is modeled as a Markov decision process (MDP) which consists of a set of environment states and a set of control actions for each state. After each state transition the controller receives a reward and observes the resulting state. The objective of the learning process is to develop a policy \( \pi \) that maximizes the cumulative rewards based on trial and error iterations. The state space consists of all possible operating environments. The action space consists of all possible control actions. A policy is a mapping from states to actions that optimizes the long term reward. The main advantage of using reinforcement learning for control is that the trial and error approach eliminates the need for expert intervention. There is no need to specifically tell the model which parameter to change and how to change it.

6.4.1 Reinforcement Learning Controller

We develop a model for a dynamic tuning system by augmenting base detection algorithms with additional components. The detection engine can be built on any underlying detection algorithm as long as two requirements are met. First, the detection algorithm must be tunable by a finite set of user definable parameters, and second, the algorithm must provide some mechanism for generating a prediction confidence level. This ability is inherent in a variety of anomaly detection approaches [7, 90].
While we do not require the operator to have any knowledge or understanding of the underlying detection algorithm, we do suppose that performance feedback can be provided. In our work we expect that the operator has some mechanism for providing feedback, but we do not suppose a particular approach. For specific methods of providing feedback see [51, 119, 169]. The detection engine uses this feedback to determine the need for tuning.

Figure 6-6: System Controller Schematic

The overall model is illustrated in Figure 6-6. Incoming traffic is passed through a Network Traffic Preprocessor. The primary function of the preprocessor is to generate traffic statistics used to calculate the network state and to generate the description vectors that are passed to the underlying
detection algorithm. The detection algorithm, analyzes the input vector according to its design methodology and passes a prediction and confidence level for that prediction to the confidence filter. The confidence filter uses this information to decide to process the traffic normally, forward for secondary processing, or block the traffic and log an alert. The confidence filter works by comparing the input to two dynamically tuned thresholds, one for alert, one for the absence of alert. If the thresholds are met, the traffic is processed according to the detection algorithm's prediction. If thresholds are not met, traffic is flagged for forwarding.

The performance of the detection algorithm and the thresholds of the confidence filter are automatically tuned on line by a reinforcement learning controller. The controller adjusts an augmented parameter set consisting of the tunable parameters of the underlying detection algorithm and the threshold values of the confidence filter.

As traffic is processed, a performance monitor accepts as input traffic statistics from the network preprocessor, prediction counts from the confidence filter, and performance feedback. It combines this information to generate the system state and reward signal used by reinforcement learning controller. When the detection engine is operating outside of operator established performance criteria, the controller is used to make adjustments. Based on the system state, the controller generates an updated parameter set. The controller uses the reward signal to learn about the effectiveness of its previous updates and to improve future updates. The detection algorithm and confidence filter then make future predictions based on the updated parameter set.

The information collected by the performance monitor is fed to the Reinforcement Learning Dynamic Controller. The Controller receives performance information from the performance monitor and network status information from the Network Monitor. Based on the performance and network information, the controller determines the current state and makes tuning changes to the parameter set. The Anomaly Detector makes predictions based on the new parameters and a reward signal is generated based on the updated performance with the new parameter set. This process is repeated.

The central element of this model is the reinforcement learning controller. 1.5.2 RL Controller. The Reinforcement learning process adapts how the controller takes actions to maximize performance. It is modeled as a Markov Decision Process. The MDP can be viewed as a tuple \((S, A, R, T)\) where:
• \( S \) is the set of all states and \( s_t \in S \) is the state the network is in at time \( t \).

• \( A \) is the set of all possible actions and \( a_t \in A \) is the action the controller performs at time \( t \).

• \( R : S \times A \times S \rightarrow \mathbb{R} \) is the reward function that maps a state \( s_t \), an action \( a_t \) and the resulting state \( s_{t+1} \) into a reward \( r_{t+1} = R(s_t, a_t, s_{t+1}) \).

• \( T : S \times A \times S \rightarrow [0,1] \) is the transition function, where \( T(s_t, a_t, s_{t+1}) \) gives the probability of arriving in state \( s_{t+1} \) when performing action \( a \) in state \( s_t \).

A system can learn by storing values for each state or for each state-action pair [146]. The goal is to learn an action selection policy \( \pi : S \times A \rightarrow [0,1] \) that optimizes the cumulative reward. Here \( \pi_t(s, a) \) gives the probability of selecting action \( a \) in state \( s \) at time \( t \). Formally, we want the controller to optimize the total discounted reward
\[
\sum_{i=0}^{\infty} \gamma^i r_{t+i}
\]
where \( 0 \leq \gamma \leq 1 \) is the discount factor for future reward.

The algorithm we use is based on the Q-learning algorithm [160]. The Q-learning approach involves maintaining values for each state-action pair. The value \( Q(s, a) \) represents the estimated reward of taking action \( a \) while in state \( s \). At each iteration, the current state is observed and a control action is selected. Actions are selected according to an \( \epsilon \) - greedy method. That is, the state-action pair with the highest Q-value is selected most of the time. However, there is a small probability \( \epsilon \), that actions will be selected at random. After each iteration, the model examines the actual reward and the subsequent state. The estimated reward for \( Q(s, a) \) is then updated toward:

\[
Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha [r_{t+1} + \gamma \max_{a} Q(s_{t+1}, a) - Q(s_t, a_t)]
\]  

Where \( \alpha \) is a control parameter that adjusts how sensitive the model is to differences between the expected reward and actual reward, and \( \gamma \) controls how much future expected rewards are factored into the current expected reward.
6.4.1.1 State Space

We describe a state space $S$ consisting of all possible state vectors $s^i$. Where each $s^i$ is the concatenation of a pair of sub-states such that $s^i = (\text{control}^i, \text{operation}^i)$. The control sub-state helps inform the controller of which metric or metrics are in most need of correction. The operating sub-state gives the controller information about the current network operating environment.

The control sub state is a three element vector representing the three performance metrics: precision, recall, and forwarding. Each element of the vector is set to 1 or -1 indicating necessity for improvement of a particular performance metric where 1 indicates improvement required and -1 indicates none. The calculation of control$^i$ is accomplished by use of a artificial neural network (ANN) to map system performance and operator priorities to the possible sub-states. The ANN accepts as input, the value at time $t$ of the three performance functions $F_q(t)$ and three operator defined performance goals $G(q)$. A control string indicating the priority order of the performance functions $H(q)$ is used to calculate weights.

Where $q = (1, 2, 3) = (\text{Precision, Recall, Forwarding})$. At the input layer each performance function is compared with its performance goal and tan-sigmoid activation signal is generated according to:

$$tanh(t) = \frac{e^{F_q(t)G_qG(t)} - e^{-F_q(t)G_qG(t)}}{e^{F_q(t)G_qG(t)} + e^{-F_q(t)G_qG(t)}}$$

(6-5)

These values are permutated and forwarded to eight possible output nodes with elements of $H$ used as a weight values. Based on the output with the highest activation signal, the elements of control$^i$ are set to 1 or -1 accordingly.

