This is to certify that the
thesis entitled

USING DEVELOPMENTAL LEARNING FOR
NETWORK INTRUSION DETECTION

presented by

DAVID B. KNOESTER

has been accepted towards fulfillment
of the requirements for the

M.S. degree in COMPUTER SCIENCE

Major Professor's Signature

5/14/04

Date

USING DEVELOPMENTAL LEARNING FOR NETWORK INTRUSION DETECTION

By

DAVID B. KNOESTER

A THESIS

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

MASTER OF SCIENCE

Department of Computer Science

2004

A key
Variability
eral techn
ing proble
of machin
network i
to leverag
ral foun
describes
problem
demonst

ABSTRACT

USING DEVELOPMENTAL LEARNING FOR NETWORK INTRUSION DETECTION

By

DAVID B. KNOESTER

A key problem in the field of network security is network intrusion detection. Variability in network attacks, frequent software vulnerabilities, and the lack of general techniques for security combine to make network intrusion detection a challenging problem. This thesis describes the application of developmental learning (a form of machine learning) to network intrusion detection. An extensible framework for network intrusion detection has been developed. The framework is general enough to leverage existing intrusion detection methods, while providing a solid mathematical foundation for identifying network attacks in packet streams. This thesis also describes difficulties encountered when casting a fundamentally systems-oriented problem into the machine-learning domain. Experimental results are presented that demonstrate the effectiveness of this approach for misuse and anomaly detection.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Ensuring an individual's right to privacy, protecting national infrastructure and preventing costly service outages all require network security. Although some network intrusions are as innocuous as undesired email, others disrupt global network services and cause billions of dollars in lost productivity [15]. The variability of network intrusions coupled with frequent software vulnerabilities and the lack of general techniques for securing networks has made network intrusion detection a challenging problem.

Network intrusions, also called network attacks, are activities that lead to the unauthorized use of computer resources. They include attempts to compromise integrity, confidentiality, or availability of network services [6]. Most network intrusions have a distinct *attack signature*, comprised of unique and observable characteristics, that allow Network Intrusion Detection Systems (NIDS) [9, 11, 18, 19, 25, 27–29, 31, 41, 42, 44, 47, 48] to positively identify them. However, slight variations in attack signature are often all that is required to foil detection. Furthermore, there are many different styles of network intrusions. Some target specific applications or operating systems, while others engage in Distributed Denial of Service (DDoS) attacks [26]. Finally, new network attacks, especially those that exploit a recently disclose

signatur

Net

intrusio

tures. S

tions. E

detector

on thes

often c

ble of d

rate [8,

Eff

the wea

sis, we

form of

measur

packets

tween p

difficult

Thesis

w

d

fe

Th

1. M

r

disclosed or unpublicized software vulnerability, often do not have a known attack signature.

Network attack variability has led to two approaches for detecting network intrusions. First, *misuse detection* searches network traffic for known attack signatures. Second, *anomaly detection* recognizes unusual or anomalous network conditions. Each approach has different strengths and weaknesses. For example, misuse detectors can recognize known attacks, but are unable to reliably detect variations on these attacks. Misuse detectors are also unable to detect previously unknown (often called zero-day) attacks [20, 21]. Conversely, anomaly detectors are capable of detecting zero-day attacks, but generally do so with a high false-positive rate [8, 20, 21, 23].

Effective network intrusion detection requires techniques that can help address the weaknesses of current approaches to misuse and anomaly detection. In this thesis, we explore the application of methods based on developmental learning [50], a form of machine learning, to network intrusion detection. By deriving *features*, or measurements suitable for use by a developmental learning algorithm, from network packets, we are able to mathematically determine the similarity (or lack thereof) between packet streams. We address both misuse and anomaly detection, and describe difficulties related to deriving features from network packets.

Thesis Statement: By applying methods based on developmental learning to network intrusion detection, it is possible to perform both misuse and anomaly detection within the same framework, providing good detection accuracy with few false-positive alarms.

The major contributions of this work can be summarized as follows:

1. *MESOTree-based Network Intrusion Detection System (MESONet) design and implementation.* MESONet is both a framework for extracting features from

m

te

to

M

be

go

di

an

2. A

w

le

ph

ap

pr

3. P

n

19

ME

ments

system.

matter.

for tran

a netw

A

MESON

[1] We a
in detai

network traffic, and an implementation of a network intrusion detection system. MESONet uses MESO-Tree [24], a classification algorithm similar to Hierarchical Discriminant Regression [22, 49], for developmental learning. MESONet is independent of the particular learning algorithm used and can be easily redeployed atop a different system, enabling the use of improved algorithms and implementations without extensive software modification. Additionally, MESONet supports a wide variety of runtime selectable features, and is easily extended to incorporate new features.

2. *Application of developmental learning to a systems-oriented problem.* Network intrusion detection systems have become increasingly complex. Machine learning techniques offer an approach that is capable of dealing with this complexity. This thesis shows that machine learning techniques can be successfully applied in a systems domain, specifically network intrusion detection, and we present difficulties encountered when doing so.

3. *Performance evaluation through experimentation.* We evaluate the performance of MESONet for misuse and anomaly detection using a subset of the 1999 DARPA IDS Evaluation datasets[1] [33].

MESONet is a platform for conducting developmental-learning based experiments into network intrusion detection. Because MESONet is a learning-based system, it must be trained on known data prior to conducting tests. As a practical matter, both training and testing data must be known; that is, every packet used for training or testing must be labeled either *normal*, meaning that it is not part of a network attack, or *attack*, predictably meaning that it is part of a network attack.

Although all experiments presented in this thesis are the results of offline tests, MESONet is capable of operating online, sniffing traffic directly from a network.

[1] We are aware of research critical of the 1999 DARPA IDS Evaluation. This issue is addressed in detail in Section 2.3.

For this reaso
files. MESON
open source J
libpcap [3], a

Experim
that for each
once. MESON
trained on o
incremental t
network.

Normal
ing execution
packets direc
attack, if pos
belong. Nex
labeled pack
are used as t
testing, each
most similar
the test resul
detection mo

The rem
ground infor
scribes the 1
Chapter 3 des
velopmental
misuse detect

For this reason, and to ensure compatibility with publicly available network tracefiles, MESONet also incorporates packet capturing, or sniffing. MESONet uses the open source JPCap package [12] for network traffic capturing, which is built upon libpcap [3], a cross-platform packet capturing facility.

Experiments presented in this thesis are the results of batch training, meaning that for each experiment, MESONet was trained on all available training data at once. MESONet is capable of incremental training, for example, MESONet can be trained on one network packet at a time. Combined with its online capabilities, incremental training enables near "plug-and-play" installation of MESONet on any network.

Normal operation of MESONet follows the sequence shown in Figure 1.1. During execution, JPCap reads packets from a previously captured tracefile (or captures packets directly from the network). Each packet is then labeled as either normal or attack; if possible, attack packets are labeled with the specific attack to which they belong. Next, an array of features, called a *feature vector*, is extracted from each labeled packet. During training, these feature vectors and their associated labels are used as training data by the developmental learning algorithm (DLA). During testing, each feature vector is used as input to the DLA, and the DLA returns the most similar instance of training data. Finally, we perform different analyses on the test results depending on whether MESONet is operating in misuse or anomaly detection mode.

The remainder of this thesis is organized as follows. Chapter 2 provides background information on developmental learning and the MESO-tree algorithm, describes the 1999 DARPA IDS Evaluation, and presents a summary of related work. Chapter 3 describes features in MESONet and the difficulties associated with the developmental learning approach to network intrusion detection. Chapter 4 describes misuse detection with MESONet, including training and testing procedures, performance analys

detection wit

analyses, and

work and co

mance analyses, and experiments performed. Similarly, Chapter 5 describes anomaly detection with MESONet, including training and testing procedures, performance analyses, and experiments performed. Finally, Chapter 6 describes possible future work and concludes.

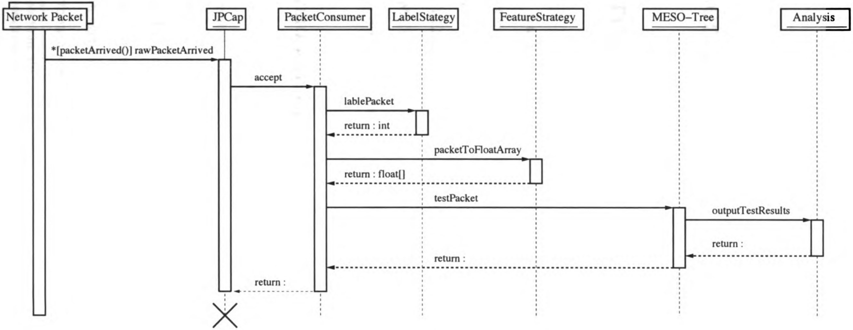


Figure 1.1: Sequence Diagram for MESONet.

Cha

Bac

MES

tection a

if any, ov

derstand

describes

Evaluatio

2.1

Deve

based on

opmenta

its enviro

mental le

For exam

evolved

[1]An age
acting upo

Chapter 2

Background

MESONet draws upon previous research from both the network intrusion detection and developmental learning communities. As these communities have little, if any, overlap, this chapter provides background information sufficient to the understanding of MESONet. Section 2.1 describes developmental learning, Section 2.2 describes the MESO-tree algorithm, Section 2.3 describes the 1999 DARPA IDS Evaluation, and Section 2.4 describes related work.

2.1 Developmental Learning

Developmental learning is a recently established machine learning paradigm based on *Automated, Animal-like Learning*, or AA-learning [50]. The goal of developmental learning is to enable an agent[1] to learn online through interaction with its environment. In contrast to task-specific machine learning approaches, developmental learning focuses on general purpose, non-task-specific autonomous agents. For example, an agent implemented using evolution-based machine learning may be evolved to perform a specific task [46], while a developmental agent is capable of

[1] An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors [43].

dynamically learning new tasks, without need for reprogramming.

A *developmental learning algorithm* (DLA) is responsible for building and maintaining a representation of what the agent has been taught. Teaching a developmental agent requires that the agent: 1) observe relevant information about its environment, and 2) is provided the correct action to perform given its observations.

Environmental observations are collected via *sensors*, which produce scalar measurements of particular environmental *features*. For example, a thermometer is a sensor that produces a measurement of the temperature feature. Features may also be calculated from sensor readings. For example, interpacket delay may be calculated using the arrival times of sequentially sensed network packets. Both sensed and calculated features are collected as an array called a *feature vector*, or *sample*, that represents all of an agent's observations at a given point in time.

Actions, or *labels* [14], are typically communicated to the developmental agent by an external entity. In some cases a human may perform this role, interactively providing feedback to the agent based on the agent's performance [10]. In other cases, software may provide the agent with a label based on specialized knowledge. For example, a database of network packets could provide labels ("attack" or "normal") to an agent based on whether or not a particular packet is present in the database. Each label is associated with a feature vector, representing the desired action an agent is to perform when observing environmental conditions similar to those contained in the feature vector.

As shown in Figure 2.1, our DLA implementation produces a statistical model using techniques based on pattern classification and agglomerative clustering [14]. This statistical model is a *classifier* [24], and its task is to determine the label for unlabeled feature vectors. Typically, DLAs have two distinct phases of operation: *training* and *testing*. During training, the DLA builds its classifier from a set of labele

featur

label

2.2

Org

labeled feature vectors. Once trained, the DLA can be queried using an unlabeled feature vector. The DLA then tests this vector against the classifier, and returns a label indicating the action that is most likely desired of the agent.

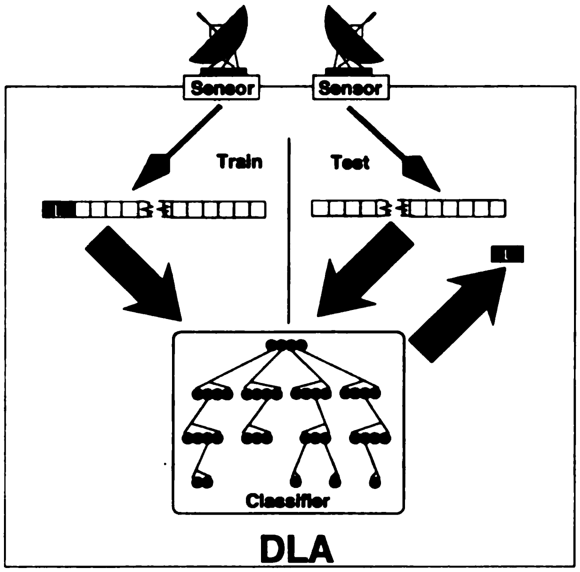


Figure 2.1: High level view of a classifier-based DLA.

2.2 MESO-Tree Algorithm

The classifier used in MESONet is called MESO-tree, for Multi-Element Self-Organizing Tree [24]. MESO-tree uses small agglomerative clusters, called *sensitivity spheres* to group similar training samples together. Each sensitivity sphere acts as a proxy for the samples it contains, which, depending on the training data, may be anywhere from a single sample to tens of thousands of samples. The motivation for utilizing sensitivity spheres rather than operating directly on the training data is to reduce computational requirements, which is especially significant in cases where sensitivity spheres represent many samples.

Sensitivity spheres are partitioned into sets during the construction of a memory-efficient decision tree [37], which produces a hierarchical model of the train-

ing data. Each node of the decision tree comprises a set of sensitivity spheres called a *partition*[2]. Each partition contains a subset of the sensitivity spheres that are most similar to each other. At subsequent tree depths, parent partitions are split according to a branching factor, producing smaller child partitions of greater detail. These child partitions comprise distinct subsets of the parent's sensitivity spheres, and provide finer discrimination and better classification of test samples.

Classifying a sample requires that the sample be compared to the sensitivity spheres within the partitions, and the path of greatest similarity followed. Once a leaf node of the tree is reached, the sample is compared directly to the training data within the most similar sensitivity sphere, and the most similar training sample therein is found. Both the label associated with this training sample and the similarity metric between it and the tested sample are returned. This final similarity metric is termed the *nearest-cluster* distance.

A novel capability of MESO-tree is its independence from a particular measure of similarity. As such, any algorithm that calculates a metric between two feature vectors may be used by MESO-tree to determine similarity between samples. For example, MESONet uses Euclidean distance, but Mahalanobis or Canberra distances could be used as well.

Additionally, MESOTree-based DLAs can support both batch and incremental training. Incremental training enables construction of the classifier's statistical model through the addition of one or only a few samples at a time. As such, the classifier can be trained and tested during concurrent interaction with users or other system components.

By leveraging this structure, a relatively simple distance metric, such as Euclidean distance, can be used to achieve high classification accuracy. One key advantage of this approach is that it requires a small amount of calculation, enabling rapid

[2] As used here, the term *partition* is unrelated to mathematical combinatorics.

classi

featu

T

spe

whi

a gra

corre

repr

two

ver

len

0.5

ave

ap

classifier training and testing, as wells as limiting the impact of high-dimensional feature vectors.

To illustrate, consider a feature vector to represent a single point in the *feature-space*, an N-dimensional space where each dimension corresponds to a single feature, which in the case of MESONet would be related to network packets. Figure 2.2 is a graphical depiction of a three-dimensional feature space, where the dimensions correspond to the packet length, average interpacket delay, and an indicator feature representing whether or not the packet uses the TCP protocol. This figure shows two clusters of feature vectors, labeled A and B, each of which contains feature vectors for 3 packets. The feature vectors in cluster A have an average packet length of approximately 256 bytes, an average interpacket delay of approximately 0.5 seconds, and are all TCP packets. The feature vectors in cluster B have an average packet length of approximately 768 bytes, an average interpacket delay of approximately 2 seconds, and are not TCP packets.

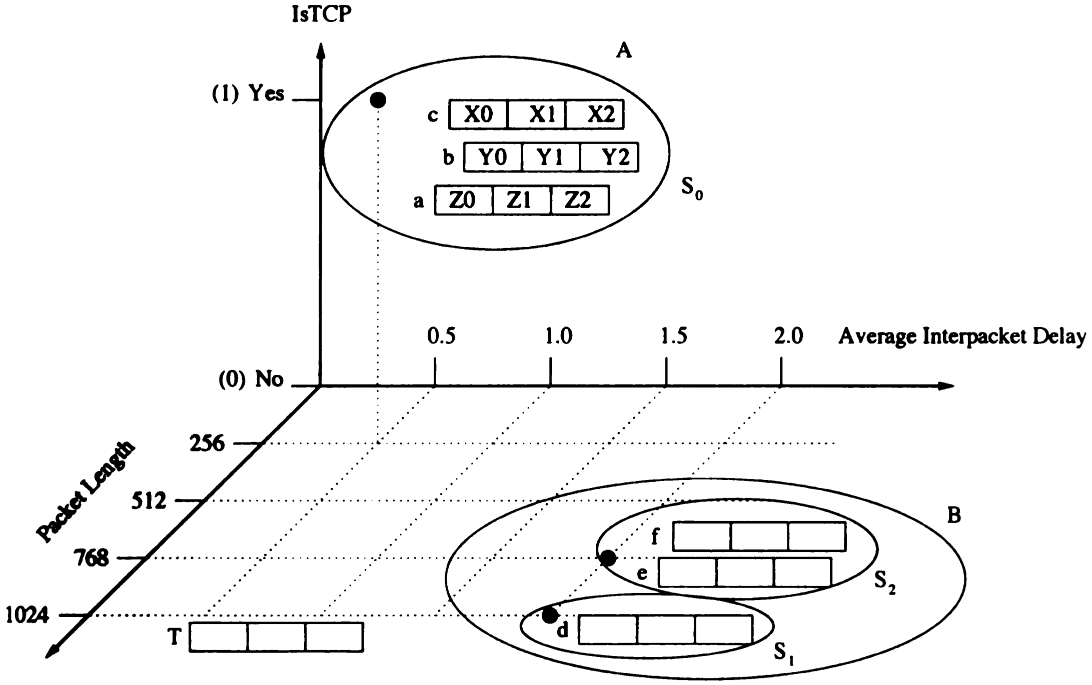


Figure 2.2: Example feature space.

Basec

partition

vectors. S

spheres, w

feature ve

a testing

similar tr

15 (The

2.3

The

mal evalu

Evaluati

as [2, 4, 3

datasets

(e.g. "att

The

selected

systems

all non-

tomata

automa

agers,

Attack

[3] We

contain

to these

Based on the preceding discussion of MESO-tree, we see that cluster A is a partition containing a single sensitivity sphere, which in turn aggregates 3 feature vectors. Similarly, we see that cluster B is a partition containing two sensitivity spheres, which together aggregate 3 feature vectors. Figure 2.2 also contains the feature vector T, which is nearest to the S_1 sensitivity sphere in cluster B. If T were a testing sample, it would be assigned the same label as feature vector *d*, the most-similar training sample, and the nearest-cluster distance would be approximately 1.5 (The Euclidean distance between feature vectors T and *d*).

2.3 1999 DARPA IDS Evaluation

The 1999 DARPA IDS Evaluation (henceforth "IDSEval") was the second formal evaluation of intrusion detection systems, the first being the 1998 DARPA IDS Evaluation. Many different datasets are available that contain network traffic, such as [2,4,38]. However, as stated in [36], the 1998 and 1999 DARPA IDS Evaluation datasets are the only ones to contain an accurate "ground truth," or actual labels (eg, "attack" or "normal") for individual network packets[3].

The IDSEval dataset consists of tracefiles of network traffic, audit logs from selected machines, and a detection scoring truth for evaluating intrusion detection systems [33]. In order to guarantee an accurate ground truth for the IDSEval, all non-attack network traffic was automatically generated by custom software automata developed by the Air Force Research Laboratory (AFRL) [33]. These custom automata were designed to simulate hundreds of programmers, secretaries, managers, and other users common to UNIX and Windows NT application programs. Attacks were performed manually, and each attack was recorded in the detection

[3] We are aware of other corpora, such as [2], which contains entirely unlabeled traffic, [4], which contains sanitized packet traces, sans payload, and [38], which contains solely attack traffic. Due to these factors, these traces are not sufficient for our purposes.

scoring

Net

two poi

suffer w

the outs

network

recorded

to an at

Net

packets

network-

given ne

packets.

We

ysis of b

McHugh

the IDS

as used

upon a p

Section

The

focuses o

cally. Ma

artifact t

1. The

2. The

scoring truth.

Network tracefiles, containing both normal and attack traffic, were captured at two points in the IDSEval network, from "inside" and "outside" sniffers. The inside sniffer was only able to capture network traffic visible on the internal network, while the outside sniffer captured all inbound and outbound network traffic. For each network attack, the attack ID, start time, duration, and victim IP address(es) were recorded in the detection scoring truth as an attack record. All traffic not belonging to an attack record is assumed to be normal traffic.

Network-based misuse detection is then the problem of detecting which of the packets in a given network tracefile belong to specific attack records. Similarly, network-based anomaly detection is the problem of detecting which packets in a given network tracefile are anomalous when compared to a known-normal set of packets, with the hope that these anomalous packets also belong to an attack record.