The operation sub state operation$^i$ consists of a vector of state variables produced by the Network Preprocessor. The variables include traffic trend data aggregated from the description vectors used by the underlying detection algorithm. This combination of information allows the controller to respond to changes in network conditions while also providing the controller information both specific to and independent from the underlying algorithm. The resulting state space is potentially high dimensional and continuous. In reinforcement learning continuous state spaces can be managed using function approximation methods.
6.4.1.2 Action Space

We define an action space $A$ over a parameter space $P$ as a set of action vectors where each element of an action vector represents a possible control action on the available control parameters in the underlying detection system ($p^1, p^2, ..., p^n$). Each control action consists of a direction and magnitude of change for that control parameter. While we expect the total number of control parameters to be small, they are typically continuous.

Extending reinforcement learning to continuous action spaces is a non-trivial problem. In our proposed approach we manage this issue by initially discretizing the action space into a set of $n$ dimensional action vectors $a = (a_1, a_2, a_3, ..., a_n)$ where $n$ is the number of tunable parameters in the objective model and each element of $a_i = \pm 1$ where the sign represents a decision to increase or decrease $p^k$. We also maintain a single $n$ dimensional vector $M$ to track the magnitude of the tuning changes. Each $m^k$ in $M$ is a dynamic range variable that increases or decreases with the series of sign changes in $a^k_i$. Each $m^k$ is a range limit in the interval $[0: \delta^k]$ where $\delta^k = \min(p^k) - \max(p^k)$ and $\min(p^k), \max(p^k)$ are the minimum and maximum values according to the underlying algorithm. Each time the sign of $a^k_i$ stays the same, $m^k$ is incremented toward $\delta^k$. Each time the sign of $\delta^k$ changes, $m^k$ is reset to 0. On each tuning decision the magnitude of the update $U^k$ is selected randomly $[0:m^k]$.

Then $p_{t+1}^k = p_t^k + (a_i^k \times U^k)$. Ceiling and floor limits of $\min(p^k), \max(p^k)$ are applied to ensure parameters stay within algorithmic limitations.

The result of this approach is that when parameters are tuned on consecutive intervals in the same direction, the magnitude of the adjustments continues to increase. When parameters are tuned on consecutive intervals in the different directions, magnitude of the change is initially reset to 0 and gradually increased as appropriate. Additionally, while the direction of the adjustment to $p^k$ is determined by the discrete action output by the RL tuner, the magnitude of the change is independent of the other parameters in the action vector. It is determined by the aggregate history of sign changes of $p^k$ only.
6.4.1.3 Reward Function

One of the primary objectives of this approach is to ensure that high priority metrics remain within established goals while lower priority metrics bear the tradeoff. The reward function is calculated to place emphasis on priority metric functions remaining within established goals. We first determine if the system is within operating parameters. If the weighted factor of $H$ is used when the system is not within parameters to ensure the priority metrics are accounted for first. We calculate a system score at time $t$:

$$
SCORE_t = \sum_{q=1}^{Q} H(q)(F_q(t) - G(q))
$$

Where $H(q), G(q),$ and $F_q(t)$ are as previously described. If over the interval $t..t+1$ the score has increased then the reward is 1 and 0 otherwise. If the system has crossed the threshold of operating standards from in standards to not in standards, the reward is 0. If the system has crossed from not in standards to in standards, the reward is 1. Otherwise the weighted or non-weighted score is used to determine reward.

6.4.1.4 Action Value Function

In our proposed model, the state space is potentially high dimensional and continuous. We expect the action space to be low dimensional, but also continuous. When the state space space is continuous, function approximation must be used. We use a neural network to approximate the state-action value function.

The state description vector $s$ is fed as input, and the network is trained to output the approximate the action values for that state $Q(s,a)$. We construct an individual network to approximate each possible action. Each network consists of a single output node that approximates the state-action value function for a single action. Each network calculates $Q_{a_i}(s_t,W)$ where $a_1, a_2, ..., a_{2^n}$ are the discretized action vectors, $s_t$ is the state vector at time $t$, and $W$ is the weight vector of the network $Q_i$. The value calculated by $Q_{a_i}(s_t,W)$ is an approximation of the value of taking action $a_i$ when in state $s_t$. This estimate is improved by updating the values of $W$. 
We accomplish updates to $W$ via an on-line Levenberg-Marquardt backproogation approach [56]. The LM algorithm is a combination of gradient decent and a Newton like algorithm based on the idea that gradient decent methods converge more quickly when the weight vectors are far from optimal and Newton's method converges more quickly as the function approaches optimal. In the LM algorithm each update to $W$ is accomplished according to:

$$W_{\epsilon+1} = W_\epsilon - [J^T J + \mu I]^{-1} J^T E$$  \hspace{1cm} (6-7)$$

Where $\epsilon$ is a learning epoch of one or more training samples, $W_{\epsilon+1}$ is the network weight vector at step $\epsilon + 1$, $J$ is the Jacobian matrix of the network, $J^T J$ is an approximation of the Hessian matrix for the network based on the first term of a taylor expansion, $I$ is the identity matrix, $E$ is the error vector of the epoch under consideration, and $\mu$ is an adjustable parameter controlling the tradeoff of the approach. When $\mu \approx 0$, the update is essentially Newtons method. When $\mu$ is very large, the update becomes a gradient decent like approach with a small step size.

A Jacobian is a matrix of all first-order partial derivatives of a vector-valued function. In this case, it is a $y$-by-$z$ matrix, where $y$ is the number of entries in the training set and $z$ is the total number of parameters (weights + biases) of the network. It is created by taking the partial derivatives of each output with respect to each weight, and takes the form:

$$
\begin{bmatrix}
\frac{\partial Q_{a_t}(s_t, W)}{\partial w_1} & \cdots & \frac{\partial Q_{a_t}(s_t, W)}{\partial w_z} \\
\frac{\partial Q_{a_{t+1}}(s_{t+1}, W)}{\partial w_1} & \cdots & \frac{\partial Q_{a_{t+1}}(s_{t+1}, W)}{\partial w_z} \\
\vdots & \ddots & \vdots \\
\frac{\partial F(s_{t+y}, W)}{\partial w_1} & \cdots & \frac{\partial F(s_{t+y}, W)}{\partial w_z}
\end{bmatrix}
$$

Where $Q_{a_t}(s_t, W)$ is the network function evaluated for $s_t$ using the weight vector $W$ and $w_j$ is the $j-th$ element of the weight vector $W$ of the network. The Jacobian is computed via backpropagation.
In traditional LM-backpropagation, the entire training set is considered all at once. After each epoch, if the new sum of squared errors has decreased, $\mu$ is decreased and the iteration ends. If it has not, then the new weights are discarded and the method is repeated with a higher value for $\mu$. The adjustment to $\mu$ is accomplished by multiplying or dividing it by an adjustment factor $B$. The process is repeated until the error decreases. When this happens, the current iteration ends.

We modify the traditional LM algorithm to be suitable for on-line training. Because we are operating on-line, it is not possible to present all training vectors at once. Instead, we use a windowed approach with a window size $l$ where vectors are presented on-line. The Jacobian then becomes an $l \times z$ matrix and is computed in the traditional manner. After $l$ vectors have been presented, the weight updates are calculated and applied and the epoch is over. The adjustment to $\mu$ is made at the beginning of the next epoch. After each $s_i$ is evaluated, an error record is added only to the network approximating the action that was actually selected. Weight updates are only accomplished once a particular network has been updated $l$ times. In this way we avoid updating every network on every tuning event.

### 6.5 Performance Evaluation

#### 6.5.1 Experimental Setup

We perform evaluation on the KDD, TD-SIM and MAWI datasets. We use the A-SVM and A-GHSOM models combined with the data presentation techniques proposed in Chapter 4. Namely augmented Flow records and Dynamic Input Normalization. We also use evolving alert correlation. The detection models are trained in the same fashion as they have been in previous experiments. We first examine the impact of confidence forwarding in a statically tuned environment. We set model parameters and benign and malicious thresholds manually. The confidence thresholds are set via the use of multipliers. Each multiplier is applied to standard deviations from the running mean of prediction confidence. That is, we maintain a running mean and standard deviation calculation for both benign predictions and malicious predictions. For example, if the benign threshold multiplier is set to -1, then all predictions with a confidence value less than -1 standard deviation from the mean are selected for forwarding. This approach enables us to account for differences in confidence calculation approach. After testing static forwarding, we examine the ability of the model to dynamically tune. We sent performance
thresholds and performance goals and evaluate the controllers ability to meet these goals. We train the controller offline using a larger subset of the training data until beneficial tuning is achieved. We then test on the remainder of the traffic.

For our experiments, we configure our controller to self-tune 7 parameters in total. Two dynamic input normalization parameters, $\alpha, \beta$, two confidence threshold parameters, one alert correlation parameter, and two model dependant parameters. In the A-SVM we tune the $\nu$ parameter and Kernel constant. In the A-GHSOM model we tune the default $\tau$ values and the adapting resolution for those parameters.

6.5.1.1 Impact of Forwarding in statically tuned model

In this experiment we examine the impact of confidence forwarding in a statically tuned environment. We first examine results on the TD-SIM dataset. In Figure 6-7 we adjust the malicious confidence multiplier from -1 to 1. In Figure 6-8 we adjust the benign confidence multiplier from -1 to 1. In Figure 6-9 and Figure 6-10 we do the same with the KDD dataset. We then examine the effect on precision, recall and forwarding. We observe that in the case of the TD-SIM dataset, the change is more subtle in the SVM model than in the GHSOM model. In the SVM model we see that when the malicious threshold is adjusted precision is gradually increased while recall is gradually decreased. The forwarding rate is steadily increased throughout this process. When the benign threshold is adjusted from -1 to 1, we again see the performance metrics steadily move in the predicted direction. We can observe substantial improvement while still only selected a small set of records for forwarding. However, in the GHSOM model on the TD-SIM data, on both malicious and benign thresholds we observe very slight increases at first and then extremely sudden increases. This is caused when the multiplier forces the threshold beyond 100% confidence requirement. At that point every prediction is forwarded causing both a sharp increase in performance, but also a sharp increase in forwarded traffic. We do not observe this behavior in the GHSOM model on the KDD dataset with the malicious threshold, but we do with the benign threshold. We attribute this to the way the A-SVM and A-GHSOM models calculate confidence and in the traffic distribution. In the SVM model confidence is calculated using current support vectors. In the GHSOM model confidence is calculated using the predicting node. There can be thousands of nodes. In the SVM model there are a very small number of support vectors. When traffic is disperse in nature, as it is in the
TD-SIM dataset, it may be many connections before a GHSOM node as made enough predictions to reliably report confidence. The TD-SIM trials are over before all nodes have established confidence. In the SVM model however the compact nature of the boundary ensure that support vectors quickly build confidence.

In the case of malicious predictions in the GHSOM on KDD data, over 70% of the connections are attack connections. Additionally, the traffic patterns for these connections are not widely dissimilar. The attack connections are consistently mapped to the same portion of the map thus allowing those nodes to build confidence values. Benign predictions however, do not have the same effect and we again see the sudden increase in forwarding when the multiplier forces the model to forward every connection.

![Figure 6-7: Impact of adjusting malicious confidence threshold using TD-SIM Dataset](image)
Figure 6-8: Impact of adjusting benign confidence threshold using TD-SIM Dataset
We observe similar behavior comparing the SVM model on the KDD data when it comes to malicious predictions. However, we do not see the sudden increase in forwarding until the multiplier is at or about 0 implying that predictions with greater confidence than the mean confidence are affected. We attribute this to fact that we expect most low confidence predictions to be incorrect predictions, and the fact that the majority of connections are attack connections with high accuracy.
6.5.1.2 Confidence Forwarding Impact on prolonged performance.

In this experiment we examine the impact of forwarding on the MAWI data set. We examine impact in the A-GHSOM and A-SVM models and compare results to the four MAWILab methods. In this trial, feedback is fixed at 10%, mild Dynamic Input Normalization is used with $\alpha$ and $\beta$ set to 0.1, Malicious Confidence multiplier is set to -0.5 and Benign Confidence Multiplier is set to -1.0. As in other MAWI trials in this thesis, the models are trained on data from Jan 2010 and then only adapted on-line.

Figure 6-10: Impact of adjusting benign confidence threshold using KDD Dataset
In Figure 6-11 we see that when the models are statically tuned we are able to achieve 99% Recall and ~96% Precision for a consistent period ranging over 2010. We are able to accomplish this by selecting a small percentage of records for forwarding. With the parameters set as they are, the GHSOM model forwards slightly less traffic than the SVM model. In the next section we examine the impact of dynamically controlling these adaptive models.

**6.5.2 Self-Tuning / Self-Optimizing to meet constrained performance objectives**

**6.5.2.1 Impact of Dynamic Tuning on TD-SIM data**

In the following experiments, we will apply confidence forwarding to achieve the desired performance goals, and examine the impact of self-tuning on self-optimization. We augment the base parameter set with malicious and benign confidence thresholds to focus the overhead of additional
processing to a subset of the testing data. With the augmented parameter set, the controller is able to achieve operator defined constraints. We examine the controller's ability to use self-tuning to also self-optimize. We compare performance when confidence thresholds alone are used to achieve constraints with performance achieved via full self-tuning. In these scenarios we consider performance metrics for precision, recall, and confidence forwarding.

![Figure 6-12: Dynamic Tuning on the TD-SIM Dataset (Tradeoff Parameter Forward – SVM)](image)

Performance goals are set for all metrics. In each experiment, the performance priority for one metric is set to 0 and priorities for the other two are equal. We call the metric with priority 0 the tradeoff metric. The performance goals of the other two metrics are treated as constraints. The objective of the controller in these scenarios is to attempt to maximize performance of the tradeoff metric subject to the constraints of the limiting metrics. The performance goal for one of the limiting metrics is fixed and the other is adjusted from 90% to 99%.
In Figure 6-12 and Figure 6-13, Precision and Recall are constrained and forwarding is the trade-off parameter. Figure 6-12 is conducted using the A-SVM Model and Figure 6-13 is conducted using the A-GHSOM model. In Figure 6-12(a)(b) and Figure 6-13(a)(b) Recall is constrained to 95% and the Precision Constraint is adjusted from 90% to 99%. In Figure 6-12(c)(d) and Figure 6-13(c)(d) Precision is constrained to 95% and the Recall Constraint is adjusted from 90% to 99%. We see in all cases that the controller is essentially able to approach operator defined constraints while passing the tradeoff to forwarding. When the constraints become more extreme, (97% - 99%) forwarding is significantly increased in order to meet the objectives. Extreme precision requires the most forwarding with ~20% in the A-SVM model and ~23% in the GHSOM model.