We are aware of two major critiques of the IDSEval: Mahoney and Chan's analysis of background traffic [34], and McHugh's general critique of the IDSEval [36]. McHugh primarily addresses problems with the setup and evaluation portions of the IDSEval. As a result of his critique, we do away with the *psuedo*-ROC curve as used in the IDSEval, instead using a traditional parametric ROC curve based upon a probabilistic measure. Furthermore, we address the base-rate fallacy [8] in Section 5.2.

The critique of the background traffic in the IDSEval by Mahoney and Chan focuses on possible simulation artifacts appearing in the IDSEval datasets. Specifically, Mahoney and Chan state that three preconditions must exist for a simulation artifact to be present:

1. The attribute is well behaved in simulated traffic without attacks.

2. The attribute is not well behaved in real traffic without attacks.

3.

I

tho

tio

TCP

in re

of th

sour

to b

spec

inclu

wil

occu

cond

pres

of M

the

occu

of th

tra

•

3. The attribute is not well behaved in traffic containing simulated attacks.

Putting aside the fact that their analysis does not prove precondition two (though we agree it is probably correct), we address the specific attributes in question as related to our work in MESONet.

TCP SYN Regularity. In [34], it is noted that the option bytes of different TCP SYN packets have the same values in the IDSEval traffic, and different values in real network traffic. We do not use the presence, or value, of option bytes in any of the features used by MESONet, as will be shown in Section 3.4.

Source Address Predictability. In [34], it is noted that a limited number of source IP addresses are used in the IDSEval data, while real network traffic appears to have a steadily increasing number of source IP addresses. We do not include specific IP addresses in any of the features used by MESONet. We do, however, include the relationship between the source and destination addresses of packets, as will be shown in Section 3.4.

Checksum Errors. In [34], it is noted that TCP and ICMP checksum errors occur in real network traffic, however not in the IDSEval data. MESONet unconditionally filters out all malformed TCP, UDP, and ICMP traffic, therefore the presence (or absence) of these packets has no effect on the operational characteristics of MESONet.

Packet Header Fields. In [34], it is noted that a large variety of values for the TTL and TOS fields occur in real network traffic, while few different values occur for the same fields in the IDSEval data. We do not use TTL or TOS in any of the features used by MESONet, as will be shown in Section 3.4.

"Crud"[sic]. In [34], the following anomalies were noticed in real network traffic and not in the IDSEval data:

- Nonzero TCP ACK fields.

- No

- No

- No

None of

HT

network

deal wi

charact

SY

networ

deal w

charac

numbe

data.

numbe

S

detec

tic ma

chara

2.4

tech

abi

the

- Nonzero URG pointer field.

- Nonzero values in TCP reserved flags.

- Nonzero values in the TCP reserved header bits.

None of these attributes were used in MESONet, as will be shown in Section 3.4.

HTTP Requests. In [34], variations in HTTP requests were noticed in real network traffic, and not in the IDSEval traffic. The particular variations mentioned deal with alphanumeric characters. As will be shown in Section 3.4, alpha-numeric characters are not used in any features appearing in MESONet.

SMTP Requests. In [34], variations in SMTP requests were noticed in real network traffic, and not in the IDSEval traffic. The particular variations mentioned deal with alphanumeric characters. As will be shown in Section 3.4, alphanumeric characters are not used in any features appearing in MESONet. However, a small number of the SMTP requests in real traffic are malformed, and include binary data. This characteristic may have a small effect on MESONet, proportional to the number of symbolic characters appearing in the first 100 bytes of the SMTP request.

SSH Requests. In [34], variations of SSH version identification numbers were detected in real network traffic, and only one in the IDSEval data. This characteristic may have a small impact on MESONet, proportional to the number of symbolic characters appearing in SSH version identification strings.

2.4 Related Work

Intrusion detection systems are based on two main approaches: anomaly detection and misuse detection. Anomaly detection systems are those that detect abnormal behavior [9, 18, 19, 28, 29, 41, 42], while misuse detection systems are those that detect known attacks [9, 11, 27, 31, 41, 47]. Furthermore, some intrusion detection systems are purely network-based [25,41,44,48]. The remainder of this section describes a number of NIDS that use techniques similar, in either approach or implementation, to MESONet.

In [28], Lane and Brodley describe a user-based anomaly detection system. They show that user profiles based on UNIX command sequences are sufficiently unique to distinguish between different users. However, as mentioned in [19], this approach has difficulty with informed intruders and concept drift. Informed intruders are aware of a privileged users habits, for example a proclivity for "vi" as opposed to "emacs," and are able to masquerade as that user. Concept drift occurs when a user's behavior changes enough over time such that the system no longer recognizes malicious behavior, or when a user's normal behavior appears malicious. This research shows that anomaly detection systems must employ machine-learning techniques to avoid problems with concept drift.

In [30], Lee et al. use auditing programs to extract an extensive set of features from network traffic, in this case traffic from the 1998 DARPA IDS Evaluation. They then apply data mining techniques to these features to learn rules that capture the behavior of both normal and attack traffic. In their particular case, they used Bro [41] and NFR [39] for IP packet filtering, packet reassembly, and to generate the audit data. They used RIPPER [13], a classification rule learning program, to mine the audit data. An interesting characteristic of this approach is that determining classification rules is entirely automated via use of RIPPER. However, a shortcoming of this particular approach is the dependency upon audit data; it is unable to operate directly on network traffic.

In [7], Asaka et al. developed a method for intrusion detection based on discriminate analysis. They developed a host-based IDS, designed to detect network intrusions on the local machine. It monitors system logs for occurrences of 11 different system calls performed by privileged network servers, and uses discriminant and

principle component analysis to determine if the behavior is indicative of an intrusion. It is capable of detecting new intrusions and can distinguish between normal and attack traffic. However, it is limited to monitoring a single host, is dependent on the logging capabilities of the host, and is not designed to operate directly on network traffic.

In [45], Chen et al. describe an anomaly detection scheme based on principle component classification. They assume that anomalies can be treated as outliers, and construct a predictive model from the major and minor principle components of normal instances. They conducted experiments with the 1999 KDD Cup data [1], which contains connection records derived from the 1998 DARPA IDS Evaluation [32], rather than raw packet traces. They show that their proposed method outperforms nearest–neighbor, density-based local outliers, and Canberra metric approaches. Although this research was successful when operating on connection records, it was not designed to operate directly on network traffic.

In [35], Mahoney and Chan describe a two-part intrusion detection system, consisting of a packet header anomaly detector (PHAD) and an application layer anomaly detector (ALAD). Both parts are nonstationary, meaning that they model probabilities based on the time since the last event, rather than the average rate of events, which reduces the importance of frequent events. The authors tested the combination of PHAD and ALAD on the 1999 DARPA IDS Evaluation data. Interestingly, a point is made of mentioning that network anomaly detection is "...essentially the machine learning problem of modeling normal network traffic from a training set." To put it another way, anomaly detection tries to learn the characteristics of the normal class and determine if new instances do not belong to that class. This research shows that time-based characteristics of network traffic are significant for intrusion detection.

Chapter 3

Features

As mentioned in Section 2.1, sensors collect environmental observations that produce scalar measurements of environmental features. In MESONet, the only available sensor is the network interface, therefore all features must be derived from network packets. Features are a critical component of MESONet because they are the only link between the classifier and network packets. Furthermore, feature vectors must comprise distinguishing characteristics of network attacks in order for MESONet to be effective at network intrusion detection.

The remainder of this chapter is organized as follows. Section 3.1 describes the use of features in MESONet and difficulties associated with the developmental learning approach to network intrusion detection. Section 3.2 describes the different types of features in MESONet and requirements for features to be effective in a developmental learning environment. Section 3.3 presents examples of the flexibility afforded by the MESONet framework to feature vector composition, and Section 3.4 details all features used in MESONet.

3.1 Features in MESONet

We define MESONet features to be scalar values that are derived from characteristics of network packets. A feature vector, or sample, is thus an array of scalar values in which each value has been derived from a network packet. This process of calculating a feature vector is termed *feature extraction*. Some features may directly depend on data within the packet (for example, packet length), while others may depend on characteristics of the packet itself (for example, the time of arrival). For every packet received by MESONet, a new feature vector is extracted, labeled, and delivered to the DLA, which then performs either training or testing.

The boundary between classifiers (such as MESOTree, discussed in Section 2.2) and feature extraction is somewhat arbitrary [14]. An ideal set of features would make classification trivial, just as an omnipotent classifier has no need of sophisticated feature extraction. The practical goal of feature extraction is to seek *discriminating* features – those features that have similar values for samples that have the same class, and different values for samples that have different classes. One of the strengths of the developmental learning approach is that very little penalty is paid for utilizing a large number of features, which allows us to err on the side of too many features (however, performance can be penalized for using "bad" features, as will be shown in Chapter 4).

The selection of the particular features to extract from network packets is the main difficulty in the application of developmental learning to network intrusion detection. In more traditional applications of developmental learning, for example in robotics [51], sensors and features are naturally related: Each feature is extracted from a sensor that relates to the physical environment in which the robot operates. However, in MESONet all features must be extracted from packets, which has no such natural model.

In MESONet, selecting the particular features to be extracted from network packets relies heavily on domain knowledge, much as in other forms of classification [14]. In general, the approach we have taken is to include those features that "cover" the distinguishing characteristics of all network attacks appearing in the IDSEval data, without developing specific features for that attack.

For example, consider the "LAND" attack [26]. This attack can be detected by a single TCP SYN packet that has equal source and destination IP addresses and port numbers. We can detect LAND by extracting a single indicator feature that checks for exactly this condition, for example:

```
if( (pkt.src_address == pkt.dst_address) &&
  (pkt.src_port == pkt.dst_port) &&
  pkt.is_tcp && pkt.is_syn ) {
    return LAND_ATTACK;
else
    return NORMAL;
```

However, this approach is not scalable when we consider that the IDSEval consists of many different types of attacks, not to mention that this approach is ineffectual at detecting zero-day attacks.

Alternatively, we can extract a series of more general features, which when used in concert are capable of detecting the LAND attack. These features include indicator features for the type of packet (TCP, UDP, or ICMP), as well as metrics for the similarity of source and destination IP address and port numbers. These features assume certain values when extracted from a LAND attack packet, and different values when extracted from a normal packet. We then rely upon MESO-tree to distinguish between the two different feature vectors. Table 3.1 is a listing of the features that detect the LAND attack, how they are calculated, and the values

Feature	Calculation	Attack	Normal
IP Address Distance	src_ip[0] XOR dst_ip[0]	0	$\neq 0$
	src_ip[1] XOR dst_ip[1]	0	$\neq 0$
	src_ip[2] XOR dst_ip[2]	0	$\neq 0$
	src_ip[3] XOR dst_ip[3]	0	$\neq 0$
Port Distance	src_port XOR dst_port	0	$\neq 0$
isTCP	pkt.is_TCP ? True : False	True	True/False
isSYN	pkt.is_SYN ? True : False	True	True/False

Table 3.1: Features used to detect the LAND attack.

that distinguish between attack and normal packets. It should be noted that, due to training, only packets that *are most similar* the values listed in the Attack column are considered attacks.

3.2 Feature Types and Requirements

Four different types of features are used in MESONet: packet features, calculated features, windowing features, and grouping features.

Packet Features. These features are directly related to values and characteristics intrinsic to a packet. For example, packet length, protocol type, time of arrival, and flag values are all packet features.

Calculated Features. These features are not derived directly from fields in network packets. Rather, they include composites or transformations of packet characteristics, as well as multi-packet characteristics, such as the running average of packet length, interpacket delay, and average packet rate. We refer to calculated features that have a dependence upon past packets as exhibiting "memory." For example, average interpacket delay depends not only on the arrival time of the current packet, but also on the arrival time of previous packets.

Windowing Features. These "meta-features" alter the feature vector so as to include features from multiple packets. The rationale for using windowing features is that many intrusions cannot be detected in a single packet, so we must include features from multiple packets within a single feature vector. For example, a five-packet windowing feature concatenates all features from four previous packets to the features from a fifth packet, generating a single, larger feature vector.

Grouping Features. These "meta-features" are used to select the particular feature extractor to use for any captured packet. Grouping enforces that all packets using a particular feature extractor either belong to the same flow, are destined for the same host, or originated from the same host. Grouping features are particularly useful when combined with windowing, in which case all packets used to construct final feature vectors are guaranteed to be related (they are destined for, and/or originate from, the same host).

In addition to being one of these types, features must satisfy requirements regarding ordering and boundedness to be effective within a developmental learning environment. These requirements help to ensure that the calculation of a distance metric (such as Euclidean distance; see Section 2.2) is meaningful with respect to observed features.

Ordering. Feature values must obey a semantically correct strict weak ordering[1]. For example, when given three different samples that have values x, y, z for the same feature, such that $x < y < z$ and $y - x < z - y$, it is semantically valid to say that x and y are "closer" to each other than to z. This requirement stems from the use of a metric to determine the distance between multiple feature vectors.

In MESONet, this requirement manifests itself in the handling of *categorical variables*, specifically nominal categorical variables such as the IP protocol and the ICMP message type fields. For example, packet length is a quantitative variable - it has meaning as a number, and obeys a semantically correct strict weak ordering. However, ICMP message type is a categorical variable; values include 1 (echo-reply),

[1] A strict weak ordering is a partial order over the "less than" relation. By semantically correct strict weak ordering, we mean that it must "make sense" to compare the feature values.

3 d

sn

is

ma

va

pr

sav

we

exa

.1

pac

Ea

th

pa

gr

an

di

va

ar

w

F

10

th

t

3 (destination unreachable), and 8 (echo-request), among others. If we applied a strict weak ordering to ICMP message type, we would determine that echo-reply (1) is "closer" to destination unreachable (3) than to echo-request (8) – obviously semantically meaningless. Similarly, the protocol field in IP datagrams is a categorical variable. If we applied a strict weak ordering to this field we would find that ICMP (protocol 1), is "closer" to TCP (protocol 6), than UDP (protocol 17). Obviously, saying that ICMP is "close" to TCP is also semantically meaningless.

To mitigate this problem, we split each categorical variable appearing in network packets into a set of indicator variables, one for each possible category. For example, we split the IP protocol field into three different indicator variables: isTCP (1 if the packet is a TCP packet, 0 otherwise), isUDP (1 if the packet is a UDP packet, 0 otherwise), and isICMP (1 if the packet is an ICMP packet, 0 otherwise). Each packet will only have one of these indicator variables set, and it is now semantically correct to say that a packet with the isTCP variable set is "close" to another packet that also has the isTCP variable set – in fact, they are equal in that feature.

Boundedness. The range of feature values must be bounded. For example, given any feature x, the maximum value that can be assumed by x must be known, and fixed. This requirement, too, stems from the use of a metric to determine the distance between multiple feature vectors.

In MESONet, this requirement manifests itself in the handling of quantitative variables, specifically in "counting" variables such as "the count of ICMP packets" and "the count of TCP fragments." This requirement is particularly important when the quantity of training traffic is significantly different than testing traffic. For example, let us consider a feature "ICMPCount," that counts the number of ICMP packets captured. If training consisted of one hour of traffic, during which time only 10 ICMP packets were captured, and testing consisted of 10 hours of traffic with an average ICMP rate of 10 per hour, then the final ICMP packets will have a Euclidean distance of *at least* 90 from the training traffic. Applying a known bound removes this dependence upon duration of testing and training.

To mitigate this problem, we encode such unbounded features as a percentage. For example, ICMPCount is encoded as the percentage of ICMP traffic versus IP traffic.

3.3 Feature Composition

MESONet is a framework for extracting features from network traffic in a manner suitable for use with developmental learning algorithms. To facilitate rapid training and testing, MESONet supports a variety of runtime selectable features, and is easily extended to incorporate new features.

Dynamic composition of feature sets, together with the ease of developing new feature classes, also allow us to easily incorporate other research efforts into the MESONet framework. For example, both Bro [41] and NFR [39] produce audit data that is suitable for inclusion into MESONet as features. Similarly, system call information [7] and nonstationary models [35] may also be incorporated into MESONet as features. In short, any scalar value that can be calculated from a packet stream can be easily included in MESONet. Appendix A explores the development of new features in detail.

For any given training or testing period, MESONet dynamically assembles a hierarchy of Java objects representing the current *feature set*, or the list of features to be extracted from each network packet. These objects perform any needed calculation to extract features from packets. The composition of the feature set is determined by an XML configuration file. For example, **`<featureStrategy classname="SinglePacketFeatures"/>`** is an example of a very simple feature set, comprised of a single Java class, in this case **`SinglePacketFeatures`**. Although

```
<featureStrategy classname="CompoundFeatures">
    <arg name="subStrategy">
        <featureStrategy classname="SinglePacketFeatures"/>
    </arg>
    <arg name="subStrategy">
        <featureStrategy classname="PayloadCountFeatures"/>
    </arg>
</featureStrategy>
```

Figure 3.1: A feature set for MESONet, consisting of all the features extracted by both the `SinglePacketFeatures` and `PayloadCountFeatures` classes.

`SinglePacketFeatures` is contained within a single XML element, it extracts 20 different features from every network packet. The ability for a class to provide multiple features allows logically related features to be collected in a single location, which not only improves performance but enhances maintainability. The particular features that are extracted by each class mentioned here are described in detail in Section 3.4.

Figure 3.1 describes a more complex feature set. In this case, `CompoundFeatures` is used to aggregate the features that are extracted by both `SinglePacketFeatures` and `PayloadCountFeatures` into a single feature vector. Again, although it is contained within a single XML element, `PayloadCountFeatures` extracts 24 features from every network packet. It should be noted that `CompoundFeatures` does not itself extract any features from network packets; it is used only to assist in assembling the final feature vector. In this case, the resultant feature vector comprises 44 features; 20 from `SinglePacketFeatures`, and 24 from `PayloadCountFeatures`.

`SinglePacketFeatures` and `PayloadCountFeatures` are examples of packet features, as described in Section 3.2. Figure 3.2 describes a feature set containing a 5-packet window of calculated features. `SlidingWindowFeatures` maintains a list of features that were previously extracted by the classes it encapsulates. This

```
<featureStrategy classname="SlidingWindowFeatures">
    <arg name="windowSize" value="5"/>
    <arg name="subStrategy">
        <featureStrategy classname="PacketNumberFeatures"/>
    </arg>
</featureStrategy>
```

Figure 3.2: A feature set for MESONet, consisting of a 5-packet sliding window of all the features extracted by `PacketNumberFeatures`.

list is treated as a FIFO sliding window, similar in spirit to TCP sliding windows, whereby older features are replaced by newer features after a certain number have been stored. There is no restriction placed upon the number of features that may be extracted at a time - indeed, `PacketNumberFeatures` extracts 6 different features from each packet.

Finally, Figure 3.3 describes a 5-packet window of `SinglePacketFeatures` that is grouped by destination IP address. In this case, `HostFeatures` instantiates a new feature set hierarchy for every unique destination IP address occurring in all captured network packets. This ensures that the same objects will be used to extract features from subsequent network packets belonging to the same packet stream. This is particularly useful when used with `SlidingWindowFeatures`, as it ensures that all features in the window belong to the same packet stream.

3.4 Feature Detail

As mentioned in Section 3.3, the ability of a single Java class to extract multiple features from a network packet provides a logical grouping for related features. Based on this grouping, this section describes a taxonomy of the features that can currently be extracted by the MESONet framework. Table 3.2 summarizes all individual features described in this section, and contains the feature name, the class in

```
<featureStrategy classname="HostFeatures">
    <arg name="sortby" value="dst"/>
    <arg name="subStrategy">
        <featureStrategy classname="SlidingWindowFeatures">
            <arg name="windowSize" value="5"/>
            <arg name="subStrategy">
                <featureStrategy classname="SinglePacketFeatures"/>
            </arg>
        </featureStrategy>
    </arg>
</featureStrategy>
```

Figure 3.3: A feature set for MESONet, consisting of a 5-packet sliding window of all the features extracted by `SinglePacketFeatures`, grouped by destination IP address.

which it is defined, and a brief description.

CompoundFeatures. The `CompoundFeatures` class, as alluded to in Section 3.3, aggregates features extracted by other classes into a single feature vector; in essence, it is a container for other feature extractors. It does not, itself, extract features from network packets. Configuration parameters to `CompoundFeatures` are:

- **subStrategy:** Any number of XML elements containing configuration information for the feature extractors to be aggregated.

DerivativeFeatures. The `DerivativeFeatures` class, similar to `CompoundFeatures`, is also a container for other feature extractors, however `DerivativeFeatures` also augments the resultant feature vector with the first derivative of the contained features. For example, if the upcoming `PacketNumberFeatures` class was contained by the `DerivativeFeatures` class, then the resultant feature vector would contain scalar values for both the percentage of TCP traffic and the rate at which the percentage of TCP traffic has been changing. The derivative is calculated by averaging the change between successive

Feature Name	Class	Description
Derivative Features	DerivativeFeatures	Augments contained feature class with its derivative.
Sliding Window	SlidingWindowFeatures	Sliding window of previous feature vectors.
Host Grouping	HostFeatures	Groups feature vectors by source and/or destination.
Compound	CompoundFeatures	Aggregates features from contained feature classes.
IP Address Distance	SinglePacketFeatures	Distance between source and destination IP addresses.
Port Number Distance	SinglePacketFeatures	Distance between source and destination port numbers.
Fragment Offset	SinglePacketFeatures	IP fragment offset
Packet Length	SinglePacketFeatures	Total length of packet
Packet Header Length	SinglePacketFeatures	Length of packet header
Is TCP	SinglePacketFeatures	Indicator; 1 if TCP, 0 else
Is UDP	SinglePacketFeatures	Indicator; 1 if UDP, 0 else
Is ICMP	SinglePacketFeatures	Indicator; 1 if ICMP, 0 else
TCP ACK	SinglePacketFeatures	Indicator; 1 if TCP and ACK, 0 else
TCP FIN	SinglePacketFeatures	Indicator; 1 if TCP and FIN, 0 else
TCP PSH	SinglePacketFeatures	Indicator; 1 if TCP and PSH, 0 else
TCP RST	SinglePacketFeatures	Indicator; 1 if TCP and RST, 0 else
TCP SYN	SinglePacketFeatures	Indicator; 1 if TCP and SYN, 0 else
TCP URG	SinglePacketFeatures	Indicator; 1 if TCP and URG, 0 else
Payload Count	PayloadCountFeatures	Counts of certain characters in payload
Payload Value	PayloadValueFeatures	ASCII value of bytes in payload
Interpacket Delay	TimeFeatures	Mean time between subsequent packets
Packet Rate	TimeFeatures	Packets per second
%SYN Packets	PacketNumberFeatures	Percent SYN traffic
%FIN Packets	PacketNumberFeatures	Percent FIN traffic
%RST Packets	PacketNumberFeatures	Percent RST traffic
%TCP Packets	PacketNumberFeatures	Percent TCP traffic
%UDP Packets	PacketNumberFeatures	Percent UDP traffic
%ICMP Packets	PacketNumberFeatures	Percent ICMP traffic

Table 3.2: Summary of feature classes.

feature values over a configurable number samples. Configuration parameters to `DerivativeFeatures` are:

- **maxWindowSize:** The number of samples over which the change in feature value is averaged.
- **granularity:** The minimum amount of time that must pass between samples to trigger a recalculation of the derivative. If large (compared to sample frequency), `DerivativeFeatures` will ignore short-term, or impulse, behavior.
- **subStrategy:** An XML element containing configuration information for the contained feature extractor; can be `CompoundFeatures`.

HostFeatures. The `HostFeatures` class, as mentioned in Section 3.3, constructs a new feature set hierarchy based upon the source or destination of captured packets. Internally, `HostFeatures` maintains a hash table of IP addresses to feature extractors. For every captured packet, a lookup in this hash table is performed, based on either the source or destination IP address of the packet. The resultant feature extractor is then used to extract features from the current packet. This approach is most powerful when used in concert with stateful feature extractors, such as `SlidingWindowFeatures` or `DerivativeFeatures`, which will then be guaranteed that all feature extractions are performed on packets from the same stream. Finally, `HostFeatures` can also be nested with itself to allow feature extraction from the same source-destination IP address pair. Configuration parameters to `HostFeatures` are:

- **sortby:** May have value "src" or "dst," which provide either source or destination grouping.

Feature Name	Description
% SYN	The percentage of traffic with the "SYN" TCP flag set. SYN packets signify the beginning of a TCP connection, and are also used during some DoS attacks.
% FIN	The percentage of traffic with the "FIN" TCP flag set. FIN packets signify the end of a TCP connection, and are also used by attackers for reconnaissance purposes.
% RST	The percentage of traffic with the "RST" TCP flag set. RST packets are used to abort an improper TCP connection setup, and are used during some DoS attacks.
% TCP	The percentage of traffic that belongs to the TCP protocol.
% UDP	The percentage of traffic that belongs to the UDP protocol.
% ICMP	The percentage of traffic that belongs to the ICMP protocol.

Table 3.3: Packet number feature detail.

- **subStrategy:** An XML element containing configuration information for the contained feature extractor; can be another `HostFeatures` with different sorting, to provide unique source-destination grouping.

PacketNumberFeatures. The `PacketNumberFeatures` class calculates the percentage of traffic that meets certain protocol characteristics. For example, this class calculates the percentage of traffic that is TCP. There are no configuration parameters to `PacketNumberFeatures`. The specific features calculated by `PacketNumberFeatures` are shown in Table 3.3.

PayloadCountFeatures. The `PayloadCountFeatures` class extracts features from the payload portion of captured packets. Rather than searching for specific strings of characters, or particular application protocols, `PayloadCountFeatures` counts the number of certain characters appearing in the payload. The alphabet of characters counted includes printable symbolic characters, many of which are used in network attacks (for example, thousands of backslash characters appear in the payload of the "back" attack). The motivation behind counting characters is to avoid problems with ordering, as described in Section 3.2. Configuration parameters to `PayloadCountFeatures` are:

- **payloadLength:** Specifies the maximum number of payload bytes to scan for counted characters.

The specific features calculated by `PayloadCountFeatures` are:

- **Character Alphabet:** The count of certain characters appearing in the packet payload. One feature for each of the following characters is extracted:

  `{ } / \ ~ ! @ # $ % * ( ) - _ = + ‘ ’ < > , . ?`

PayloadValueFeatures. The `PayloadValueFeatures` class extracts features from the payload portion of captured packets. In opposition to `PayloadCountFeatures` and the ordering principle discussed in Section 3.2, this class copies the ASCII value of payload bytes directly into the feature vector. This feature extractor is used primarily as motivation to follow the ordering principle. Configuration parameters to `PayloadValueFeatures`:

- **payloadLength:** Specifies the maximum number of bytes to copy into the feature vector.

The specific features calculated by `PayloadValueFeatures` are:

- **Character Value(s):** One feature for each ASCII character value of payloadLength bytes are copied into the feature vector.

SinglePacketFeatures. The `SinglePacketFeatures` class extracts features from each packet based upon the packet's protocol, flags, source, and destination.

Feature Name	Description
IP Distance	Actually four features, one for each IP address byte, IP distance is used to measure the similarity between source and destination *network masks*. The calculation is performed by taking the XOR of the source and destination IP address bytes: $src[i] \oplus dst[i]$. These feature values will be zero for packets that have identical source and destination addresses, and nonzero otherwise.
Port Number Distance	Similar to IP distance, port number distance measures the similarity between source and destination port numbers, and is also calculated via an XOR: $src\_port \oplus dst\_port$. This feature is only extracted from TCP and UDP packets.
Fragment Offset	The packet's fragment offset from the IP header.
Packet Length	The total length of the packet.
Packet Header Length	The length of the header portion of the packet.
Is TCP	An indicator feature, this feature is set to 1 if the captured packet is a TCP packet, and 0 otherwise.
TCP ACK	An indicator, this feature is set to 1 if the captured packet is a TCP packet and the ACK bit is set, 0 otherwise.
TCP FIN	An indicator, this feature is set to 1 if the captured packet is a TCP packet and the FIN bit is set, 0 otherwise.
TCP PSH	An indicator, this feature is set to 1 if the captured packet is a TCP packet and the PSH bit is set, 0 otherwise.
TCP RST	An indicator, this feature is set to 1 if the captured packet is a TCP packet and the RST bit is set, 0 otherwise.
TCP SYN	An indicator, this feature is set to 1 if the captured packet is a TCP packet and the SYN bit is set, 0 otherwise.
TCP URG	An indicator, this feature is set to 1 if the captured packet is a TCP packet and the URG bit is set, 0 otherwise.
Is UDP	An indicator feature, this feature is set to 1 if the captured packet is a UDP packet, and 0 otherwise.
Is ICMP	An indicator feature, this feature is set to 1 if the captured packet is an ICMP packet, and 0 otherwise.

Table 3.4: Single packet feature detail.

In short, it extracts features from the packet header. There are no configuration parameters to **`SinglePacketFeatures`**. The specific features calculated by **`SinglePacketFeatures`** are shown in Table 3.4.

SlidingWindowFeatures. The **`SlidingWindowFeatures`** class, as alluded to in Section 3.3, creates a FIFO sliding window, similar to TCP sliding window, of features extracted by its contained class. It can be thought of as adding "history" to the feature vector; **`SlidingWindowFeatures`** adds a configurable number of prior feature vectors to every feature vector extracted by the contained class. **`SlidingWindowFeatures`** does not, itself, extract features from network packets. Configuration parameters to **`SlidingWindowFeatures`** are:

- **subStrategy:** An XML element containing configuration information for the contained feature extractor; can be **`CompoundFeatures`**.

TimeFeatures. The **`TimeFeatures`** class extracts time-based features from network packets, specifically mean interpacket delay and packets per second statistics. When combined with **`HostFeatures`**, **`TimeFeatures`** can calculate these statistics for all packets destined for a host, thus helping to detect DoS attacks. Note that using **`DerivativeFeatures`** in conjunction with **`TimeFeatures`** would calculate the rate of change of interpacket delay and the packet acceleration. The specific features calculated by **`TimeFeatures`** are shown in Table 3.5. Configuration parameters to **`TimeFeatures`** are:

- **windowSize:** The number of packets over which statistics are to be calculated.

Feature Name	Description
Mean Interpacket Delay	The average of the time between subsequent network packets.
Packets Per Second	The number of network packets arriving during the last second.

Table 3.5: Time feature detail.

Chapter 4

Misuse Detection

Misuse detection systems search network traffic for known attack signatures. Although misuse detection systems are unable to reliably detect zero-day attacks, they are able to detect known attacks with very low false positive rate [20,21], and are thus a critical component of effective network intrusion detection.

Experiments presented in this chapter show largely negative results; that is, although MESONet achieves good accuracy when performing misuse detection, it does not achieve a good recall rate. These experiments were, however, integral to our understanding of feature extraction, and ultimately led us to better results with anomaly detection, as will be shown in Chapter 5.

The remainder of this chapter describes misuse detection with MESONet, and is organized as follows. Section 4.1 describes the training and testing procedures for misuse detection with MESONet. Section 4.2 describes the analyses used to evaluate the performance of misuse detection with MESONet. Sections 4.3 through 4.6 describe experimental results using different feature sets, and Section 4.7 summarizes misuse detection with MESONet.

Tracefile	Source	Use	Tracefile Size	Packet Count	Attacks
A	3/8/1999	training	329MB	1263543	7; 7178 packets
B	3/11/1999	testing	330MB	1263543	9; 25047 packets

Table 4.1: Tracefiles used for misuse detection with MESONet.

4.1 Training and Testing Procedures

Performing misuse detection with MESONet requires both training and testing phases. Table 4.1 summarizes the tracefiles used to perform training and testing. Tracefile A, used for training, contains 1.25M normal packets and 7,178 attack packets, comprising 7 different network attacks. Tracefile B contains 1.26M normal packets and 25,047 attack packets, comprising 9 different network attacks. It is typical in network intrusion detection for the normal traffic to far outweigh the attack traffic [8].

Figure 4.1, termed a *packet graph*, shows the specific attacks occurring in the training data (Tracefile A). Each point in Figure 4.1 represents one attack packet (for the sake of clarity, normal packets are not plotted; there are 1.25M of them). As shown in this figure, 7 different attacks occur in Tracefile A: NTInfoscan, pod, back, httptunnel, land, secret, and ps attack [26]. Of these attacks, httptunnel contains the most attack packets, 5,061 out of 7,178.

Figure 4.2 is a packet graph of the attacks occurring in Tracefile B, the tracefile used for misuse detection testing. As shown in this figure, 8 different attacks occur in Tracefile B, comprising 25,047 packets. Furthermore, only one of the attacks in Tracefile B is also represented in Tracefile A: the LAND attack. Effective misuse detection requires that MESONet be trained with attacks that are representative of the attacks expected to appear in the testing traffic. However, the realities of network intrusion detection dictate that misuse detection systems also be evaluated using non-trained attacks. For this reason, the testing data used with MESONet contains 7 non-trained attacks.

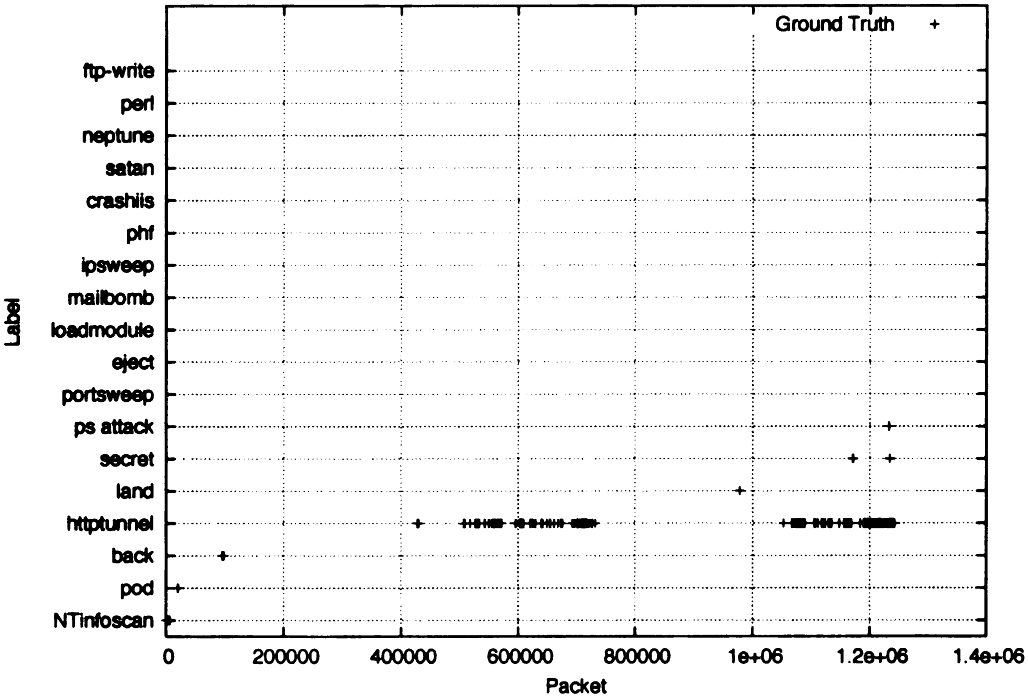


Figure 4.1: Tracefile A ground truth packet graph.

All misuse detection experiments are performed with a different feature set. Two different tests are performed with each feature set: a resubstitution test and a validation test. For the resubstitution test, MESONet is both trained and tested on the same data, in this case Tracefile A. This test is conducted to show the sufficiency of the feature set to be used for learning-based misuse detection. In theory, the results of a resubstitution test should be perfect, however, this is not always the case. Intuitively, resubstitution tests are performed to make sure that the feature set contains discriminating features; a poor resubstitution suggests that the feature set does not contain enough information to discriminate between attack and normal traffic. For the validation test, MESONet is trained on the entirety of Tracefile A and tested on the entirety of Tracefile B. This test is conducted to show

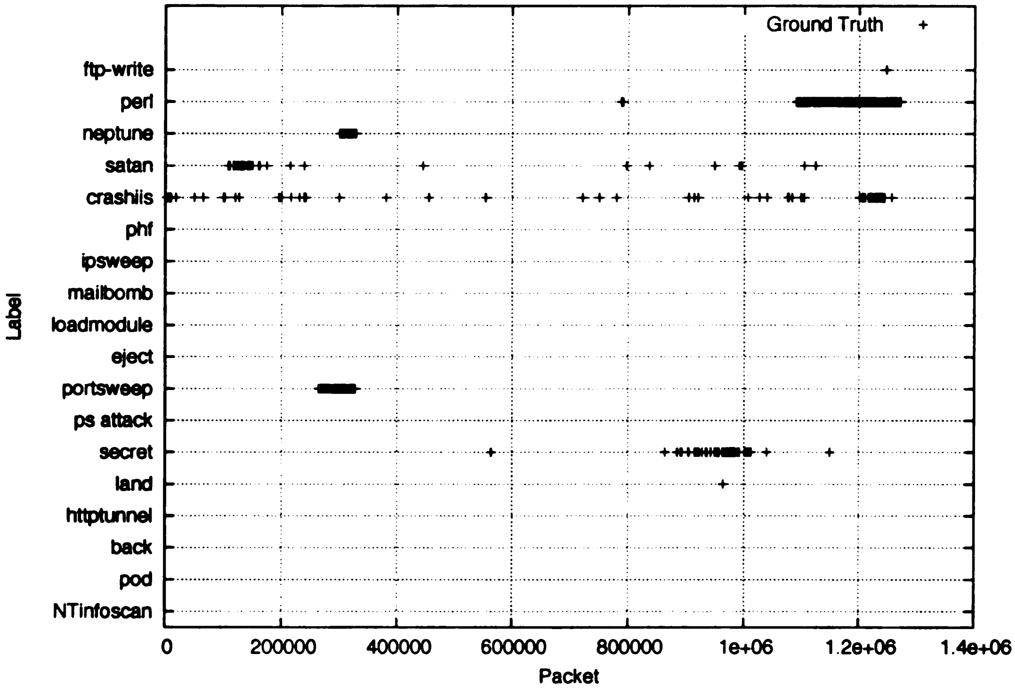


Figure 4.2: Tracefile B ground truth packet graph.

the applicability of what MESONet learned during training to a different tracefile.

4.2 Performance Analysis

The method by which performance of an intrusion detection system is judged is as important as the intrusion detection system itself [8,32,36]. In order to provide an unbiased performance analysis of misuse detection with MESONet, we generate packet graphs and calculate precision, recall, false positive rate, accuracy, and a confusion matrix for both resubstitution and validation tests.

Packet graphs allow an intuitive understanding of how MESONet classifies each packet by showing the attack classification of each packet.

ab

Re

nu

nu

po

als

pr

act

fals

rat

of

int

wh

for

If T

TP

eff

Re

Fa

A **confusion matrix**, also known as a correlation matrix, tabulates information about actual and predicted classifications performed by a classification system [5]. Referring to the sample confusion matrix in Table 4.2, the value in cell a is the number of correct normal predictions, also known as true negatives. Cell b is the number of normal packets that are predicted to be an attack, also known as false positives. Cell c is the number of attack packets that are predicted to be normal, also known as false negatives, and cell d is the number of attack packets that are predicted to be attack, also known as true positives.

		Predicted	
		Normal	Attack
Actual	Normal	a	b
	Attack	c	d

Table 4.2: Sample confusion matrix.

False positive rate is defined as the proportion of predicted attacks that are actually normal traffic. If $TN = TrueNegatives$ and $FP = FalsePositives$, the false positive rate FPR is defined as $FPR = FP/(FP + TN)$. A low false positive rate is desirable. Intuitively, false positive rate is a measure of the trustworthiness of attack predictions. False positive rate is also the typical method of evaluating intrusion detection systems, but is somewhat suspect due to the base-rate fallacy [8], which states that false positive rates well below .0001% may be needed to account for the vast number of packets per day, typically well over 1 million.

Precision is defined as the proportion of predicted attacks that are correct. If $TP = TruePositives$ and $FP = FalsePositives$, precision P is defined as $P = TP/(TP + FP)$. A high precision is desirable. Intuitively, precision measures the effectiveness of training to distinguish between attack and normal classes.

Recall is defined as the proportion of all attacks that are correctly predicted. Recall is equivalent to *True Positive Rate.* If $TP = TruePositives$ and $FN = FalseNegatives$, recall R is defined as $R = TP/(TP + FN)$. A high recall is desirable. Intuitively, recall measures the effectiveness of training to detect attacks.

Accuracy is defined as the proportion of correctly identified samples. If $S = Samples$, $TP = TruePositives$, and $TN = TrueNegatives$, accuracy A is defined as $A = (TP + TN)/S$. A high accuracy is desirable.