Figure 6-13: Dynamic Tuning on the TD-SIM Dataset (Tradeoff Parameter Forward - GHSOM)
In Figure 6-14 and Figure 6-15, Forwarding is constrained with either Precision or Recall as the trade off parameter. Figure 6-14 is conducted using the A-SVM Model and Figure 6-15 is conducted using the A-GHSOM model. In Figure 6-14 (a)(b) and Figure 6-15 (a)(b) Forwarding is constrained to 15% and the Precision Constraint is adjusted from 90% to 99%. In Figure 6-14 (c)(d) and Figure 6-15 (c)(d) Forwarding is constrained to 15% and the Recall Constraint is adjusted from 90% to 99%. We see in all cases that the controller is essentially able to approach operator defined constraints while passing the tradeoff to the requested parameter. However, with forwarding constrained it is more difficult for the controller to meet constraints and keep the tradeoff parameter as acceptable levels. We see that when precision and forwarding are both constrained and the precision constraint becomes extreme, Recall is significantly decreased. In the SVM model it is reduced to ~76% and in the GHSOM model it is reduced to ~82%.

Figure 6-14: Dynamic Tuning on the TD-SIM Dataset (Precision / Recall Tradeoff - SVM)
This is also true when recall and forwarding are both constrained and the recall constraint becomes extreme, Precision is significantly decreased. In the SVM model it is reduced to \(\sim 87\%\) and in the GHSOM model it is reduced to \(\sim 80\%\). Now GHSOM

![Precision](image1.png)

![Recall](image2.png)

**Figure 6-15: Dynamic Tuning on the TD-SIM Dataset (Precision / Recall Tradeoff - GHSOM)**

### 6.5.2.2 Impact of Dynamic Tuning on MAWI Data

In these experiments we examine the impact of dynamic tuning in the long term using the MAWI data set. As in previous MAWI trials, we train the models using data from 1 Jan 2010 and then test on data from Jan – Dec 2010. In each experiment, the performance priority for one metric is set to 0 and priorities for the other two are equal. We call the metric with priority 0 the tradeoff metric.
Figure 6-16: Dynamic Tuning on the MAWI Set (SVM)
In this experiment we have also added the constraint that forwarding thresholds may not be \( \geq 1.0 \). This will prevent the model from forwarding 100% of the predictions if the confidence forwarding multipliers cause the threshold to exceed 1.0. In Figure 6-16 we examine impact of dynamic tuning on the A-SVM model. Figure 6-17 In Figure 6-16(a)(b) and Figure 6-17(a)(b), the model is statically tuned. In Figure 6-16(c)(d) and Figure 6-17(c)(d) Precision and Recall constraints are set to 99% and Forwarding is the trade off metric. In Figure 6-16(e)(f) and Figure 6-17(e)(f) the precision constraint is set to 99% and Forwarding is constrained to 10%. In Figure 6-16(g)(h) and Figure 6-17(g)(h) Recall constraint is set to 99% and the Forwarding Constraint is set to 5%. We see that in all cases, the model is able to approach the constraints while passing the tradeoff to the desired metric. In both the A-SVM and A-GHSOM models when Precision and Recall are constrained to 99% and forwarding is the tradeoff parameter, the controller is able to approach but not meet the 99% constraint. Precision is stable at about 97% - 98%. Forwarding is initially very high and then tapers off as the system processes more records.

We also observe that in the SVM model, when Recall is the tradeoff parameter Recall performance is substantially affected. The most effective way to control Precision in these models is by forwarding. However, in when forwarding is constrained, the controller tunes other parameters to make the system as insensitive as possible to reduce false positives. In the GHSOM model, the model is still able to achieve reasonable Recall within the 10% forwarding constraint. However, as we see in the SVM model under statically tuned conditions, the forwarding rates in the MAWI set were consistently above 10% the SVM model is unable to achieve reasonable Recall rates within the forwarding constraint and therefore Recall takes the brunt of the tuning changes. We attribute this partially to the way the GHSOM and SVM models adapt. In the SVM model, there is a direct tradeoff during adaptation. Adaptations designed to reduce false positive predictions are more likely to directly reduce true positive predictions. Adaptations in the GHSOM model are more independent from each other and therefore less likely to cause a direct tradeoff.
Figure 6-17: Dynamic Tuning on the MAWI Set (GHSOM)
6.6 Summary and Discussion

In this chapter we introduced and justify the need for confidence forwarding. The need for confidence filtering in an anomaly detection system comes from the fact that compared to signature based systems, while anomaly based systems are able to identify a much larger number of attacks, they are also known for a high false alarm rate. Additionally, the inherent overlap between anomalous <-> benign and normal <-> malicious traffic requires attention. We argued the need for confidence filtering/forwarding in network anomaly detection systems.

Confidence forwarding allows us to take additional action on a small subset of events increasing overall system effectiveness while reducing the cost of additional processing to just a small subset of events. The forwarded events can be forwarded to a tier-2 classifier, can be mitigated through quality-of-service mechanisms, or can be completely quarantined until their threat level can be ascertained. The concept of forwarding also gives us an additional level of control while tuning systems. In our model, we allow operators to set performance goals on two metrics and then dynamically tune to ensure the trade-off burden is absorbed by the third.

We also proposed a reinforcement learning based controller that dynamically tunes detection systems on line. The tuner is platform independent under certain conditions. First the detection model must be tunable with a small set of parameters. In both of our proposed models, the sum of all tuning parameters was seven. Second, the model must have some mechanism for reporting confidence in its predictions. We demonstrated the effectiveness of our approaches on two datasets, the TD-SIM dataset and the MAWI dataset. We examined the impact of confidence forwarding and dynamic tuning on two detection models, one based on A-GHSOM and the other based on A-SVM. We compared our proposed models to four other models on the MAWI data set and consistently exceeded their performance. We demonstrated that exceptional performance (99% Recall and 95% Precision) can be achieved with static manual tuning while selecting 5% to 10% of the records for additional processing. We demonstrated that with dynamic tuning, operators can set performance goals and priorities and the models can be tuned on-line to achieve these objectives.
In this chapter we introduce the concept of confidence forwarding. With confidence forwarding, a small set of records are selected for additional processing. However, we do not directly address how they should be processed. These records can be forwarded for deep packet inspection, flagged for direct operator review, or selected for QoS mitigation. In the next chapter we propose two algorithms that could possibly be used to accomplish QoS mitigation.
Chapter 7.

Automated Response

7.1 Introduction

In this chapter we propose and develop algorithms that can be used for automated response to anomaly events. There are two main components to our approach: correlated alerts, and scalable response. In order for an operator to effectively respond to a potential threat, they must have more than a generic anomaly alert. Alerts must be correlated to guide response actions. In Chapter 5 we proposed a method for correlating individual alerts. In Chapters 5 and 6 we proposed a method for determining prediction confidence and then using the calculation to select a small subset of records for additional processing. In this chapter we propose a model and algorithms that can be used to provide QoS based scalable response.