4.3 Single Packet Features

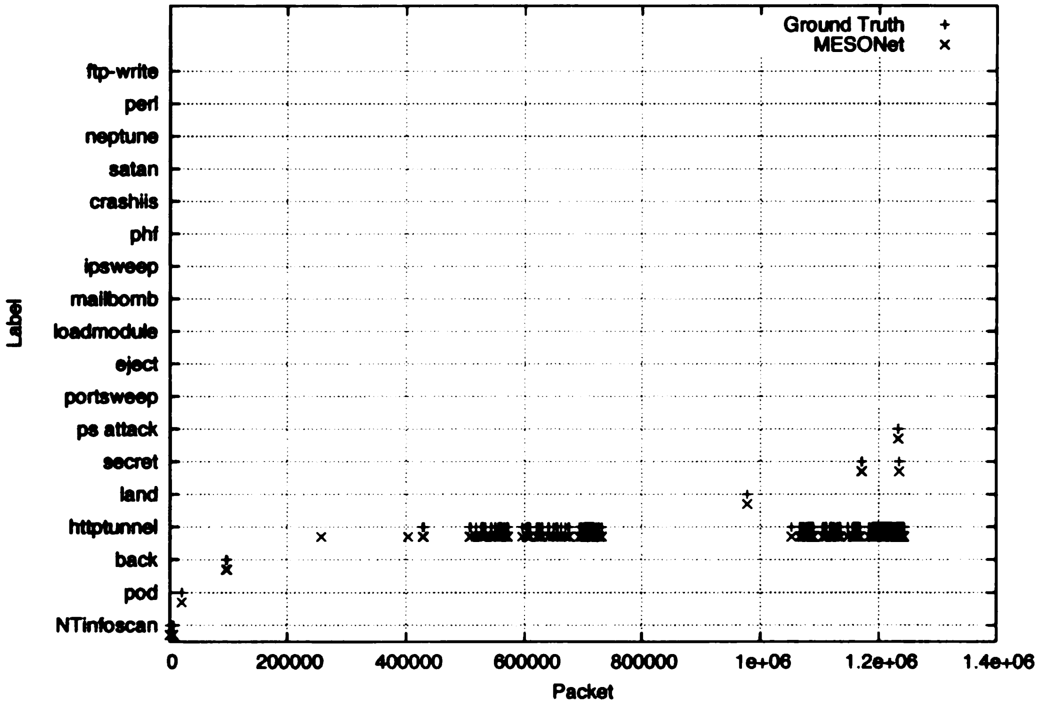
In this section we describe experiments based on features that can be extracted from a single network packet. Experiment 1 uses only the `SinglePacketFeatures` class, Experiment 2 adds the `PayloadCountFeatures` class, which is then replaced by `PayloadValueFeatures` in Experiment 3. All feature classes used here are described in Section 3.4. Complete feature set configuration for all experiments conducted in this chapter can be found in Appendix B.

Experiment 1. The first misuse detection experiment performed with MESONet used only the `SinglePacketFeatures` class. This class includes features such as IP address distance, protocol indicators, and payload length. The rationale behind selecting `SinglePacketFeatures` as the first test was that all network attacks have packet characteristics. Indeed, some network attacks, such as the "land" and "pod" attacks are detectable based solely on packet characteristics.

As shown by the resubstitution confusion matrix in Figure 4.3, less than half of the attack packets were correctly classified. The recall rate of 46% verifies that this feature set alone is insufficient for attack detection. This confusion matrix does show an interesting characteristic, one that will be carried through to other tests, and that is the relationship between the false positive and recall rates. A low false positive rate shows that MESONet is correctly learning the characteristics of normal packets; however, a low recall rate shows that MESONet is not learning the characteristics of attack packets. In other words, based on the results of the resubstitution test, we can hypothesize that this feature set adequately captures the behavior of normal packets, but not of attack packets.

The validation confusion matrix shown in Figure 4.4 verifies that **SinglePacketFeatures** is not a viable feature set for attack detection. Although accuracy is still high due to the large number of normal packets, recall and precision are low, 0.09% and 1.6% respectively. Furthermore, the false positive rate has reached 0.1%, representing 1,351 false alerts. Using **SinglePacketFeatures** alone describes normal packet characteristics with 98% accuracy, but is incapable of describing attack packet characteristics.

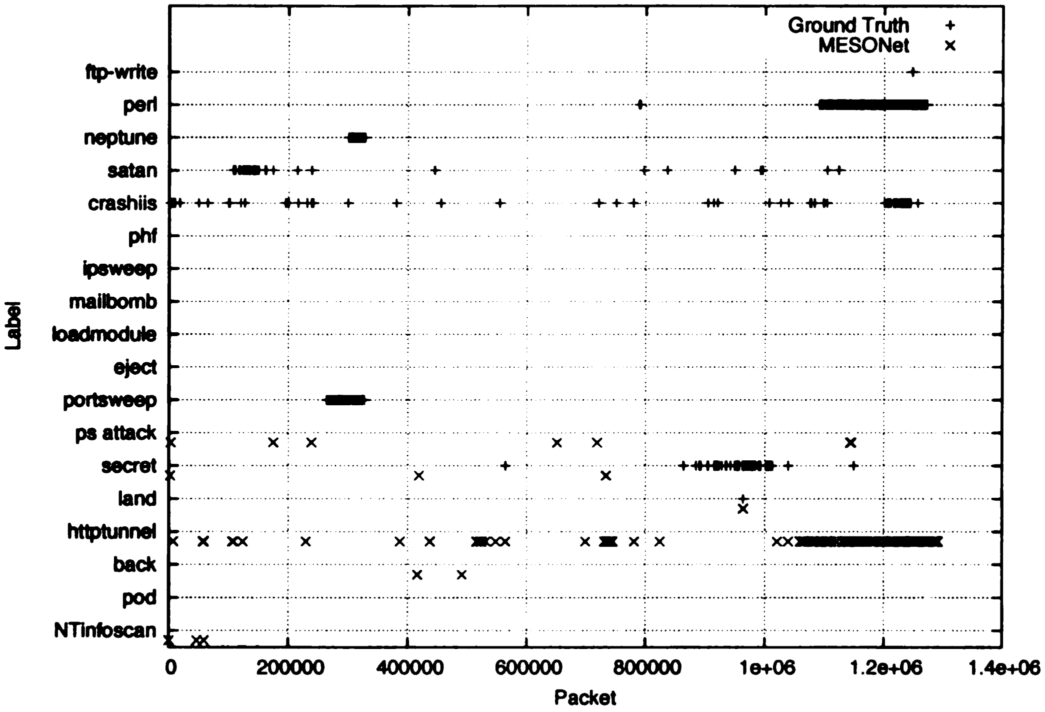
All future experiments include **SinglePacketFeatures** because, based on domain knowledge, we know that network attack detection must be predicated upon individual packet characteristics. Upcoming experiments show that although **SinglePacketFeatures** alone is not a sufficient feature set, performance is improved when **SinglePacketFeatures** is used in conjunction with other feature classes.



	Predicted	
	Normal	Attack
Normal	1256206	159
Attack	3819	3359

False Positive Rate	.0001
Accuracy	.9968
Recall	.4679
Precision	.9548

Figure 4.3: Experiment 1 resubstitution test results: packet graph (top) and confusion matrix (bottom).



	Predicted	
	Normal	Attack
Normal	1312660	1351
Attack	25024	23

False Positive Rate	.0010
Accuracy	.9803
Recall	.0009
Precision	.0167

Figure 4.4: Experiment 1 validation test results: packet graph (top) and confusion matrix (bottom).

Experiment 2. The second misuse detection experiment performed with MESONet augments `SinglePacketFeatures` with payload characteristics. Specifically, we add `PayloadCountFeatures` to the feature set. `PayloadCountFeatures` adds features for the count of certain characters appearing in each packet's payload. The rationale behind adding payload characteristics to the feature set is that many network attacks contain payload designed specifically to exploit network applications. For example, the "back" attack payload contains thousands of "/" characters.

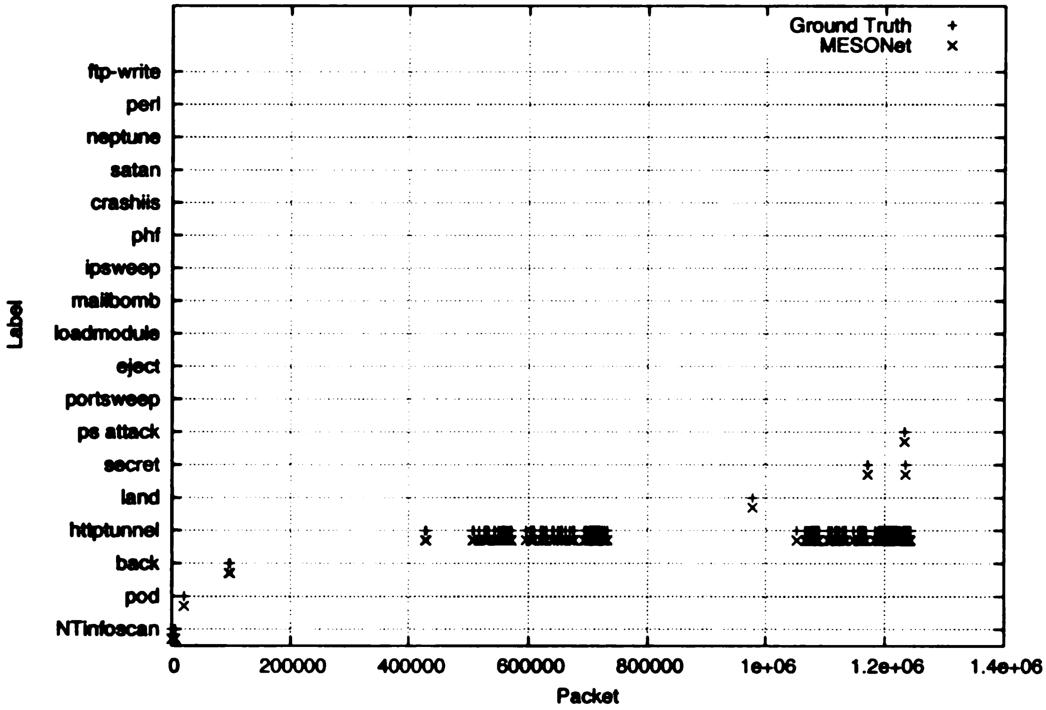
The resubstitution test confusion matrix in Figure 4.5 is different than that from Experiment 1 in two respects. First, the false positive rate increased from 0.01% to 0.06%. showing that this feature set has more difficulty distinguishing normal packets from attack packets. Second, the recall rate increased to 68%, showing that this feature set is better at describing the characteristics of attack packets.

The confusion matrix shown in Figure 4.6 verifies that `SinglePacketFeatures`, even when used in conjunction with `PayloadCountFeatures`, is still an insufficient feature set for misuse detection. The false positive rate is at the higher level of 0.37%, representing 4,956 false alerts. The high false positive rate confirms that this feature set does not accurately describe the characteristics of normal packets. However, recall rate has risen to 0.11%, representing an increase of 5 correctly detected attacks.

The packet graph in Figure 4.6 shows that there is a weak correlation between attack predictions made by MESONet and actual attack packets. For example, MESONet predicts a large series of "httptunnel" attack packets at the same time that the ground truth shows a series of "perl" attack packets. This behavior shows that, based on this particular feature set, there are similarities between these two attacks.

With the exception of Experiment 3, all future experiments include

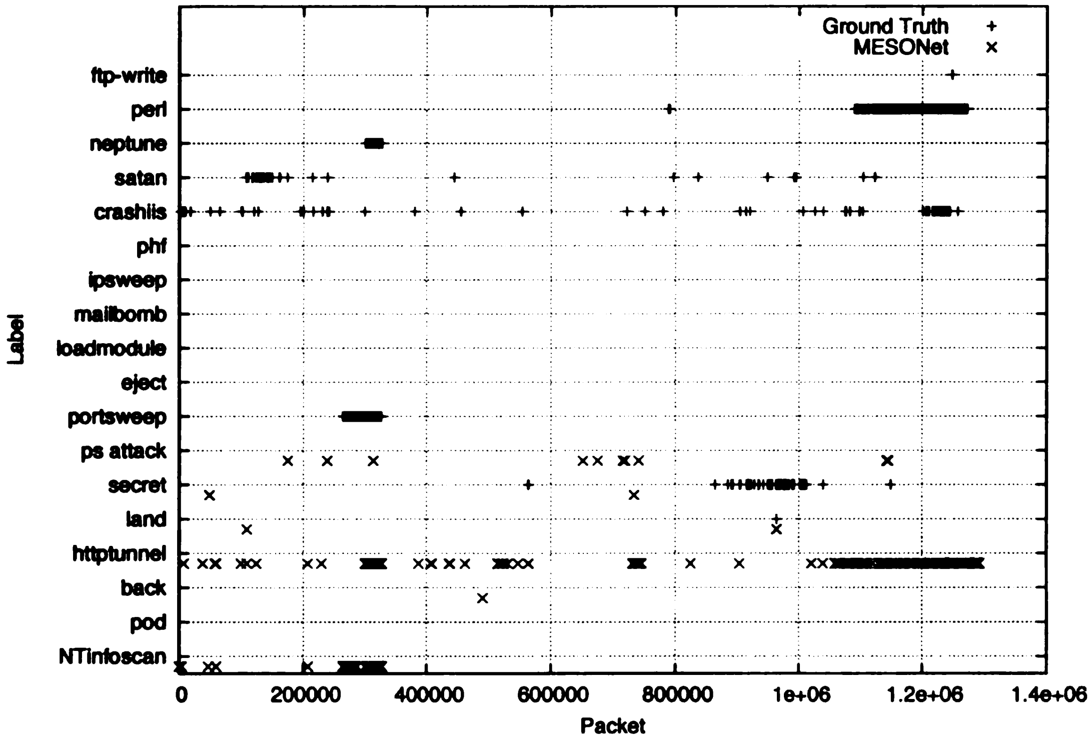
PayloadCountFeatures because, based on domain knowledge, we know that network attacks are payload specific. Upcoming experiments show that although **SinglePacketFeatures** and **PayloadCountFeatures** are not a sufficient feature set, performance is improved by the addition of other feature classes.



	Predicted	
	Normal	Attack
Normal	1255595	770
Attack	2228	4950

False Positive Rate	.0006
Accuracy	.9976
Recall	.6896
Precision	.8653

Figure 4.5: Experiment 2 resubstitution test results: packet graph (top) and confusion matrix (bottom).



	Predicted	
	Normal	Attack
Normal	1309055	4956
Attack	25019	28

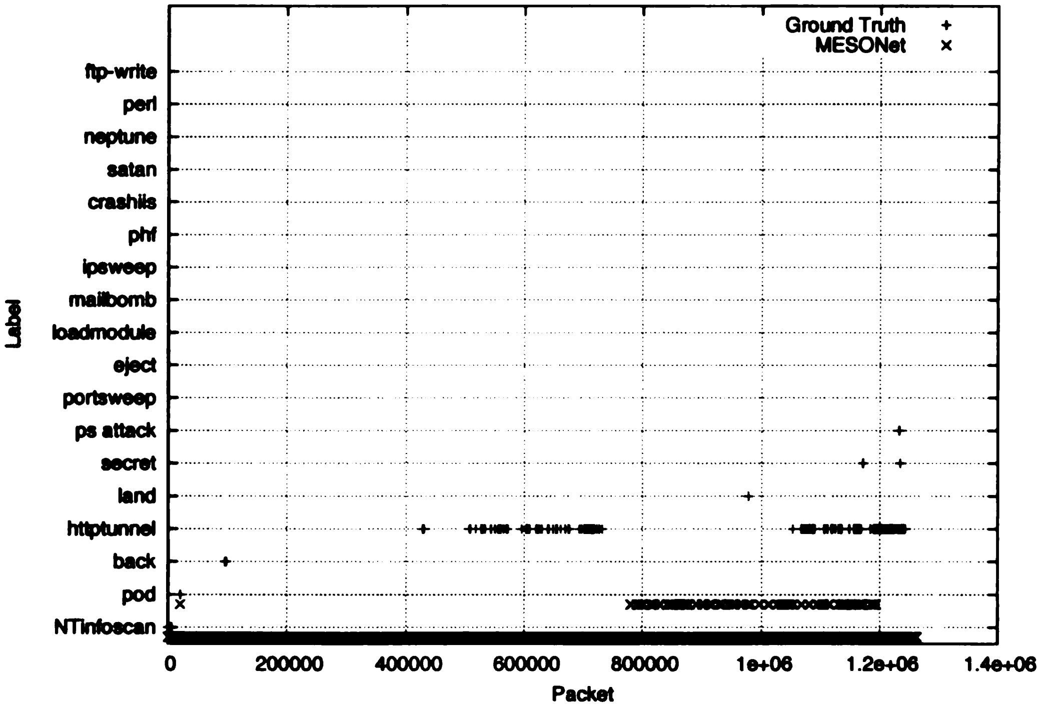
False Positive Rate	.0037
Accuracy	.9776
Recall	.0011
Precision	.0056

Figure 4.6: Experiment 2 validation test results: packet graph (top) and confusion matrix (bottom).

Experiment 3. The third misuse detection experiment performed with MESONet augments `SinglePacketFeatures` with `PayloadValueFeatures`. `PayloadValueFeatures` adds features for the ASCII value of the first 100 characters appearing in each packet's payload. It should be noted that the features added by the `PayloadValueFeatures` class violate the ordering requirement described in Section 3.2. In fact, the poor performance of this experiment was the motivation for the ordering requirement. As a result of this poor performance, upcoming experiments do not use `PayloadValueFeatures`.

Compared to previous resubstitution tests, the use of `PayloadValueFeatures` has a detrimental effect on misuse detection performance. The confusion matrix shown in Figure 4.7 lists 15,302 false positives and only 455 true positives. The recall rate of 6.3% verifies that this feature set is not sufficient for detecting attacks, while the high false positive rate of 1.2% shows that this feature set is incapable of distinguishing between attack and normal traffic.

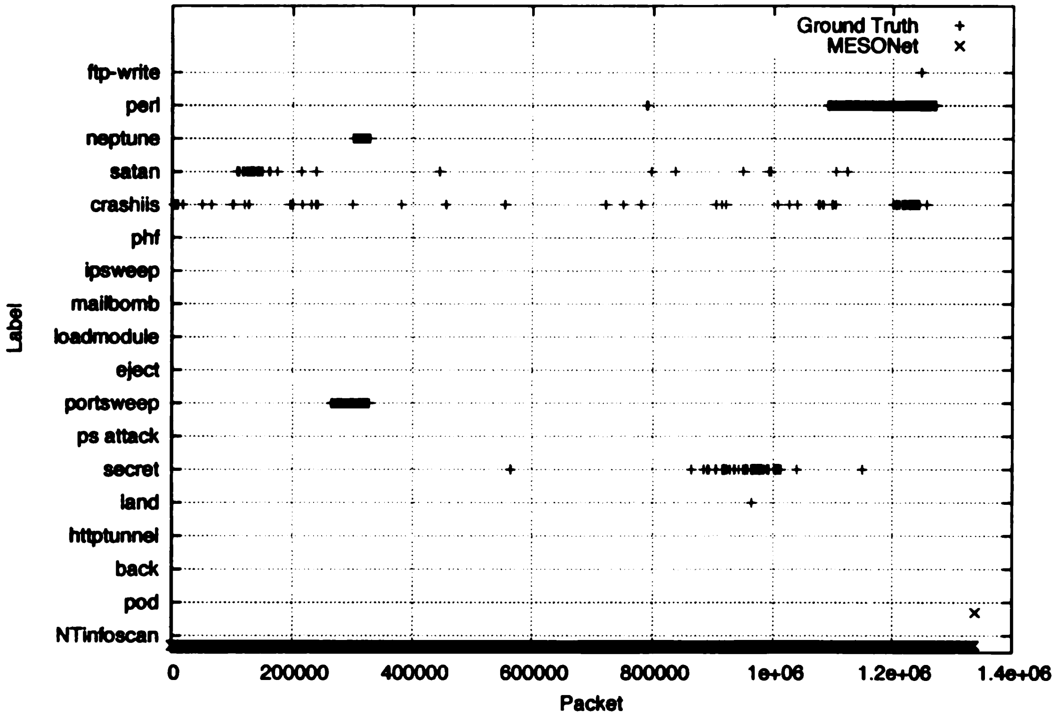
Even though the recall rate of 0.21%, shown in Figure 4.8, is larger than that of previous validation tests, the number of false positives has increased dramatically, to 18,125. The greater recall rate suggests that this feature set is better at detecting attacks, however, the false positive rate shows that MESONet is still incapable of distinguishing between attack and normal packets. Furthermore, the packet graph in Figure 4.8 shows that MESONet is detecting only a single attack, the "NTinfoscan" attack, and that the attack predictions are in no way correlated to actual attack packets.



	Predicted	
	Normal	Attack
Normal	1241063	15302
Attack	6723	455

False Positive Rate	.0121
Accuracy	.9825
Recall	.0633
Precision	.0288

Figure 4.7: Experiment 3 resubstitution test results: packet graph (top) and confusion matrix (bottom).



	Predicted	
	Normal	Attack
Normal	1295886	18125
Attack	24994	53

False Positive Rate	.0137
Accuracy	.9677
Recall	.0021
Precision	.0029

Figure 4.8: Experiment 3 validation test results: packet graph (top) and confusion matrix (bottom).

4.4 Sliding Window Features

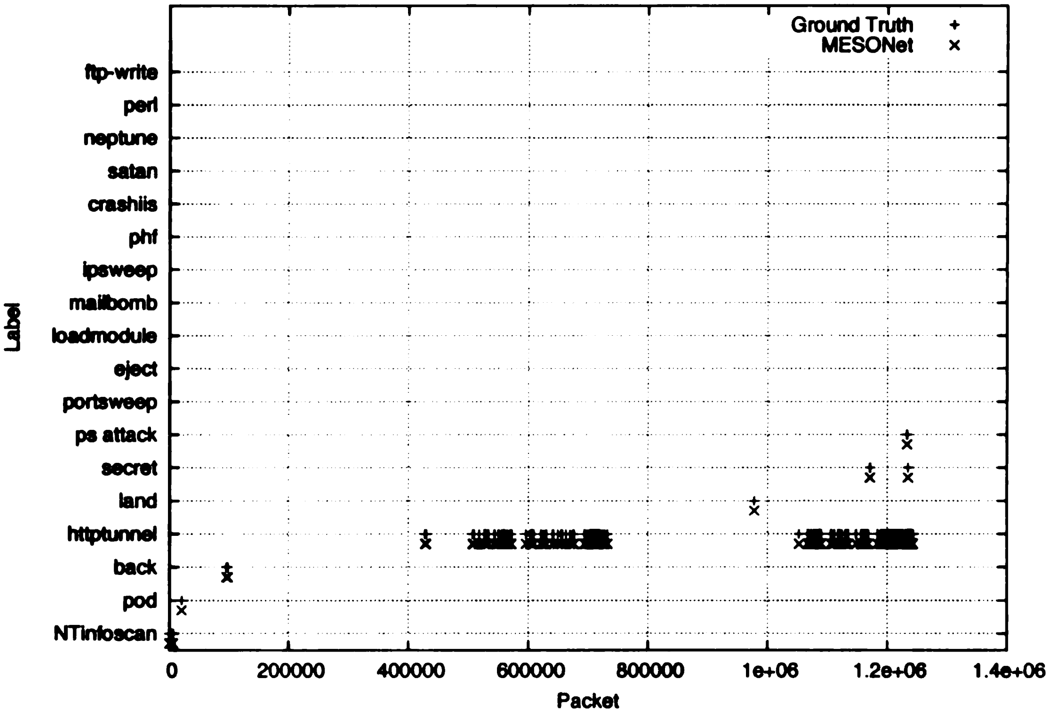
In this section we describe experiments based on the use of sliding window features. Experiment 4 adds `SlidingWindowFeatures` to the feature set from Experiment 2, Experiment 5 and 6 add different configurations of `TimeFeatures`, and Experiment 7 uses a larger sliding window. All feature classes used here are described in Section 3.4. Complete feature set configuration for all experiments conducted in this chapter can be found in Appendix B.

Experiment 4. The fourth misuse detection experiment performed with MESONet applies a two-packet sliding window to the feature set from Experiment 2. Specifically, we use `SlidingWindowFeatures` to construct feature vectors from the `SinglePacketFeatures` and `PayloadCountFeatures` of two packets. The rationale for using a sliding window of features is that very few network attacks consist of only a single packet. For example, in Tracefile A and B, only the "land" attack consists of a single packet. Therefore, accurate attack detection must consider features from multiple packets. It should be noted that the sliding window is updated with packets as they are captured off the network, that is to say, there may or may not be any correlation between the two feature sets in the window.

Through the use of `SlidingWindowFeatures`, both recall and false positive rates improve dramatically over previous experiments, as shown by Figure 4.9. Indeed, this particular feature set is the first to suggest that recall and false positive rates are independent of each other, and that they can both reach near-optimal values. Figure 4.9 lists a recall rate of 97.9%, verifying that the feature set is sufficient for detecting attacks, while the false positive rate of .01% shows that the training data and feature set together are capable of distinguishing between attack and normal packets. Of course, considering that these are resubstitution test results, near-optimal values are to be expected.

As shown by the confusion matrix in Figure 4.10, the number of correct attack predictions has increased dramatically as compared to previous experiments, from the previous maximum of 53 in Experiment 3 to the present 358. At the same time, false positives have reduced from 4,956 in Experiment 2 to 3050.

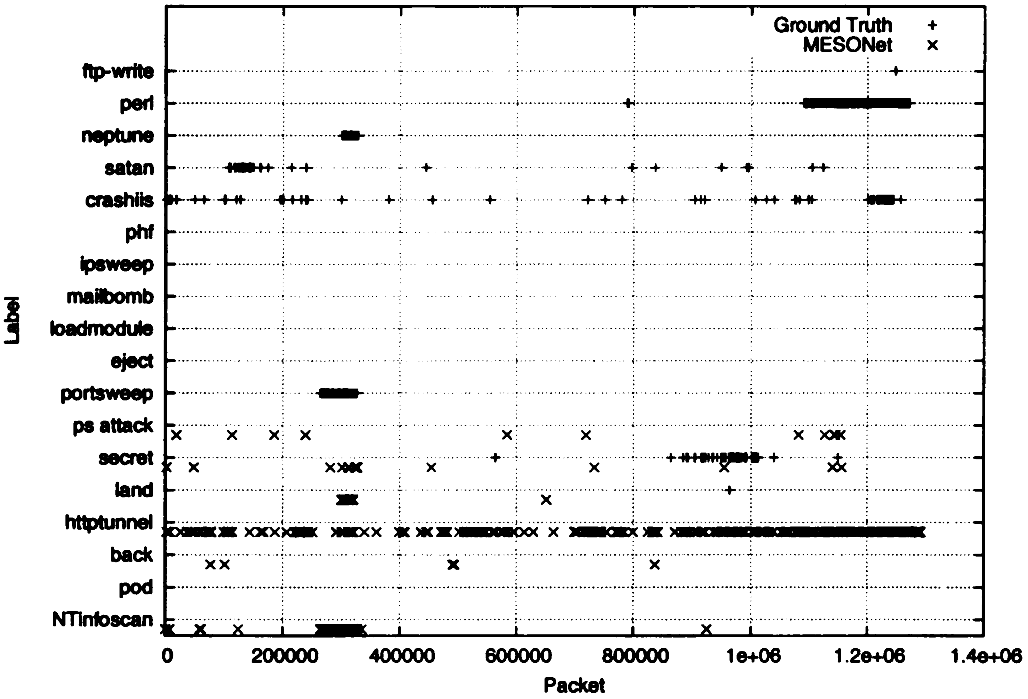
Use of `SlidingWindowFeatures` is clearly beneficial, however the false positive rate is still high. Subsequent experiments attempt to reduce the false positive rate, while minimizing adverse affect on recall rate. It should be noted that the 1.42% recall rate in this experiment is the maximum recall rate achieved on misuse detection with MESONet.



	Predicted	
	Normal	Attack
Normal	1256224	141
Attack	145	7033

False Positive Rate	.0001
Accuracy	.9997
Recall	.9797
Precision	.9803

Figure 4.9: Experiment 4 resubstitution test results: packet graph (top) and confusion matrix (bottom).



	Predicted	
	Normal	Attack
Normal	1310961	3050
Attack	24689	358

False Positive Rate	.0023
Accuracy	.9792
Recall	.0142
Precision	.1050

Figure 4.10: Experiment 4 validation test results: packet graph (top) and confusion matrix (bottom).

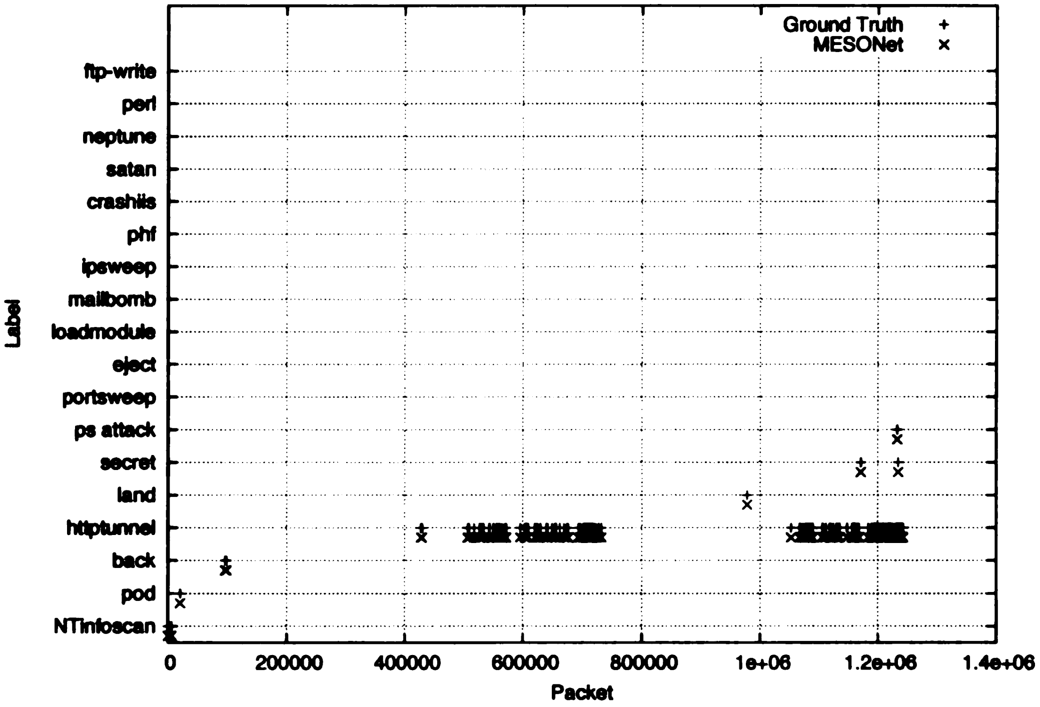
Experiment 5. The fifth misuse detection experiment performed with MESONet adds `TimeFeatures` to the feature set from Experiment 4. Specifically, we use `SlidingWindowFeatures` to construct feature vectors from the `SinglePacketFeatures`, `PayloadCountFeatures`, and `TimeFeatures` of two packets. `TimeFeatures` adds both interpacket delay and packet rate features. The rationale for including time-based features is based on observations of the timing characteristics of certain network attacks. For example, the "secret" attack has a highly regular timing pattern of packets.

As shown by the resubstitution test confusion matrix in Figure 4.11, the addition of `TimeFeatures` to the feature set has resulted in a near-ideal resubstitution test, with only 2 false positives and 1 false negative. The high recall rate of 99.9% signifies that the characteristics of the attacks are being learned, while the near-zero false positive rate shows that the feature set accurately describes characteristics of normal packets.

At the cost of 9 true positives, the addition of `TimeFeatures` to the feature set has resulted in a reduction of 360 false positives, as shown by the validation test confusion matrix in Figure 4.12. Unfortunately, the false positive rate is still an unmanageably high 0.2%.

Similar to Experiment 4, the packet graph in Figure 4.12 shows that MESONet is predicting nearly all attacks to belong to the "httptunnel" class. This result indicates that, based on this feature set, the "httptunnel" attack is similar to normal network traffic.

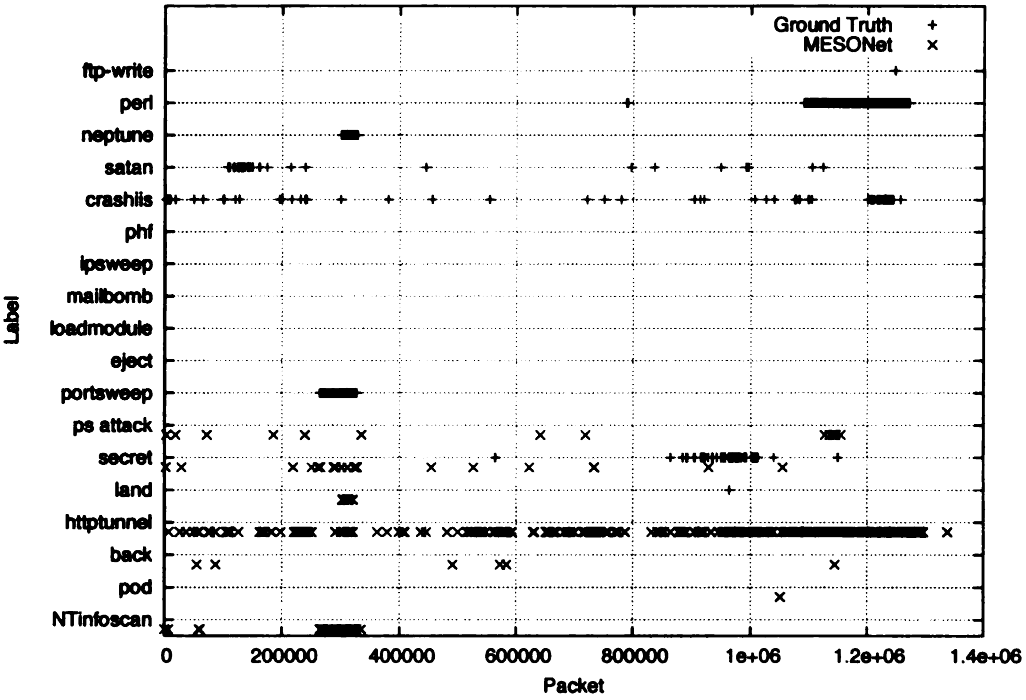
Although this experiment achieved the best overall results for misuse detection, we were concerned with the placement of `TimeFeatures` within the sliding window. Specifically, this experiment includes `TimeFeatures` for both packets appearing in the sliding window, rather than including a single set of `TimeFeatures` (that would have been calculated, in part, from both packets appearing in the window).



	Predicted	
	Normal	Attack
Normal	1256363	2
Attack	1	7177

False Positive Rate	1E-6
Accuracy	.9999
Recall	.9998
Precision	.9997

Figure 4.11: Experiment 5 resubstitution test results: packet graph (top) and confusion matrix (bottom).



	Predicted	
	Normal	Attack
Normal	1311321	2690
Attack	24698	349

False Positive Rate	.0020
Accuracy	.9795
Recall	.0139
Precision	.1148

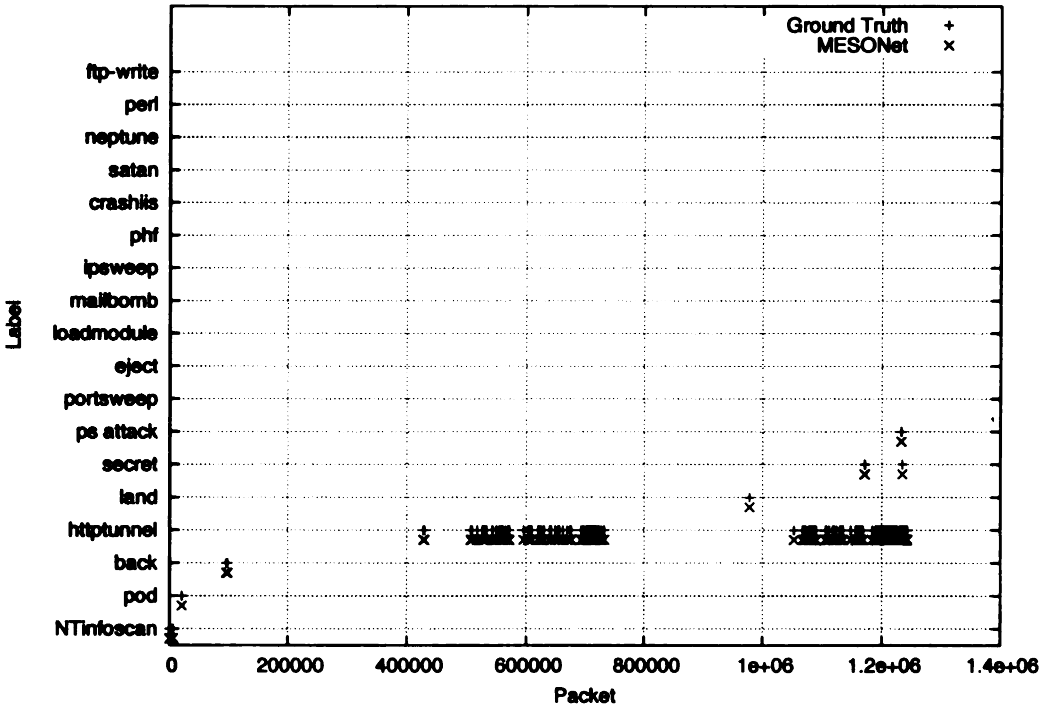
Figure 4.12: Experiment 5 validation test results: packet graph (top) and confusion matrix (bottom).

Experiment 6. The sixth misuse detection experiment performed with MESONet is identical to Experiment 5, with the exception of rearranging the time-based feature with regard to the sliding window. In this experiment we append **`TimeFeatures`** to a two-packet sliding window consisting of **`SinglePacketFeatures`** and **`PayloadCountFeatures`**, rather than including them directly in the sliding window (and hence duplicating them). The rationale for this feature set is to examine the effect of rearranging the feature set.

As shown by the resubstitution test confusion matrix in Figure 4.13, the feature set is clearly sufficient for training. The recall rate of 99.9% and false positive rate of 0.000001% verify that this feature set and training data are sufficient for distinguishing between attack and normal packets.

The validation test confusion matrix in Figure 4.14 shows that this feature set approaches the performance of Experiments 4 and 5. Compared to Experiment 5, the false positive rate is 0.2% greater, while the recall rate is 0.12% less. Compared to Experiment 4, the false positive rate is 0.17% greater, while the recall rate is 0.15% less. The results of this experiment show that time-based features should in fact be including in the sliding window.

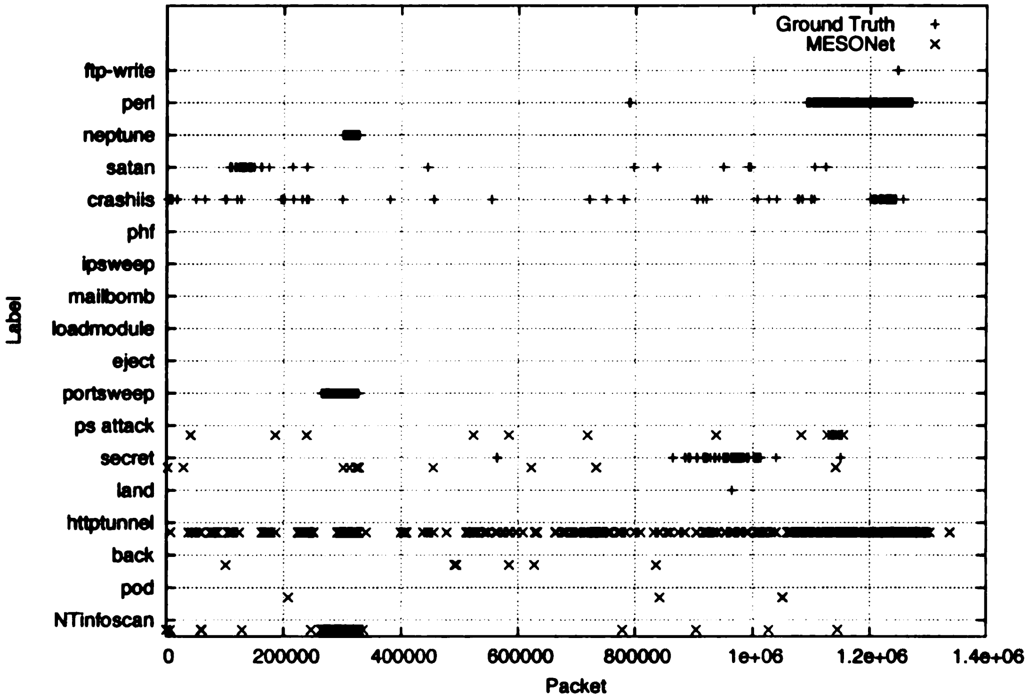
Although this experiment shows that time-based features should be included in the sliding window, Experiments 8 and 9 show that including features in the sliding window is not a generally applicable rule.



	Predicted	
	Normal	Attack
Normal	1256363	2
Attack	1	7177

False Positive Rate	1E-6
Accuracy	.9999
Recall	.9998
Precision	.9997

Figure 4.13: Experiment 6 resubstitution test results: packet graph (top) and confusion matrix (bottom).