Most existing detection methods are passive methods. That is, they raise alerts, but they do not actively respond to attacks [37, 72, 169]. While there are some active response anomaly detection systems, they do not provide automatic scalable response actions [125, 140]. Instead, they provide mechanisms for terminating sessions. In this chapter we develop bandwidth sharing and delay differentiation quality of service algorithms that could be used for implementing scalable response mechanisms.

Our approach combines the actions of the detection engines discussed in chapter Chapter 5, with the actions of the correlation engine proposed in Chapter 5 and response engine discussed in this chapter. The correlation engine provides pre-screening of traffic flows, and attempts to correlate/aggregate individual alerts into correlated alerts that can be used by operators or downstream components to guide response actions. Traffic that matches existing correlated alerts is processed according to the taxonomy. Traffic that does not match existing correlated alerts is passed to the detection engine. Alerts generated by the detection engine are fed back to the correlation engine for event correlation. Traffic processed by the Detection and Correlation Engines are labeled in one of three broad ways: Malicious, Benign, or confidence filter. Once traffic is labeled by either the detection engine or correlation engine, it is either
processed normally, or handled by the response engine. Events identified for confidence forwarding will neither be unmolested nor will they be handled solely according to attack taxonomy. Scalable QoS measures can be applied based on a combination of the prediction and the confidence in that prediction. Figure 7-1 illustrates this concept.

![Automated Response Approach](image)

**Figure 7-1: Automated Response Approach**

### 7.2 QoS Provisioning for Scalable Response

A key element of our proposed approach is the idea of confidence filtering. Anomaly detection systems, when compared to signature based systems, are inherently over sensitive. Not all anomalies are attacks. We therefore propose methods for not only making attack predictions, but also learning to assess
confidence in those predictions. When we have low confidence in the predictions that we make, we find that there are one of two possible situations.

- We have predicted that an event is benign. However, we reasonably believe that it may be malicious.
- We have predicted that an event is malicious. However, we reasonably believe that it may be benign.

Traditional methods for responding to malicious activity involve various ways of completely denying access to the suspected attack. We argue that in many circumstances, scalable response is warranted. By using scalable responses against low confidence predictions, we are able to partially mitigate potential attacks while limiting collateral damage. Consider a low confidence attack prediction against a DoS, worm, or information theft attack. Using our proposed approach we are able to implement QoS partial mitigation to the suspected activity. In the case of a DoS attack the partial mitigation could build a firewall against aggressive behavior and hence protect network resources from saturation. In the case of a worm, the partial mitigation could delay spreading of the infection and limit potential damage. In the case of information theft, the partial mitigation could limit the amount of information that could be retrieved. In the case of Cyber Warfare, attack actions are partially mitigated without completely tipping the enemy off to that fact that their activity has been identified. In all these cases, if the attack prediction turns out to be accurate, the delay implemented by the partial mitigation offers security operators time to respond before critical damage has been achieved. If the attack prediction turns out to be incorrect, while some legitimate traffic has been interrupted, overall quality of service has been reduced, but not terminated thus limiting collateral damage.

Our proposed response algorithms allow QoS provisioning in delay, bandwidth utilization and packet loss. These algorithms could be combined with confidence scores and operator preferences to control the degree of mitigation required. Events with higher confidence could receive more severe mitigation. Events with very low confidence could receive limited mitigation. Specific algorithms are developed in the following section.

7.2.1 FBS-DD: A Fair Bandwidth Sharing and Delay Differentiation Mechanism

In this section we discuss proposed algorithms for packet forwarding with buffer management Fair Bandwidth Sharing and Delay Differentiation (FBS-DD). We consider a lossless and work-conserving
packet scheduler that serves \( N \) queues, one for each class. The FBS-DD model is to maintain the multi-dimensional QoS spacing of two classes with respect to both their delay ratio \( \frac{D_i}{D_j} \) and bandwidth sharing ratio \( \frac{B_i}{B_j} \) be proportional to their pre-specified differentiation parameters \( \delta_i \) and \( \delta_j \). That is,

\[
\frac{D_i(T, T+t)}{D_j(T, T+t)} \cdot \frac{B_j(T, T+t)}{B_i(T, T+t)} = \frac{\delta_i}{\delta_j}, 1 \leq i, j \leq N \tag{7-1}
\]

for time intervals \((T, T+t)\) where \( t \) is the monitoring timescale. Note that the lower average delay or higher bandwidth sharing represents higher QoS. For implementation as part of a scalable automated response capability the learning engine combined with the suspected attack class and confidence level will establish QoS categories for network defense.

### 7.2.1.1 VPS-TWP: Throughput normalized waiting time priority scheduling

First, we consider the general case, that is, packets from a class have various sizes and different classes may have different packet size distributions. We revisit the time dependent priority scheduling discipline and propose the VPS-TWP scheme, which focuses on the instantaneous behavior. The time dependent priority scheduling was first studied in queuing foundations. It was later studied in WTP [36] for PDD provisioning. We describe VPS-TWP as the throughput normalized waiting time priority scheduling algorithm for FBS-DD provisioning.

At the beginning of scheduling, \( TWP_i = \infty \) for \( 1 \leq i \leq N \). Suppose that class \( i \) is backlogged at time \( t \), \( s_i(t) \) is the size of the packet at the head of the class \( i \) at \( t \), and that \( w_i(t) \) is the head waiting time of class \( i \) at \( t \), i.e., the waiting time of the packet at the head of the class \( i \) at \( t \). We define the throughput normalized head waiting time of class \( i \) at \( t \) as

\[
TWP_i(t) = \frac{w_i(t)}{\delta_i s_i(t)} \tag{7-2}
\]

Every time a packet is to be transmitted, the VPS-TWP scheduler selects the backlogged class \( j \) with the maximum throughput normalized head waiting time,

\[
j = \arg \max_{i \in \mathcal{G}(t)} TWP_i(t) \tag{7-3}
\]
where \( G(t) \) is the set of backlogged classes at time \( t \). Tie breaks by the use of priority. The throughput of class \( j \) is increased by the size of the transmitted packet. Its throughput normalized head waiting time will be minimized as its packet delay will not increase any more. VPS-TWP attempts to minimize the differences between the bandwidth normalized waiting times of successively departing packets. It essentially aims to achieve instantaneous FBS-DD.

Next, we consider a special case, that is, all packets have the uniform size. The experienced bandwidth ratio of classes \( i \) and \( j \) is given as \( B_i(T, T + t) / B_j(T, T + t) = b_i / b_j \) where \( b_i \) and \( b_j \) are the number of packets departed in the interval \( (T, T+t) \), essentially the throughput of the classes in the interval. When all packets have the uniform packet size, VPS-TWP is reduced to UPS-TWP. The throughput normalized head waiting time of class \( i \) at \( t \) is calculated as

\[
TWP_i(t) = \frac{w_i(t)}{b_i \delta_i} \quad (7.4)
\]

### 7.2.1.2 VPS-TAD: Throughput normalized average delay scheduling

While VPS-TWP focuses at the instantaneous behavior, we propose the VPS-TAD scheme which focuses on the long-term behavior. (3) can be rewritten as

\[
\frac{D_i(T, T + t)}{\delta_i B_i(T, T + t)} = \frac{D_j(T, T + t)}{\delta_j B_j(T, T + t)} \quad (7.5)
\]

The way to interpret it is that the throughput normalized average delay (TAD) must be equal in all classes. That is \( TAD_i = TAD_j \). Note that given a same interval, bandwidth sharing ratio of two classes is the same as the throughput ratio. The VPS-TAD scheme, tailored from PAD [36], aims to equalize the throughput normalized average delays among all classes so as to achieve the FBS-DD goal.

Let \( G(t) \) is the set of backlogged classes at time \( t \), \( L_i(t) \) be the sequence of class \( i \) packets that were transmitted during the interval \( (T, T+t) \), \( d_{im}^n \) be the delay of the \( m \)th packet in \( L_i(t) \), and \( s_{im}^n \) be the size of the \( m \)th packet in \( L_i(t) \). Assuming that there was at least one packet transmitted from class \( i \) during interval \( (T, T+t) \), the throughput normalized average delay of class \( i \) at \( t \) is
\[
TAD_i(t) = \frac{D_i(T, T = t)}{\delta_i B_i(T, T + t)} = \frac{1}{\delta_i \sum_{m=1}^{\mid L_i(t) \mid} s_i^m} \frac{\sum_{m=1}^{\mid L_i(t) \mid} d_i^m}{\mid L_i(t) \mid} \tag{7-6}
\]

where \( \mid L_i(t) \mid \) is the number of packets in \( L_i(t) \).

At the beginning of scheduling, \( TAD_i = \infty \) for \( 1 \leq i \leq N \). Suppose that a packet is to be transmitted at time \( t \). VPS-TAD selects the backlogged class \( j \) with the maximum bandwidth normalized average delay,

\[
j = \arg \max_{i \in G(t)} TAD_i(t) \tag{7-7}
\]

Tie is broken by the priority. The rationale of VPS-TAD is that each time a packet from class \( j \) is transmitted, its throughput normalized average delay decreases. This is because its throughput increases by the size of the transmitted packet. The delay of that transmitted packet will not increases any more, and thus the increase to the average packet delay will be minimized. VPS-TAD therefore attempts to minimize the differences between the throughput normalized average delay of classes. It essentially aims to achieve FBS-DD in the long term. It, however, needs to maintain the state information about the current throughput and average delay per each class.

When all packets have the uniform size, VPS-TAD is reduced to UPS-TAD. The throughput normalized average delay of class \( i \) is calculated as

\[
TAD_i(t) = \frac{D_i(T, T + t)}{\delta_i B_i(T, T + t)} = \frac{1}{b_i \delta_i} \frac{\sum_{m=1}^{\mid L_i(t) \mid} d_i^m}{\mid L_i(t) \mid} \tag{7-8}
\]

### 7.2.1.3 PID Control-theoretic Buffer Management

When the overall workload is greater than the link capacity, packet loss is inevitable and loss rate becomes the dominant QoS metric. The proposed packet scheduling schemes, however, have no control over the loss rate differentiation between classes. We propose a control-theoretic buffer management scheme, to be integrated with the packet scheduling schemes, for the FBS-DD provisioning and proportional loss rate differentiation at the same time. One nice feature of the buffer management based approach is that the
packets will be dropped from the tail due to the buffer overflow. This avoids the packet pushout issue and facilitates the packet ordering.

The buffer management scheme is to dynamically allocate the buffer space into a number of virtual mini-buffers, one mini-buffer for one class. The size of a mini-buffer directly affects a class’s loss rate. Feedback control theory has been applied to adjust the resource allocation for service differentiation provisioning [85, 94]. We propose to use a proportional integral derivative (PID) controller to adjust the buffer allocation. Let \( l_i \) be the loss rate of class \( i \). The goal is to ensure that the observed relative loss rate \( l_i \) be proportional to the pre-specified QoS parameter \( \delta_i \), that is, \( l_i/l_j = \delta_i/\delta_j \). Let \( L_i \) be the relative loss rate ratio of class \( i \), that is, \( L_i = \frac{l_i}{l_1+l_2+\cdots+l_n} \). Let \( L_i^d \) be the desired relative loss rate ratio of class \( i \), that is, \( L_i^d = \frac{\delta_i}{\delta_1+\delta_2+\cdots+\delta_n} \). During the \( k \)th sampling period, the relative error is calculated as difference between the desired value and the observed value, that is,

\[
e_i(k) = L_i^d(k) - L_i(k)
\]  

One property of the model is the sum of the relative errors is always zero since

\[
\sum_{i=1}^{n} e_i(k) = \sum_{i=1}^{n} \left( L_i^d(k) - L_i(k) \right) = 0
\]  

This important property makes it feasible for us to adaptively adjust the buffer allocation for a class independent of the adjustments of other classes while maintaining a constant overall buffer size.

The buffer size allocated to a class is adjusted in proportion to the error between the desired relative loss rate ratio and the observed one. Specifically, the operation of the PID controller is described as follows:

\[
s_i(k+1) = s_i(0) + G_pe_i(k) + G_i \sum_{j=0}^{k-1} e_i(j) + G_D \Delta e_i(k)
\]  

(7-11)
$s_i(k + 1)$ denotes the buffer size allocated to class $i$ in the new sampling period. $s_i(0)$ denotes the initial buffer size allocated. The three terms added to $s_i(0)$ denote proportional, integral, and derivative components, respectively. Setting a large proportional feedback gain ($G_P$) typically leads to faster response at the cost of increasing system instability. The integral controller ($G_I$) can eliminate the steady-state error and avoid over-reactions to measurement noises. The derivative control ($G_D$) considers the change of errors in adjusting the buffer size allocation and hence responds fast to errors. The derivative error with class $i$ is calculated as $\Delta e_i(k) = e_i(k) - e_i(k - 1)$.

7.3 Performance Evaluation

7.3.1 Experimental Setup

We developed a simulator to study the performance of the packet scheduling schemes and the feedback control based buffer management. For the packet size distribution of two classes, we used two Bell Labs-I trace files adopted from the National Laboratory for Applied Network Research\textsuperscript{3}. Without loss of generality, let Class-1 be the high priority class and Class-2 be the low priority class.

7.3.1.1 Packet Scheduling Algorithms

The first set of experiments is to study the impact of the packet scheduling schemes on FBS-DD provisioning when the overall workload is within the link capacity. We considered a lossless model. Figure 13 shows the performance of the packet scheduling schemes. The differentiation weight ratio of two classes ($\delta_1 : \delta_2$) is set to 1:2 and their workload ratio is set to 1:3. Figure 7-2(a) shows the achieved FBS-DD ratio with its 95th and 5th percentiles when the overall workload changes from 55% to 100% of the link capacity. Figure 7-2 (b) shows the achieved delay ratio with its 95th and 5th percentiles.

\textsuperscript{3} NLANR. PMA: Special traces archive. http://pma.nlanr.net/Special/.
The results show that the scheduling schemes can achieve the goal of providing fair bandwidth sharing with delay differentiation when the overall workload is greater than 60%. But the variance, as demonstrated by the 95th and 5th percentiles, is a nontrivial issue. It is due to the variance of the packet size distributions and the inter-arrivals. When the workload is light, there is a feasibility issue with the packet scheduling for service differentiation provisioning [36].

Figure 7-2: The Performance of VPS when overall workload changes from 55% to 100%.