	Predicted	
	Normal	Attack
Normal	1308638	5373
Attack	24727	320

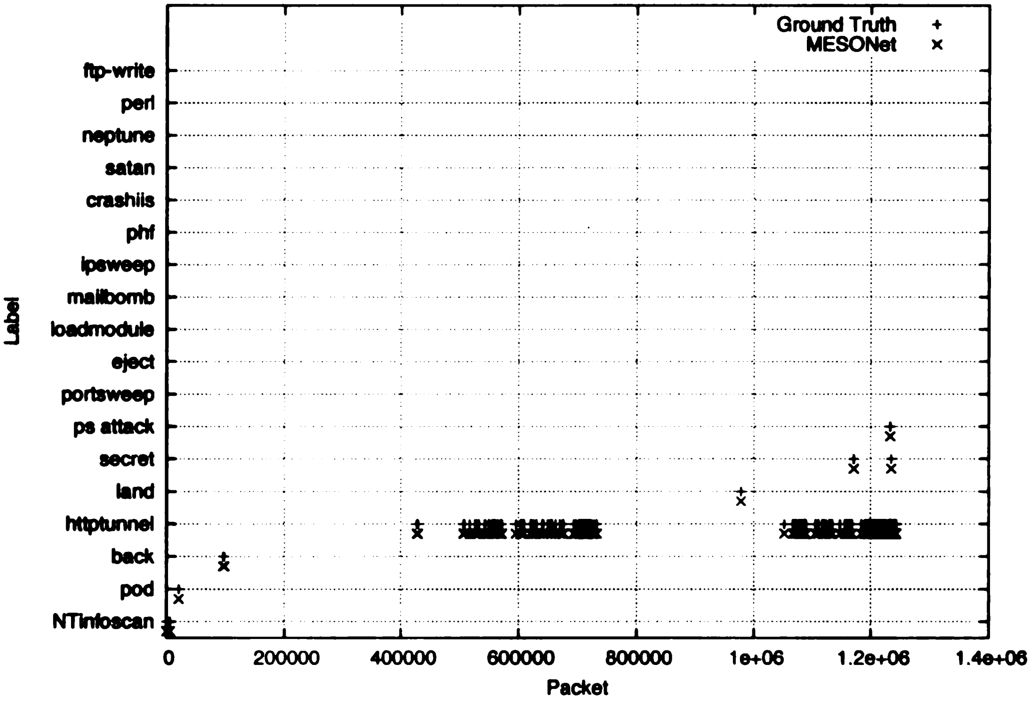
False Positive Rate	.0040
Accuracy	.9775
Recall	.0127
Precision	.0562

Figure 4.14: Experiment 6 validation test results: packet graph (top) and confusion matrix (bottom).

Experiment 7. The seventh misuse detection experiment performed with MESONet explores the effect of larger sliding windows on detection accuracy. Specifically, we use a three-packet sliding window of `SinglePacketFeatures`, `PayloadCountFeatures`, and `TimeFeatures`.

The resubstitution test confusion matrix in Figure 4.15 shows that the feature set and training data are sufficient for distinguishing between attack and normal packets; indeed, the recall and false positive rates are ideal.

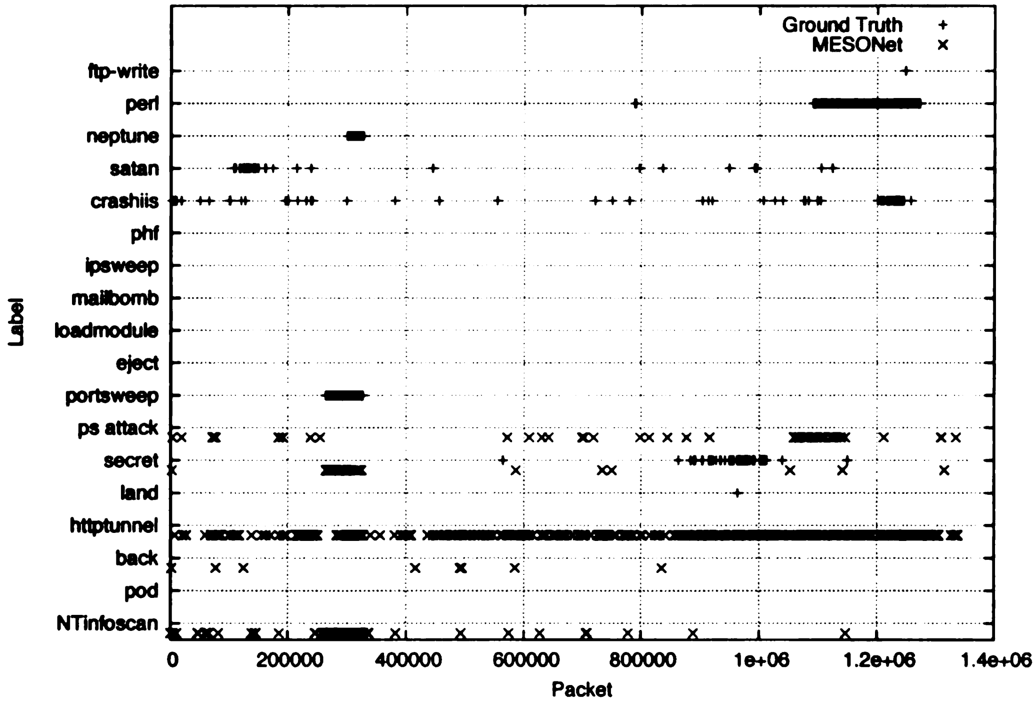
The validation test confusion matrix in Figure 4.16 shows that with this feature set, larger sliding windows have a beneficial effect on false positive rate, while at the same time they have a detrimental effect on recall rate. Compared to Experiment 6, the false positive rate is 0.05% less, while the recall rate is 0.07% less. This experiment shows that although the window size does have an effect upon performance, window size is not a substitute for proper feature selection.



	Predicted	
	Normal	Attack
Normal	1256365	0
Attack	0	7178

False Positive Rate	0.0
Accuracy	1.0
Recall	1.0
Precision	1.0

Figure 4.15: Experiment 7 resubstitution test results: packet graph (top) and confusion matrix (bottom).



	Predicted	
	Normal	Attack
Normal	1309396	4615
Attack	24745	302

False Positive Rate	.0035
Accuracy	.9780
Recall	.0120
Precision	.0614

Figure 4.16: Experiment 7 validation test results: packet graph (top) and confusion matrix (bottom).

4.5 Packet Number Features

In this section we describe experiments based on the use of packet number features. Experiment 8 and 9 use different configurations of **`PacketNumberFeatures`** in conjunction with the feature set from Experiment 5. All feature classes used here are described in Section 3.4. Complete feature set configuration for all experiments conducted in this chapter can be found in Appendix B.

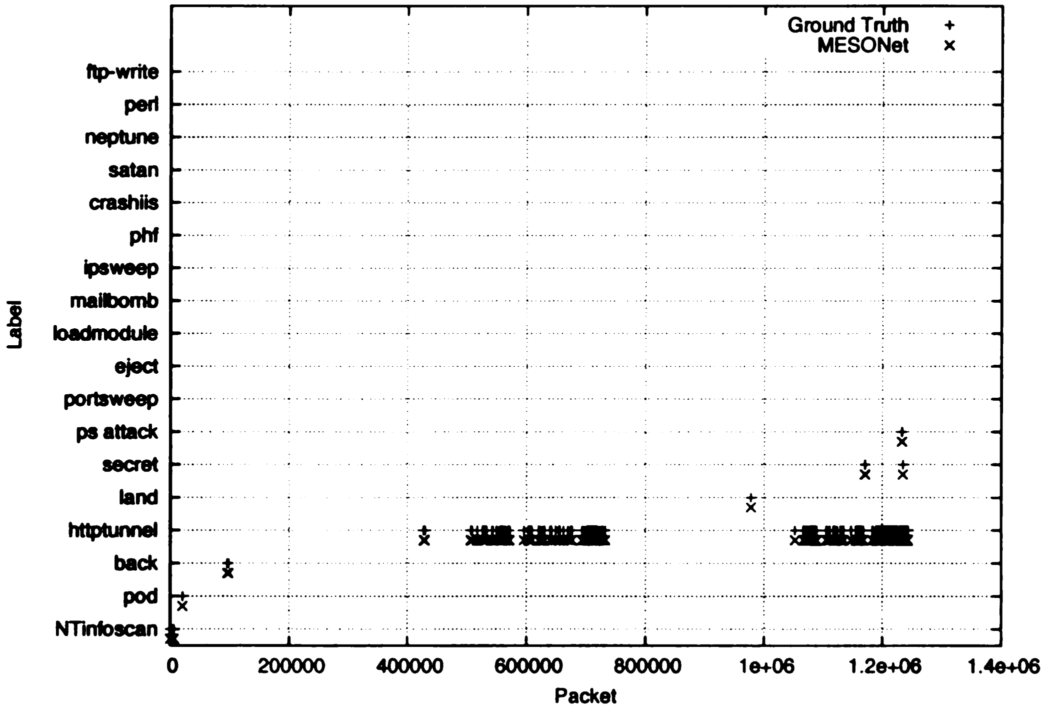
Experiment 8. The eighth misuse detection experiment performed with MESONet adds **`PacketNumberFeatures`** to the feature set from Experiment 5. Specifically, we use **`SlidingWindowFeatures`** to construct feature vectors from the **`SinglePacketFeatures`**, **`PayloadCountFeatures`**, **`TimeFeatures`**, and **`PacketNumberFeatures`** of two packets. **`PacketNumberFeatures`** add features such as the percentage of TCP, UDP, and ICMP traffic. The rationale for including packet number-based features is based on the observation that a number of attacks, specifically port and IP sweeps, use inordinately high numbers of SYN, RST, or FIN packets.

The resubstitution test confusion matrix shown in Figure 4.17 shows that this training data and feature set are capable of distinguishing between attack and normal packets, and adequately describe the characteristics of attack packets.

Unfortunately, the confusion matrix in Figure 4.18 shows that the validation test has been adversely affected by the addition of **`PacketNumberFeatures`**. Indeed, the false positive rate has reached 0.34%, representing 4,546 false alerts, and the recall rate has dropped to 0.19%.

The packet graph in Figure 4.18 shows very little correlation between predicted and actual attack packets, though MESONet does appear to detect the "portsweep" attack (however, mislabeled as "httptunnel").

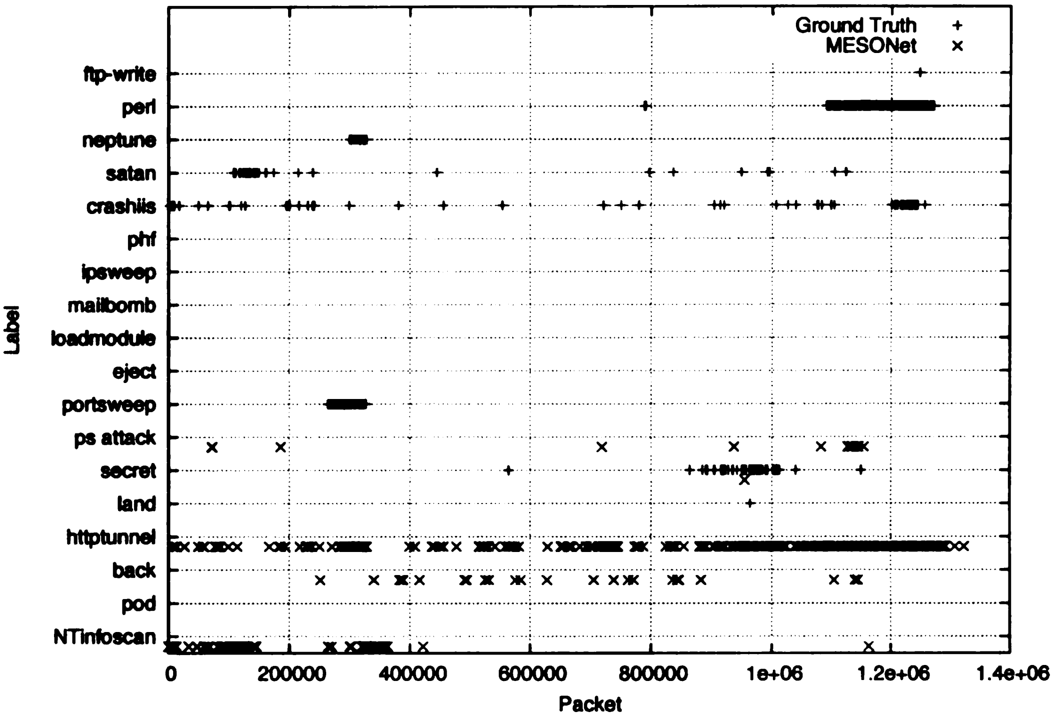
Based on these validation test results, we can conclude that this particular feature set is unsuitable for misuse detection with MESONet, especially when compared to earlier, better performing, experiments.



	Predicted	
	Normal	Attack
Normal	1256365	0
Attack	0	7178

False Positive Rate	0.0
Accuracy	1.0
Recall	1.0
Precision	1.0

Figure 4.17: Experiment 8 resubstitution test results: packet graph (top) and confusion matrix (bottom).



	Predicted	
	Normal	Attack
Normal	1309465	4546
Attack	24998	49

False Positive Rate	.0034
Accuracy	.9779
Recall	.0019
Precision	.0106

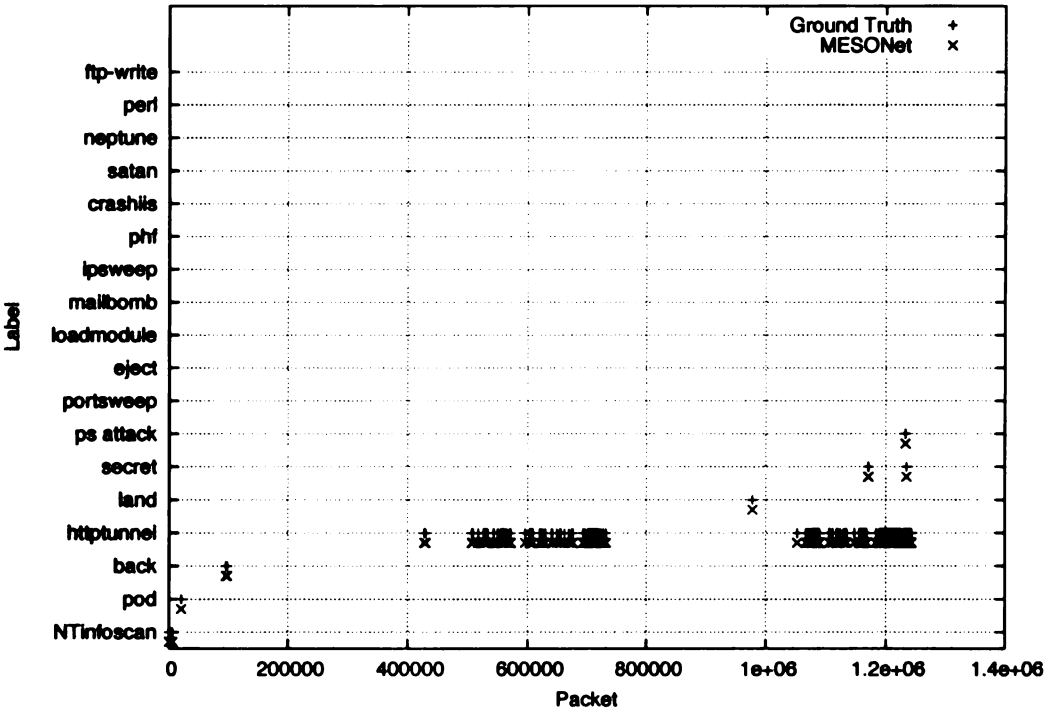
Figure 4.18: Experiment 8 validation test results: packet graph (top) and confusion matrix (bottom).

Experiment 9. The ninth misuse detection experiment performed with MESONet uses the identical feature classes as Experiment 8, however with a different arrangement of features with regard to the sliding window. Specifically, in this experiment we append `TimeFeatures` and `PacketNumberFeatures` to a two-packet sliding window consisting of `SinglePacketFeatures` and `PayloadCountFeatures`, rather than including them directly in the sliding window (and hence duplicating them). The rationale for this rearrangement of features is that time- and number-based features cannot change rapidly over the course of two packets, and Experiment 8 showed that the typical organization of features was insufficient.

As with the resubstitution test in Experiment 8, the confusion matrix in Figure 4.19 is ideal. The recall rate of 100% verifies that the feature set is sufficient to describe characteristics of attacks, and the false positive rate of 0% shows that the feature set and training data are able to distinguish between attack and normal traffic.

Compared to Experiment 8, the validation test confusion matrix in Figure 4.20 adds 33 true positives and reduces the number of false positives by 517. The recall rate of 0.32% has increased marginally, still less than one-quarter of that in Experiment 4, while the false positive rate of 0.3%, representing 4,029 false positives, is 0.07% greater.

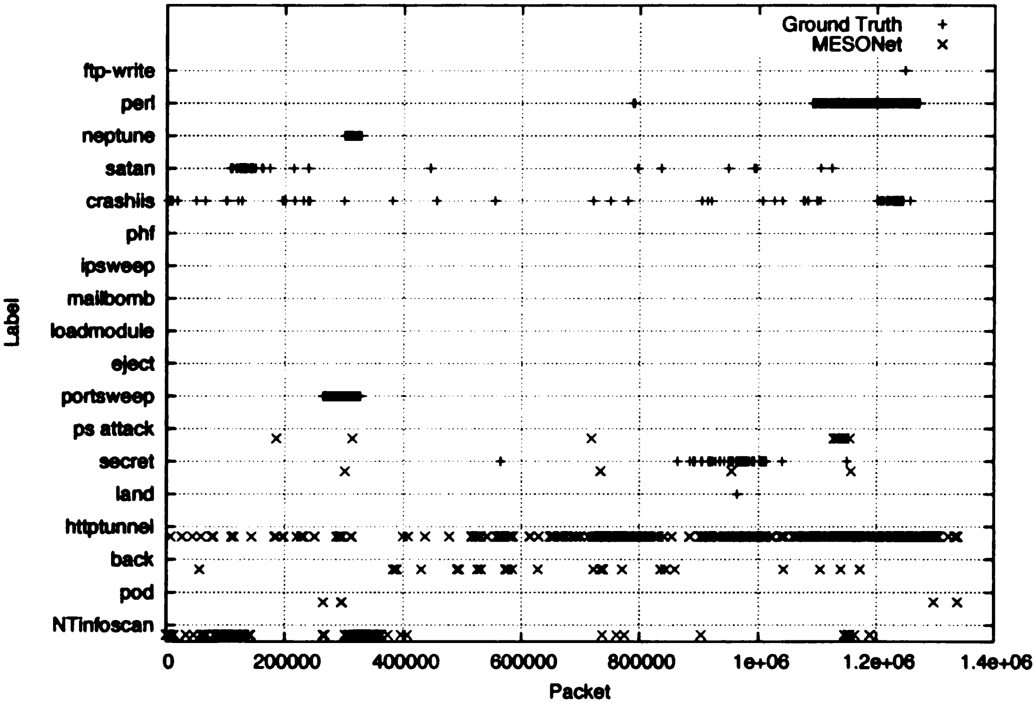
Experiment 9 shows that removing packet number-based features from the sliding window had a positive effect on the validation test, but the ill-effect of including `PacketNumberFeatures` still dominates. Additionally, this experiment, in combination with Experiment 6, show that there is no rule regarding the placement of features with respect to the sliding window.



	Predicted	
	Normal	Attack
Normal	1256365	0
Attack	0	7178

False Positive Rate	0.0
Accuracy	1.0
Recall	1.0
Precision	1.0

Figure 4.19: Experiment 9 resubstitution test results: packet graph (top) and confusion matrix (bottom).



	Predicted	
	Normal	Attack
Normal	1309982	4029
Attack	24965	82

False Positive Rate	.0030
Accuracy	.9783
Recall	.0032
Precision	.0199

Figure 4.20: Experiment 9 validation test results: packet graph (top) and confusion matrix (bottom).

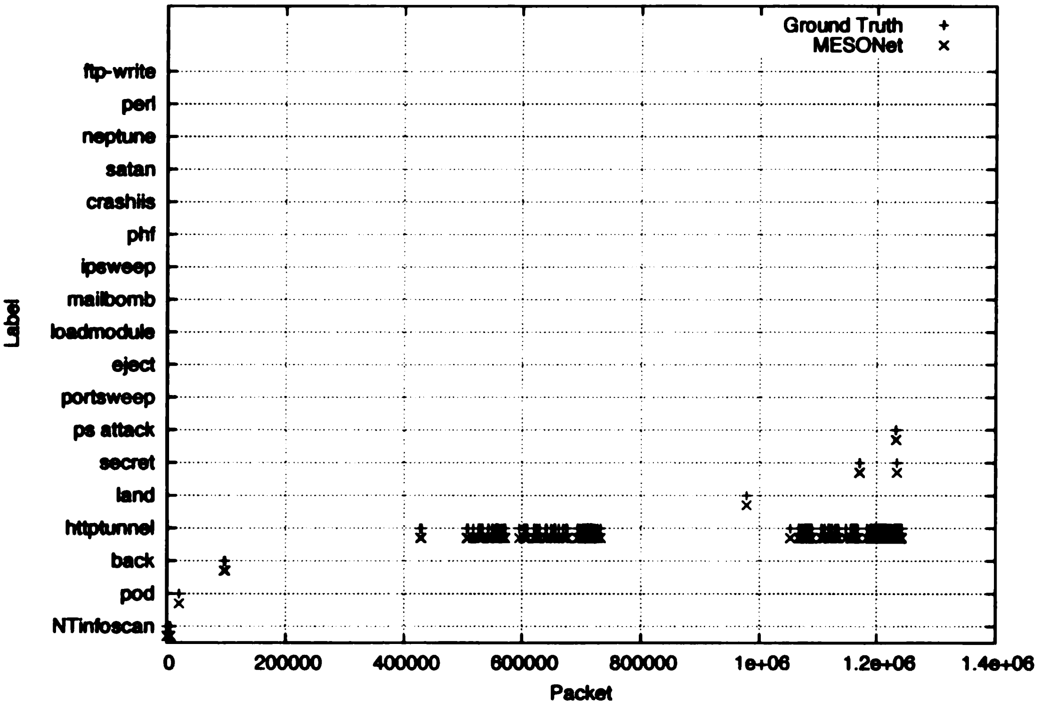
4.6 Destination Grouping Features

In this section we describe experiments based on the use of grouping features, specifically destination grouping. Experiment 10 applies destination grouping, via the `HostFeatures` class, to the feature set from Experiment 9, while Experiment 11 does the same to the feature set from Experiment 5. All feature classes used here are described in Section 3.4. Complete feature set configuration for all experiments conducted in this chapter can be found in Appendix B.