We next change the ratio of $\delta_1 : \delta_2$ to 1:3. We fix the overall workload to 80% and vary the Class-1’s workload from 10% to 90% of the overall workload. Figure 7-3(a) shows the achieved FBS-DD ratio with its 95th and 5th percentiles. It shows the FBS-DD ratio can be achieved as expected. But the variance is high when Class-1’s workload deviates from the middle value 50%. This is due to the fact that there are too few or too many packets from Class-1, limiting the capability of the packet scheduling schemes. Figure 7-3 (b) shows the achieved delay ratio with its 95th and 5th percentiles. We can see that the proposed VPS
scheduling schemes can achieve the fair bandwidth sharing with delay differentiation when the workload percentage of the classes changes dynamically. When the Class-1 contributes 75% of the overall workload, the delay ratio of two classes becomes 1. This demonstrates the benefit of the FBS-DD model that can make adaptive tradeoff between delay differentiation and fair bandwidth sharing. While both VPS-TAD and VPS-TWP schemes can achieve FBS-DD provisioning from the long-term perspective, they have different behaviors.

Figure 7-4: The behaviors of VPS scheduling schemes for FBS-DD provisioning in different sampling intervals

Interestingly, in short sampling intervals, VPS-TAD does not perform well for FBS-DD provisioning. Figure 7-4 (a) shows that its performance improves as the sampling interval increases. This is explained by the fact that VPS-TAD takes into account the average of a number of packets in the interval. It aims to minimize the differences between the normalized average class delays and thus its performance improves as the sampling interval increases. On the other hand, Figure 7-4 (b) shows that VPS-TWP achieves desirable FBS-DD ratios when the sampling interval is short and the performance deteriorates as the interval increases. This is due to the fact that VPS-TWP attempts to minimize the differences between the normalized head waiting times. Essentially, it aims to achieve the instantaneous FBS-DD provisioning.

7.3.1.2 Performance of PID Control Buffer Management

Previous experimental results have shown that the packet scheduling schemes can achieve the fair bandwidth sharing and delay differentiation at the same time. But when the overall workload is beyond the
link capacity, there will be packet loss and the packet scheduling schemes have no control over the loss rate differentiation between classes. Figure 16 depicts the impact of the PID control-theoretic buffer management on loss rate differentiation and the controllability of FBSDD provisioning. The overall workload is set to 150% of the link capacity. The differentiation weight ratio of two classes ($\delta_1 : \delta_2$) is set to 1 : 3. Figure 7-5(a) shows the impact of the PID control-theoretic buffer management on the proportional loss rate differentiation. It shows that with the buffer management, the loss rate ratio of two classes is fairly proportional to the differentiation weight ratio as the percentage of the Class-1’s workload changes from 10% to 90% of the overall workload. On the other hand, without the buffer management, both classes experience almost the same loss rate.

Figure 7-5: The impact of the control-theoretic buffer management on FBS-DD provisioning

Figure 7-5 (b) shows the achieved FBS-DD ratio by the use of VPS-TAD packet scheduling scheme with and without the PID controller based buffer management. The results show that with the PID controller based buffer management, the VPS-TAD scheme is able to achieve more consistent and desirable FBS-DD ratios with respect to both the mean and the variance. Without the feedback control based buffer management, the variance of the FBS-DD ratio is higher. One reason is that if there is no feedback based buffer management, a class with some bursty traffic can saturate the buffer easily, leaving little buffer space for another class. The VPS-TAD scheduling schemes aims to minimize the normalized average delays. Its capability is limited by the availability of packets from certain classes for scheduling. This benefits the low-priority but high-workload class. Therefore, the buffer management should be integrated
with packet scheduling for controllable FBS-DD provisioning. The integrated approach is capable of self-adapting to varying workloads from different classes, which automatically builds a firewall around aggressive clients and hence protects network resources from saturation.

7.4 Summary and Discussion

In this chapter we proposed algorithms that can be used for automated response to anomaly events. In Chapter 5 we proposed a method for correlating individual alerts. In Chapters 5 and 6 we proposed a method for determining prediction confidence and then using the calculation to select a small subset of records for additional processing. In this chapter we proposed a model and algorithms that can be used to provide QoS based scalable response.

We develop a mechanism for Fair Bandwidth Sharing and Delay Differentiation. We develop two packet scheduling algorithms, throughput normalized waiting time priority scheduling, and throughput normalized average delay scheduling. We also propose a PID Control buffer management technique to provide proportional loss. We demonstrate that our proposed methods are able to achieve predicted results.

We introduce the idea that these algorithms could be combined with alert correlation and prediction confidence filtering to provide scalable response. However, we do not directly propose specific methods for providing this link. Our future work will expand on these ideas and develop this mechanism.
Chapter 8.

Conclusion

8.1 Summary and Discussion

Anomaly detection is a challenging problem that has been researched within a variety of application domains. In network intrusion detection, anomaly based techniques are particularly attractive because of their ability to identify previously unknown attacks without the need to be programmed with the specific signatures of every possible attack. There is a significant body of work in anomaly based intrusion detection applying statistical analysis, data-mining, and machine learning disciplines. However despite more than two decades of active research, there is a striking lack of anomaly based systems in commercial use today. Many of the currently proposed anomaly based systems do not adequately address a series of challenges making them unsuitable for operational deployment. In existing approaches, every step of the anomaly detection process requires expert manual intervention. This dependence makes developing practical systems extremely challenging.

In this thesis, we integrate the strengths of machine learning and quality-of-service mitigation techniques for network anomaly detection, and build an operationally practical framework for anomaly-based network intrusion detection. We propose methods for self-adaptive, self-tuning, self-optimizing, and automatically responsive network anomaly detection.

We proposed an efficient method for preprocessing network traffic generating augmented flow records. Nearly all network intrusion detection systems require some form of pre-processing data. Unless the system is going to perform its prediction by inspecting each packet in its entirety, there is some form of pre-processing or data presentation that must be performed. In an attempt to find known attacks or unusual behavior, modern intrusion detection systems traditionally inspect the contents (payload) of every packet. The problem of packet inspection, however, is that it is hard, or even impossible, to perform it at the speed of multiple Gigabits per second (Gbps). Additionally, Opaque traffic that is compressed or encrypted may make packet content inspection impossible. and accomplishing dynamic input normalization. We develop
a method of augmenting basic flow records with an array of additional information that directly provides volume metrics, but also allows detectors to infer information about distribution changes in specific fields. Each generated flow is identified by the 5-tuple \{src_addr, src_port, dst_addr, dst_port, protocol\}. Each flow record also contains metrics specific to that flow (duration, packet count, byte count, etc…). When a flow record is generated, we use statistical sketches to augment that record with estimated flow, packet, and byte information from all other related flows previously observed. We demonstrate experimentally that these augmented flows provide discriminable information that detectors can use to make predictions.

We developed an adaptive input normalization approach that will bring attention to the true difference between individual input vectors. In many approaches, data is normalized to account for differences in scale between two different data points. We propose an adaptive input normalization approach that automatically tunes scaling parameters online. We incorporated dynamic input normalization into two detection adaptive detection engines we developed and experimentally demonstrate the ability of this method to significantly increase detection.