Experiment 10. The tenth misuse detection experiment performed with MESONet introduces destination grouping. Specifically, this experiment uses the same feature set as in Experiment 9, appending `TimeFeatures` and `PacketNumberFeatures` to a two-packet sliding window consisting of `SinglePacketFeatures` and `PayloadCountFeatures`, and additionally applies destination grouping through the use of `HostFeatures`. The rationale for utilizing destination grouping is twofold: First, we hypothesize that destination grouping will further reduce the false positive rate associated with the use of `PacketNumberFeatures`. Second, destination grouping ensures that the features extracted from both packets in the sliding window are related, insofar as they are destined for the same host.

As with Experiment 9, the resubstitution test confusion matrix shown in Figure 4.21 is perfect. The low false positive and high recall rates suggest that the training data and feature set are sufficient for misuse detection.

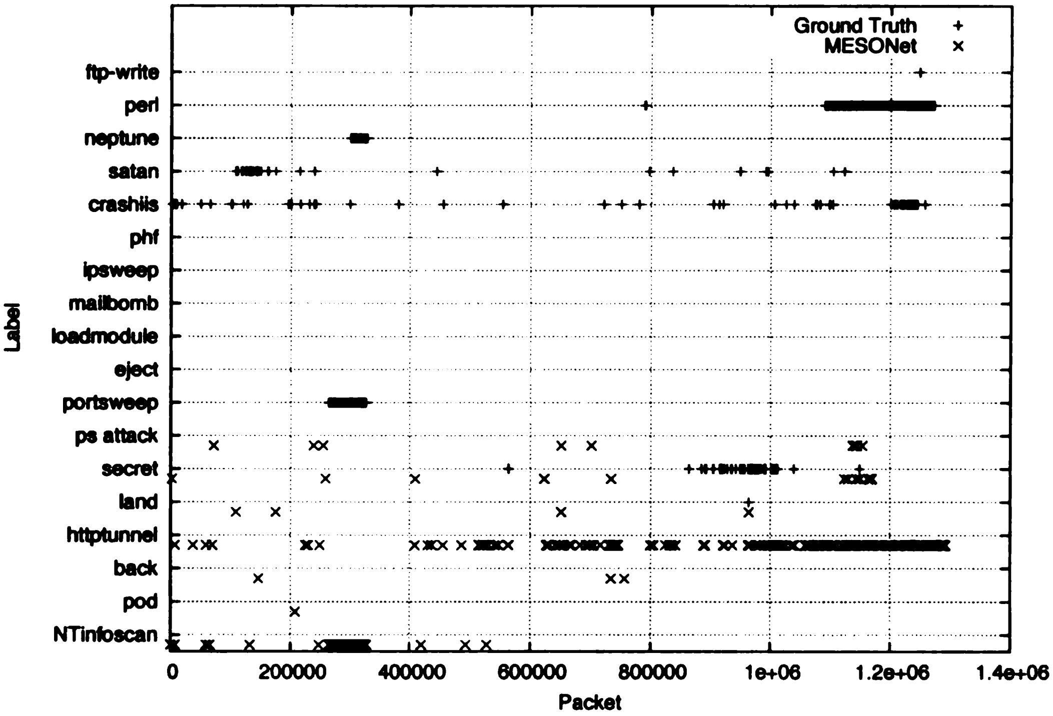
As shown by the confusion matrix in Figure 4.22, the number of true positives has increased slightly over that in Experiment 9 (by 3), however the number of false positives has increased dramatically. In this case, when compared to Experiment 9, the false positive rate has increased to 1.07%, representing 14,101 false alerts. Though this feature set is sufficient for training, grouping by destination is harmful to the correct classification of normal traffic.



	Predicted	
	Normal	Attack
Normal	1256365	0
Attack	0	7178

False Positive Rate	0.0
Accuracy	1.0
Recall	1.0
Precision	1.0

Figure 4.21: Experiment 10 resubstitution test results: packet graph (top) and confusion matrix (bottom).



	Predicted	
	Normal	Attack
Normal	1299910	14101
Attack	24962	85

False Positive Rate	.0107
Accuracy	.9708
Recall	.0033
Precision	.0059

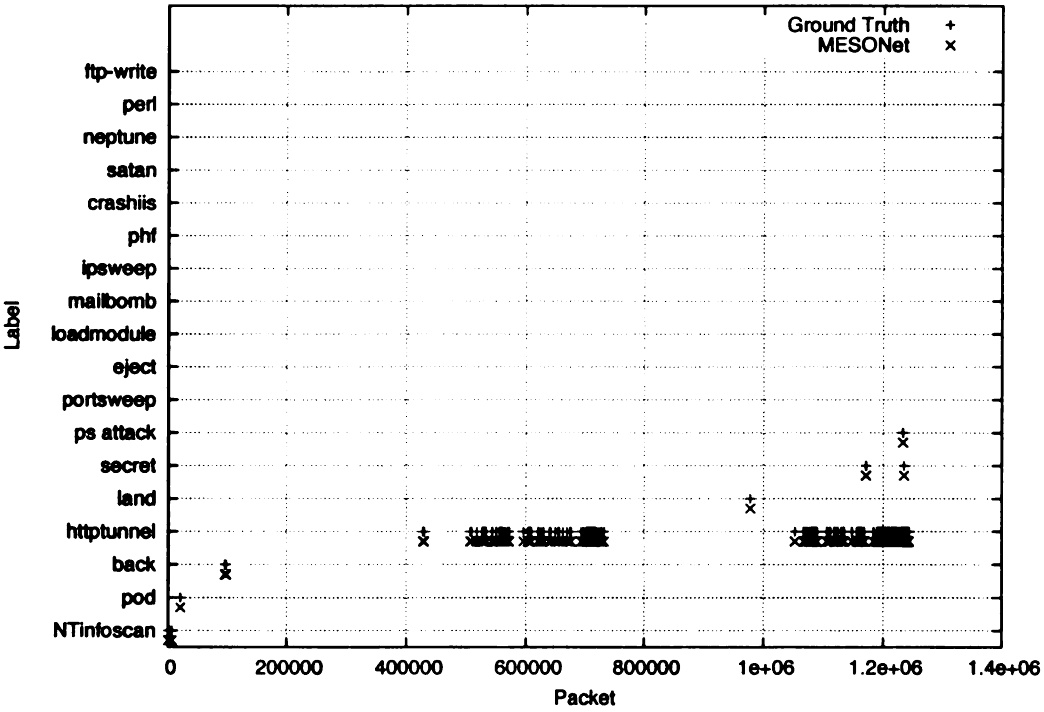
Figure 4.22: Experiment 10 validation test results: packet graph (top) and confusion matrix (bottom).

Experiment 11. The eleventh misuse detection experiment performed with MESONet uses a feature set identical to that in Experiment 10, with the exception of removing the PacketNumberFeatures class. Specifically, Experiment 11 appends TimeFeatures to a two-packet sliding window consisting of SinglePacketFeatures and PayloadCountFeatures, and additionally applies destination grouping through the use of HostFeatures.

The resubstitution test confusion matrix shown in Figure 4.23 shows that this feature set is sufficient for describing the characteristics of normal packets, and for discriminating between normal and attack packets. The low false positive rate of 0.00007% and high recall rate of 99.9% are very nearly ideal.

Compared to Experiment 10, the validation test confusion matrix in Figure 4.24 has 1,686 fewer false positives, and 167 more true positives, corresponding to a false positive rate of 0.94%, and and a recall rate of 1.0%, respectively.

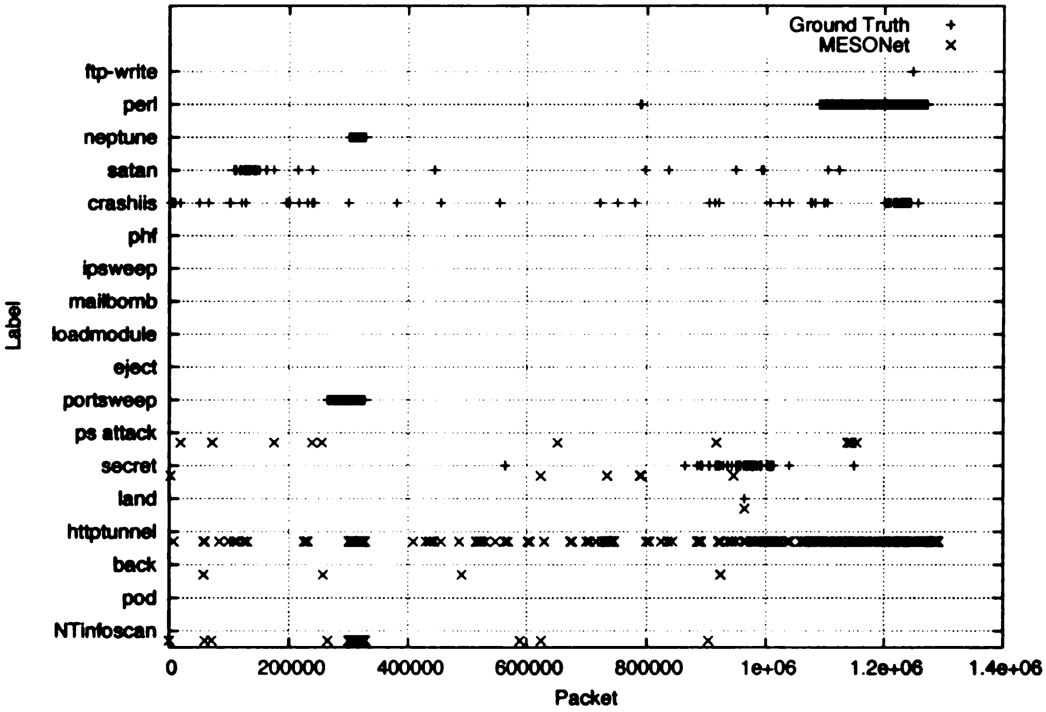
Although the removal of PacketNumberFeatures has clearly had a beneficial effect on performance, the false positive rate is still much higher than that in Experiment 5; 0.94% compared to 0.20%.



	Predicted	
	Normal	Attack
Normal	1256364	1
Attack	7	7171

False Positive Rate	7E-7
Accuracy	.9999
Recall	.9990
Precision	.9998

Figure 4.23: Experiment 11 resubstitution test results: packet graph (top) and confusion matrix (bottom).



	Predicted	
	Normal	Attack
Normal	1301596	12415
Attack	24795	252

False Positive Rate	.0094
Accuracy	.9722
Recall	.0100
Precision	.0198

Figure 4.24: Experiment 11 validation test results: packet graph (top) and confusion matrix (bottom).

4.7 Summary of Results

This chapter has presented the results of 11 different experiments using MESONet for misuse detection. Different feature sets were used for each experiment, and we tested packet features, calculated features, window features, and grouping features.

Table 4.3 summarizes each experiment described in this chapter. It lists the feature classes used and the false positive, accuracy, and recall rates for each validation test. As shown in this table, Experiment 1 achieved the best false positive rate, 0.1%, however it also has the lowest recall rate, 0.09%. Experiment 4 achieved the best recall rate, 1.4%, and has the second-lowest false positive rate, 0.23%.

In conclusion, we should not expect learning-based misuse detection systems to perform well when very little correlation exists between training and testing traffic. More precisely, based on the feature sets used, there is very little similarity between the attacks present in Tracefile A and those in Tracefile B. However, as we shall see in Chapter 5, feature sets used here for misuse detection perform admirably for anomaly detection.

Experiment	False Positive Rate (%)	Recall Rate (%)	Accuracy (%)	Feature Classes
1	0.10	0.09	98.0	SinglePacketFeatures
2	0.37	0.11	97.7	SinglePacketFeatures, PayloadCountFeatures
3	1.37	0.21	96.7	SinglePacketFeatures, PayloadValueFeatures
4	0.23	1.42	97.9	SlidingWindowFeatures, SinglePacketFeatures, PayloadCountFeatures
5	0.20	1.39	97.9	SlidingWindowFeatures, SinglePacketFeatures, PayloadCountFeatures, TimeFeatures
6	0.40	1.27	97.7	SlidingWindowFeatures, SinglePacketFeatures, PayloadCountFeatures, TimeFeatures
7	0.35	1.20	97.8	SlidingWindowFeatures, SinglePacketFeatures, PayloadCountFeatures, TimeFeatures
8	0.34	0.19	97.7	SlidingWindowFeatures, SinglePacketFeatures, PayloadCountFeatures, TimeFeatures, PacketNumberFeatures
9	0.30	0.32	97.8	SlidingWindowFeatures, SinglePacketFeatures, PayloadCountFeatures, TimeFeatures, PacketNumberFeatures
10	1.07	0.33	97.0	HostFeatures, SlidingWindowFeatures, SinglePacketFeatures, PayloadCountFeatures, TimeFeatures, PacketNumberFeatures
11	0.94	1.00	97.2	HostFeatures, SlidingWindowFeatures, SinglePacketFeatures, PayloadCountFeatures, TimeFeatures

Table 4.3: Summary of misuse detection validation tests.

Chapter 5

Anomaly Detection

Anomaly detection systems search network traffic for unusual, or anomalous, traffic. Although anomaly detection systems are unable to reliably detect known attacks, they are capable of detecting zero-day attacks [8,20,21,23], and are thus a critical component of effective network intrusion detection. The remainder of this chapter describes anomaly detection with MESONet, and is organized as follows. Section 5.1 describes the training and testing procedures for anomaly detection with MESONet. Section 5.2 describes the analyses used to evaluate the performance of anomaly detection with MESONet. Sections 5.3 through 5.5 describe experimental results using different feature sets, and Section 5.6 summarizes anomaly detection with MESONet.

5.1 Training and Testing Procedures

Performing anomaly detection with MESONet requires both training and testing phases. However, as stated in [35], anomaly detection tries to learn the characteristics of normal packets and determine if unknown packets do *not* belong to that class. For this reason, when performing anomaly detection, MESONet must

Tracefile	Source	Use	Tracefile Size	Packet Count	Attacks
C	3/8/1999	training	329MB	1256365	0 (filtered)
D	3/29/1999	testing	216MB	1208920	12; 7789 packets

Table 5.1: Tracefiles used for anomaly detection with MESONet.

be trained only with normal traffic. Table 5.1 summarizes the tracefiles used to perform training and testing. Tracefile C, used for training, contains 1.25M normal packets and 0 attack packets (attack packets were filtered out of Tracefile A to construct trcefile C). Tracefile D contains 1.2M normal packets and 7,789 attack packets, comprising 12 different network attacks. It is typical in network intrusion detection for the normal traffic to far outweigh the attack traffic [8].

Figure 5.1, termed a *packet graph*, shows the specific attacks that occur in the testing data (Tracefile D). Each point in Figure 5.1 represents one attack packet (For the sake of clarity, the 1.2M normal packets are not plotted). As shown in this figure, 12 different attacks occur in Tracefile D: smurf, yaga, guesstelnet, snmpget, xsnoop, sshtrojan, ntfsdos, sendmail, ftp-write, portsweep, ps attack, and secret [26]. It should be noted that although Figure 5.1 lists the particular attacks to which packets belong, anomaly detection itself does not distinguish between attacks.

Because an anomaly detection system is trained with only normal traffic, resubstitution tests (such as those performed in Chapter 4) are inappropriate; we therefore only perform a validation test for each different feature set. With the exception of Experiment 1, the proof-of-concept experiment, MESONet is trained on the entirety of Tracefile C and tested on the entirety of Tracefile D.

5.2 Performance Analysis

Performance analysis of anomaly detection with MESONet is significantly different than that of misuse detection. For misuse detection, MESONet labeled every packet as either attack or normal traffic. For anomaly detection, MESONet labels

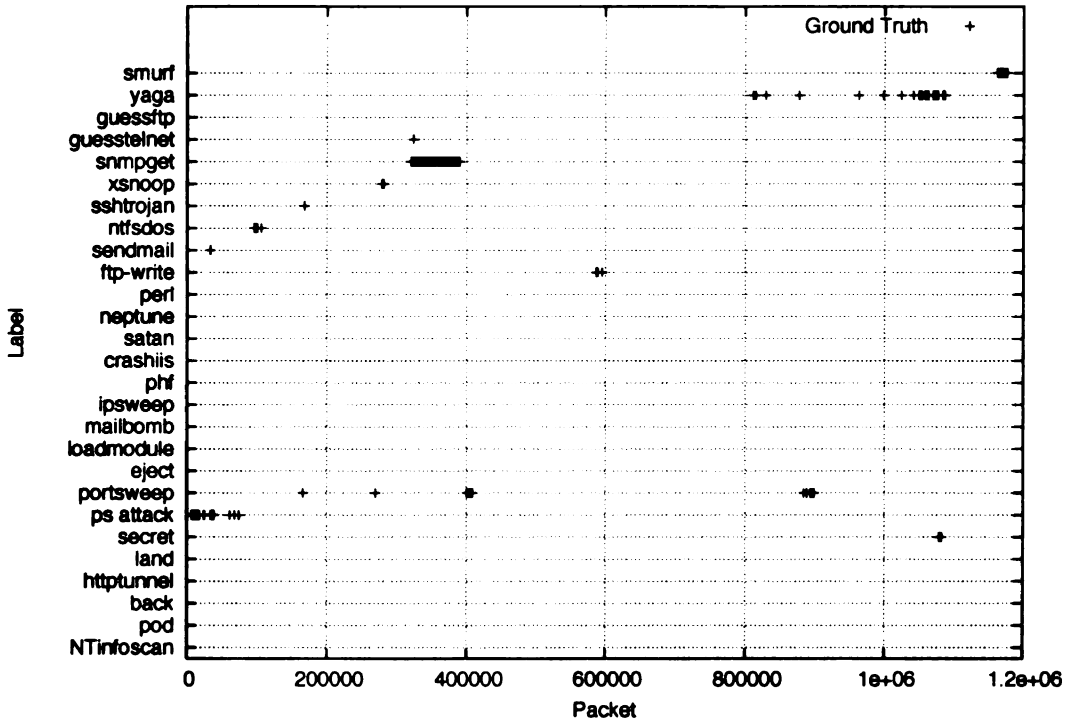


Figure 5.1: Tracefile D ground truth packet graph.

every packet as normal traffic (because it is trained on only normal traffic), *and* returns the nearest-cluster distance for each feature vector with which it is tested.

The IDSEval states that intrusion detection systems should be evaluated based on the number of false-positives occurring during one day [33]. However, as mentioned in [36], this is not an appropriate way to evaluate the performance of an intrusion detection system. In particular, McHugh finds fault with the fact that the IDSEval does not require intrusion detection systems to report a confidence measure with each alert[1]. However, using MESONet we are able to use nearest-cluster distance as a measure of anomaly probability.

Nearest-cluster distance, recalling Section 2.2, is the distance between the tested

[1] To be fair, McHugh also mentions that this may not be possible with some IDSs. To quote: "For example, a system that performs pattern matching either finds a match or it does not."

feature vector and the nearest trained feature vector. As a large nearest-cluster distance signifies a feature vector that was not seen during training, it is by definition an anomaly. Analyzing the performance of anomaly detection with MESONet then focuses on the correlation between large nearest-cluster distances and attack traffic, with the hope that large nearest-cluster distances closely correspond to attack traffic.

Because the analyses performed in Chapter 4 are unsuited for analyzing the correlation between nearest-cluster distances and attack traffic, we use a parametric *Receiver Operating Characteristic* (ROC) curve to analyze the performance of anomaly detection with MESONet.

The goal of ROC analysis is to determine two probabilities: Event+noise, and noise alone [14, 16]. Applied to intrusion detection, a ROC curve is a plot of true-positive detection rate (event+noise) versus false-positive rate (noise alone), where true-positives represent detected attacks and false-positives represent normal packets incorrectly labeled as attacks. Ideally, a ROC curve increases along the true-positive axis very quickly. The strength of this approach is that it allows calculation of the true positive rate given a tolerable false positive rate.