We developed two adaptive detection engines. First we develop an Adaptive Growing Hierarchical Self Organizing Map. In a GHSOM, the size and dimensionality of the map architecture are determined during the training phase. The map is grown horizontally by adding rows and columns to the map to reduce quantization error. The map is grown vertically by adding child layers to parent layers that exceed the pre specified maximum dimensionality of each map. We developed an enhanced A-GHSOM that is able to adapt its architecture on-line as events are processed. Additionally, it uses adaptive thresholds to classify dis-similar events mapped to similar portions of the model. We also propose an Adaptive Support Vector Machine. Support vector machines use hyper-planes to divide samples into one of two classes. We proposed an SVM model that maintains a dynamic on-line training set and uses the SMO algorithm to efficiently adapt itself during live operation. We developed and tested algorithms for heuristically adding and removing training samples reducing the number of on-line passes the training algorithm must make to adapt. In both these models we developed novel approaches for determining prediction confidence.

We developed an evolving alert correlation engine. Many anomaly detectors generate alerts on individual connections or flows. However, network wide anomalies are rarely isolated to a single flow. Individually identifying anomalous flows does not provide operators with accurate threat assessments. On
the other hand, many anomaly detectors capable of identifying network wide events fail to identify the specific flows composing the anomalous event. In this thesis we developed and tested a novel evolving alert aggregation and correlation approach that generates correlated alerts used to guide initial response actions. We use efficient linear-counting distinct value estimation techniques to dynamically aggregate alerts on individual connections giving operators and downstream response engines an evolving picture of the current threat environment. We experimentally demonstrate our methods ability to not only increase anomaly prediction capability enhancing recall, but also helping focus model feedback on false positive responses and thereby also increasing precision.

We developed and tested a reinforcement learning approach to automated tuning and optimization in anomaly detection systems. We propose an optimization scheme that incorporates prediction confidence with precision and recall metrics. Our approach allows an operator to set performance goals and priorities in any two metrics and a reinforcement learning based controller attempts to meet these goals while simultaneously optimizing performance in the third. We integrate neural network function approximation methods into our state space calculation. We propose an action-space discretization / aggregation method making our controller suitable for use in anomaly detection systems.

We perform extensive tests on the KDD and MAWI datasets and we also develop a new dataset based on a combination of live trace data and simulated traffic. Extensive tests demonstrate that our augmented flow records provide discriminable information that detectors can use to make predictions. Our detection engines are able to make accurate predictions and also adapt on-line. Our alert correlation approach aggregates individual alerts into network wide events enhancing performance capability. Our dynamic controller is able to approach operator defined constraints and our quality of service algorithms are able to provide proportional methods that could be used to provide initial response. By using multiple data sets we demonstrate that our methods are capable of operating in numerous environments. By developing several detection engines we demonstrate that our approaches are model independent.

Compared to eight different approaches published from 2002 to 2012 using the KDD data, our methods outperform them using the KDD dataset in both detection capability and false positive rates. In Table 8-1 we show that detection engines approaching our detection capability only able to do so with substantially higher false positive rates.
### Table 8-1: Summary of performance comparisons on KDD dataset

<table>
<thead>
<tr>
<th>Approach</th>
<th>Recall</th>
<th>False Positive</th>
</tr>
</thead>
<tbody>
<tr>
<td>A-SVM (Chapter 5)</td>
<td>98.09%</td>
<td>0.64%</td>
</tr>
<tr>
<td>A-GHSOM (Chapter 5)</td>
<td>99.63%</td>
<td>1.8%</td>
</tr>
<tr>
<td>GHSOM [Palomo et al 2008]</td>
<td>90.87%</td>
<td>2.69%</td>
</tr>
<tr>
<td>Sivitha et al 2012</td>
<td>98.38%</td>
<td>4.2%</td>
</tr>
<tr>
<td>Bouzida et al 2004</td>
<td>91.89%</td>
<td>0.48%</td>
</tr>
<tr>
<td>Sarasamma et al 2005</td>
<td>93.46%</td>
<td>3.99%</td>
</tr>
<tr>
<td>Kayacik et al 2007</td>
<td>90.4%</td>
<td>1.38%</td>
</tr>
<tr>
<td>Eskin et al 2002 (Data Mining)</td>
<td>90.00%</td>
<td>2.0%</td>
</tr>
<tr>
<td>Eskin et al 2002 (Clustering)</td>
<td>93.00%</td>
<td>10.0%</td>
</tr>
<tr>
<td>Eskin et al 2002 (SVM)</td>
<td>98.00%</td>
<td>10.0%</td>
</tr>
<tr>
<td>Yu et al 2008 (Adaptive)</td>
<td>96.02%</td>
<td>10.0%</td>
</tr>
</tbody>
</table>

Compared to four different models published from 2007 through 2011 using the MAWI dataset, our methods are able to significantly outperform them in both Precision and Recall. Two of the models achieve slightly higher Precision rates than our method. However, they have Recall rates of ~55% and ~70% compared to our methods achieving 95% Precision and 99% Recall.

### 8.2 Future Work

Cyber Defense and in specific network anomaly detection is a hot-bed topic. In this thesis we proposed a framework for automated anomaly detection. Many of the methods we proposed rely on operator interaction. However, we do not explore in depth methods for providing that interaction. An interesting extension to this work would be to examine methods for detection engines to receive feedback both direct and indirect. Methods for operators to directly interact with underlying detection engines without requiring knowledge of specific algorithms is an open topic. Methods have been proposed that use challenges, direct correction, and sampled feedback. Additionally, indirect feedback garnered by monitoring operator behavior and adapting based on observations rather than direct corrective interaction could be used to
enhance this work. Incorporating network health, and topology changes, could also add additional capability when interpreting feedback.

We also propose the idea of scaled response based on confidence. We develop algorithms that can be used for this purpose but do not directly define methods for implementing this strategy. A logical extension is to further develop automatic scaled response actions based on the ideas proposed in this dissertation.

8.3 Publications

The work accomplished while pursuing this dissertation has generated several peer-reviewed publications listed below.

In IEEE Peer-reviewed Conference Proceedings


3. **Packet Scheduling with Buffer Management for Fair Bandwidth Sharing and Delay Differentiation**, Dennis Ippoliti, Xiaobo Zhou, and Liqiang Zhang, Proc. of the 16th IEEE International Conference on Computer Communications and Networks (ICCCN), IEEE Communications Society, pages 569-574, Honolulu, August 2007. (acceptance rate 29%)

In Peer-reviewed Journals


Submitted for publication in

Paper Submitted For Publication

1. Online Adaptive Anomaly Detection for Augmented Network Flows

Dennis Ippoliti, Xiaobo Zhou, Submitted for review to International Conference on Dependable Systems and Networks 2014
Bibliography


[68] Z. Ji, B.L. Lu A Support Vector Machine Classifier with Automatic Confidence and its Application to Gender Classification In Neurocomputing 74. 1926-1935 (2011)


[105] NLANR. PMA: Special traces archive. [http://pma.nlanr.net/Special/](http://pma.nlanr.net/Special/).


South African Institute of Computer Scientists and Information Technologists on IT Research in Developing Countries, Stellenbosch, Western Cape, South Africa, 2004, pp. 221–228.


[165] WINPCAP Application Library http://www.winpcap.org