Figure 5.2 is a sample non-parametric ROC curve. Point A is located at .8 along the True Positive Fraction axis and at .4 along the False Positive Fraction axis. If this ROC curve were to belong to an intrusion detection system, it would be correct to state that it would detect 80% of all attacks with a 40% false positive rate.

A parametric ROC curve is constructed based upon the decreasing probability of detection (the nearest-cluster distance, in this case). For MESONet, this means that to construct a parametric ROC curve all tested samples must first be sorted in descending order by their nearest-cluster distance. The sorted list must then be traversed, and for each sample if it is an attack we plot a point higher along the true-positive axis, while if it is normal we plot a point farther right along the false-

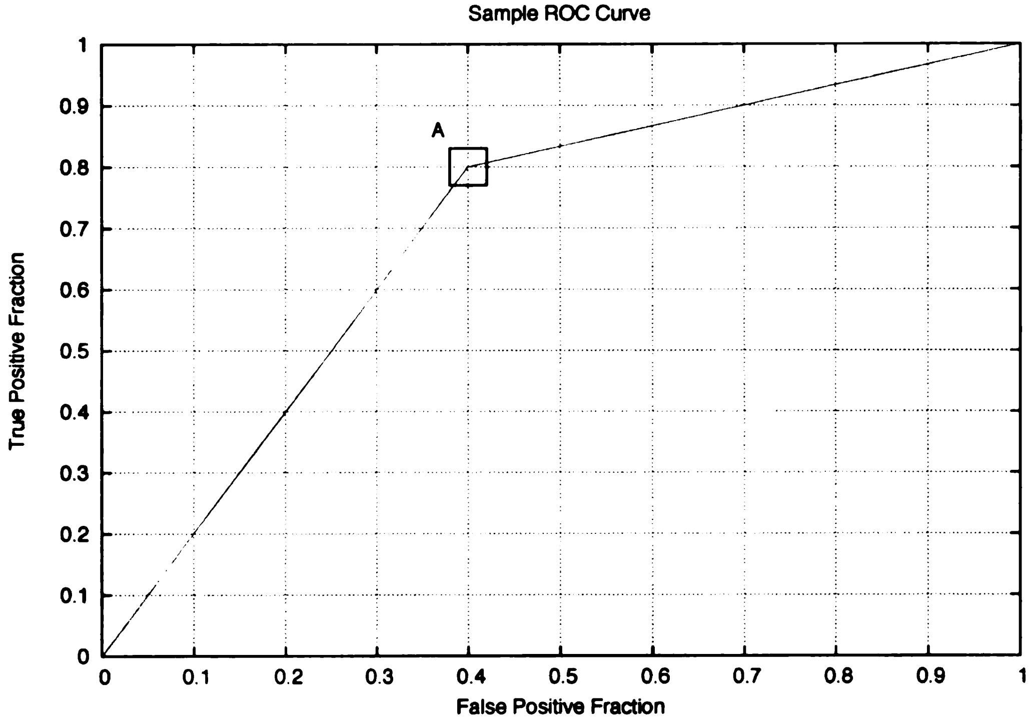


Figure 5.2: Sample ROC curve.

positive axis. In this way the nearest-cluster distance decreases from the origin to the point $(1, 1)$.

In addition to a per-packet ROC curve, where the curve is adjusted based on the classification of each packet, we also construct two additional ROC curves: A per-attack ROC curve, and a time-agglomerative ROC curve.

In a per-attack ROC curve, we change the true-positive axis to represent the fraction of different *attacks* that are detected, rather than the fraction of attack *packets* that are detected. This ROC curve is constructed in response to the realization that MESONet may not detect all attack packets belonging to a given attack, but it may detect *at least one* attack packet belonging to each attack.

In a time-agglomerative ROC curve, we agglomerate per-attack results via a time-based sliding window. When traversing the sorted list of nearest-cluster distances, we also examine packets that arrived at a similar time to the packet currently being examined. If *any* of them is actually an attack packet, then we count it as a detection. For a 1-second sliding window this automatically reduces the detection unit from the number of packets to the number of seconds. Applied to Tracefile D, this reduces the total possible number of detections from 1.2M (the number of packets) to 79,200 (the number of seconds in the 22-hour long tracefile), thereby addressing the base-rate fallacy [8], and keeping the number of false alerts at a manageable level.

5.3 Proof of Concept

Experiment 1. To establish that MESONet is capable of using the nearest-cluster distance for anomaly detection, we performed an m-fold cross-validation test (with $m = 10$), as described in [14], on the first 20,000 packets from tracefile A and recorded the nearest-cluster distance for each tested packet. We then calculated the mean and standard deviation of these distances, and found the mean nearest-cluster distance to be 62.52, and the standard deviation to be 59.5.

We then trained MESONet on the first 20,000 normal packets from tracefile A, tested it on the first 20,000 packets from tracefile B (contains both attack and normal packets), and recorded the resulting nearest-cluster distances. Figure 5.3 is a per-packet plot of the nearest-cluster distances. It shows that the majority of packets have a low nearest-cluster distance, though there are groups of packets that have significantly larger nearest-cluster distances.

Figure 5.4 is a plot of the packet label (attack or normal) and nearest-cluster distance, modulo the standard deviation. It should be noted that the horizontal lines are in fact closely-spaced marks, each of which represents one packet. The

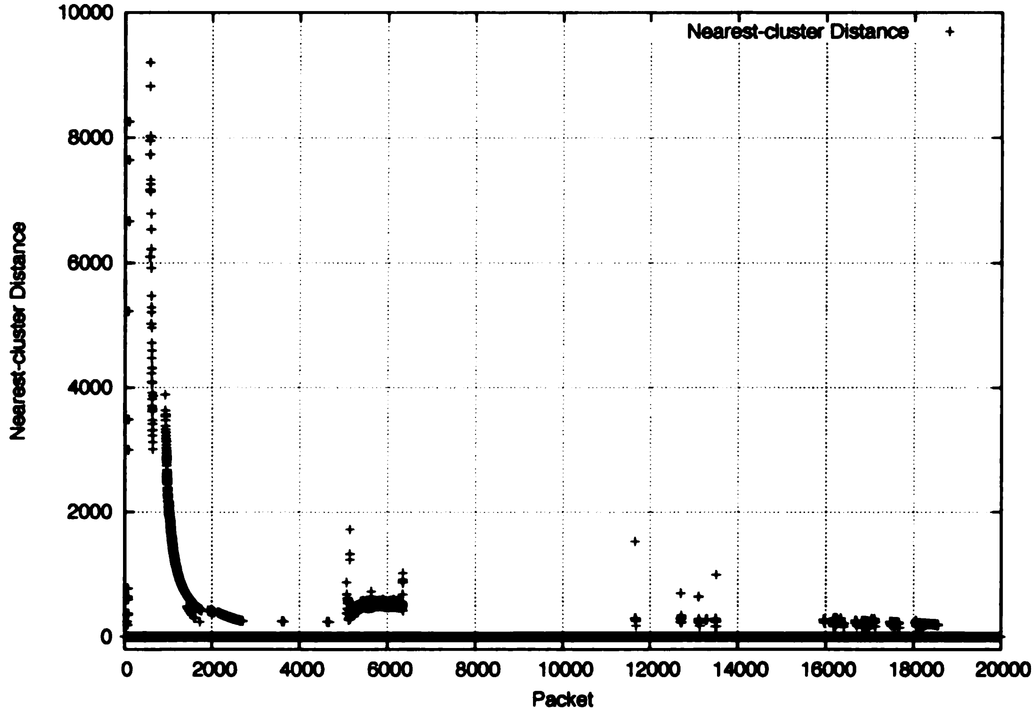


Figure 5.3: Nearest-cluster distances.

"line" appearing at standard deviation 0 near the middle of the figure represents a large amount of traffic having a nearest-cluster distance less than or equal to the mean, while marks above it represent packets with a nearest-cluster distance larger than the mean. The marks (and "lines") in the packet label portion of the figure represent known labels (based on the IDSEval detection truth), and are marked as either "Normal" or "Attack."

Figure 5.4 shows that all attacks contained in the tested traffic are identified as having nearest-cluster distances greater than two standard deviations from the mean. Although a number false positives did occur, a nearest-cluster distance of greater than two standard deviations from the mean clearly separates the majority of attack and normal traffic.

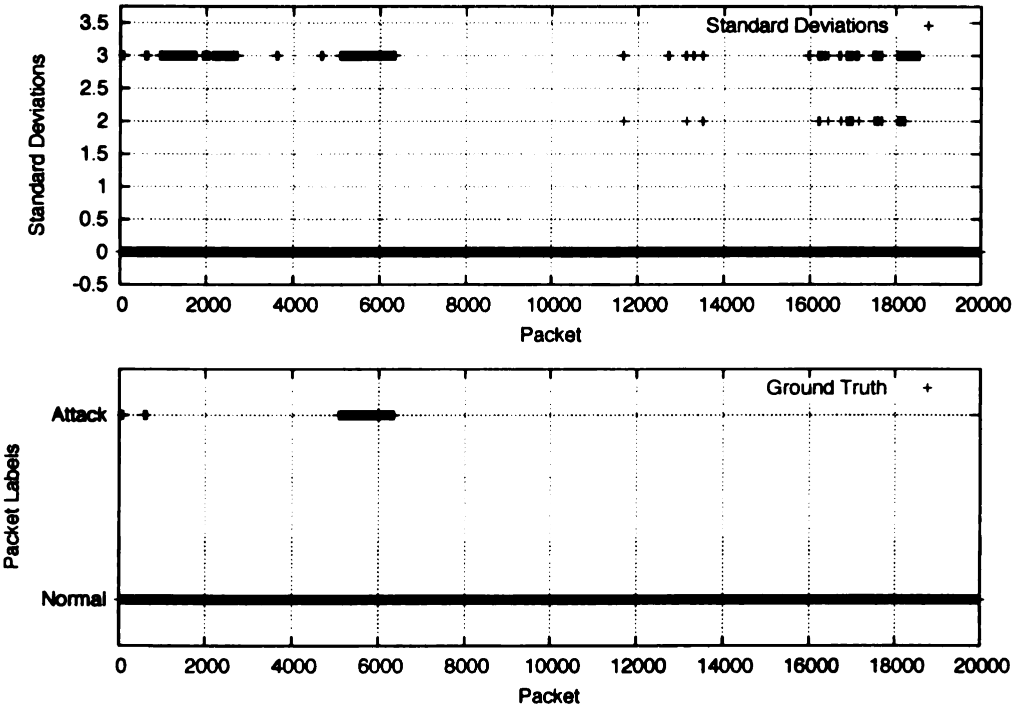


Figure 5.4: Nearest-cluster standard-deviations(top) and packet labels (bottom).

This experiment suggests that nearest-cluster distance may be an accurate measure of network anomalies, and subsequent experiments establish the correlation between attack packets and large nearest-cluster distance.

5.4 Feature Sets from Misuse Detection

In this section we describe experiments based on the best performing feature sets from the misuse detection experiments of Chapter 4. If any of these feature sets also perform well for anomaly detection, then it will show that the features used here are significant for both misuse and anomaly detection. Complete feature set configuration for all experiments conducted in this chapter can be found in Appendix C.

Experiment 2. The second anomaly detection experiment performed with MESONet used the same feature set as Experiment 6 in Section 4.4. For this experiment, we append `TimeFeatures` to a two-packet sliding window consisting of `SinglePacketFeatures` and `PayloadCountFeatures`. This experiment is the first of three using the best performing feature sets from Chapter 4.

Figure 5.5 is the per-packet ROC curve resulting from using this feature set for anomaly detection. As shown in this figure, initial performance quickly reaches a 72% true positive rate, with a 15% false positive rate. Performance then slowly increases to a 90% true positive rate, with a 60% false positive rate. As shown by Figure 5.5, and confirmed by upcoming experiments, per-packet ROC analysis is, at best, a losing battle; the false positive rates are much too high. Indeed, even a 10% false positive rate results in 120,000 false alerts.

Figure 5.6 shows more clearly that the feature set used for this experiment captures the characteristics of all attacks. Indeed, 100% of attacks are detected with a false positive rate of 28%. However, even though all attacks are detected, the 28% false positive rate results in 336,000 false alerts.

Figure 5.7 shows that there is a time-based correlation between large nearest-cluster distances and attack packets. When using a 1-second window, this feature set detects all attacks with a false positive rate of 16.5%. However, when utilizing

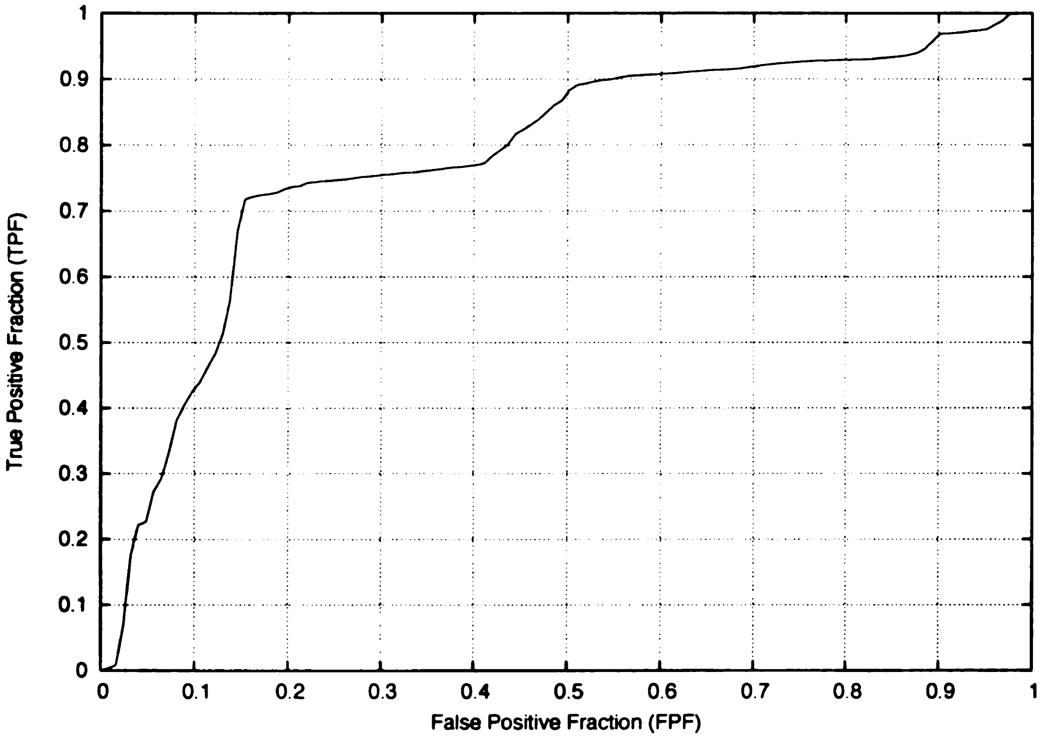


Figure 5.5: Per-packet ROC curve for Experiment 2.

a 60-second window, this feature set detects all attacks with a false positive rate of 0.5%, corresponding to 396 false alerts throughout the 22-hour tracefile – The first practical level of false alerts achieved by MESONet. Subsequent experiments attempt to lower the false positive rate even further, while at the same time reducing the size of the sliding window.

This experiment establishes the trend, born out by subsequent experiments, that per-packet and per-attack ROC analyses are plagued by a high false positive rate. This trend is not surprising, especially when considering research showing that network traffic is inherently "strange" [17, 40]. However, subsequent experiments also confirm that time-based ROC analysis, even when using the small 1-second window, are effective at detecting network attacks with a low false positive rate.

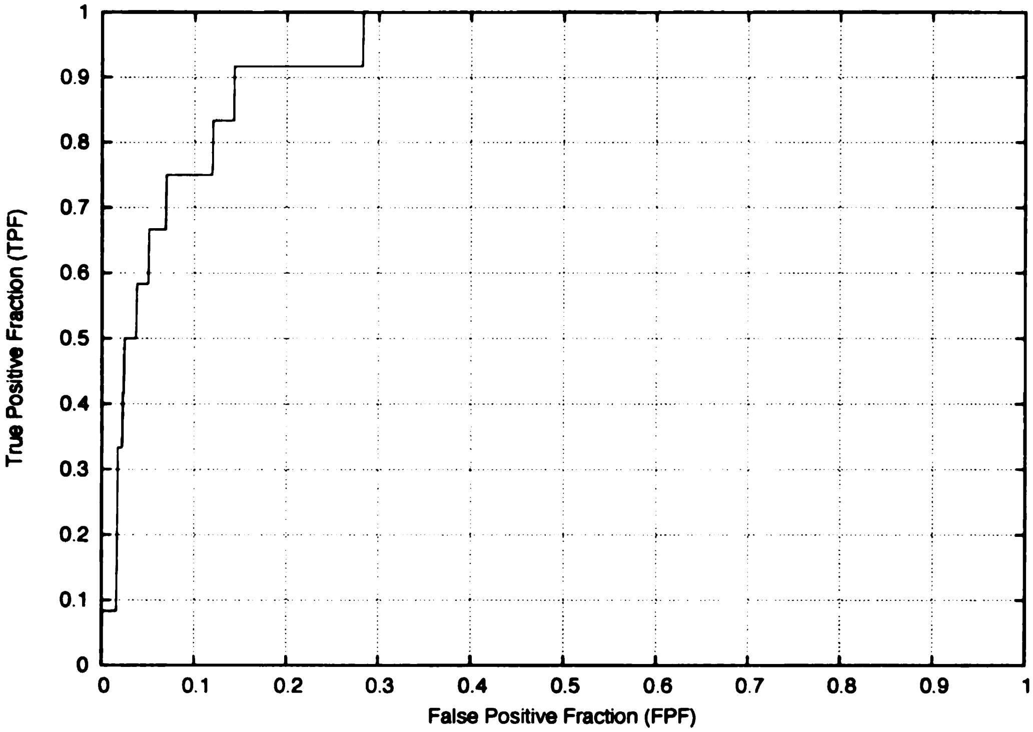


Figure 5.6: Per-attack ROC curve for Experiment 2.

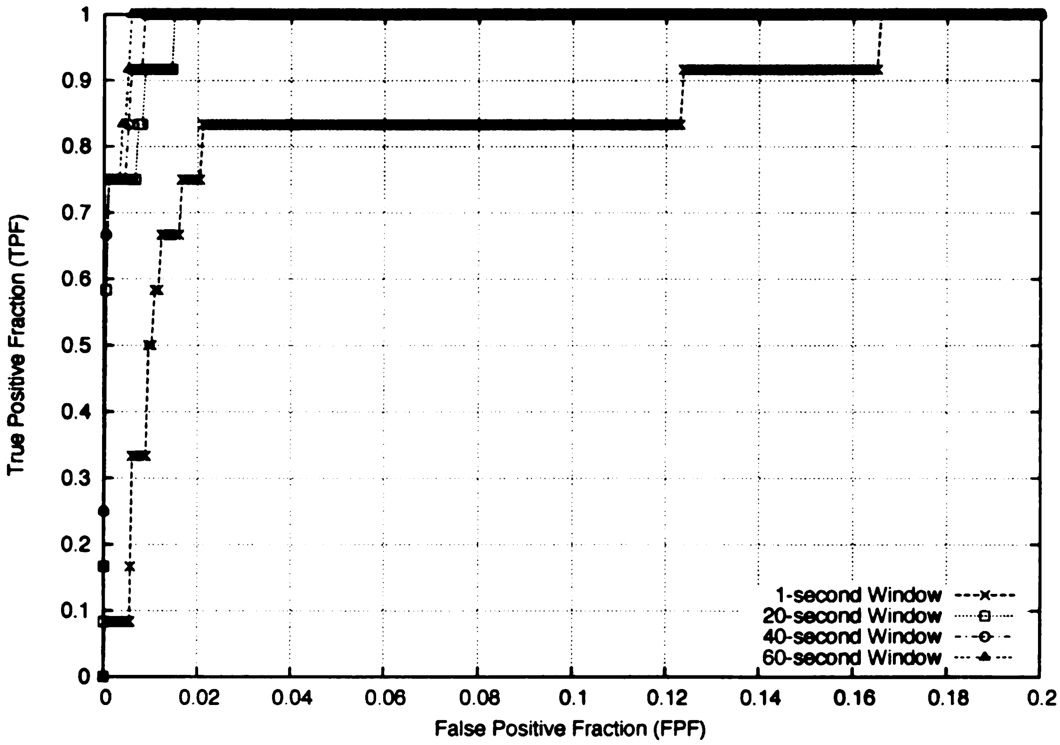


Figure 5.7: Time-agglomerative ROC curve for Experiment 2.

Experiment 3. The third anomaly detection experiment performed with MESONet used the same feature set as Experiment 5 in Section 4.4. For this experiment, we use `SlidingWindowFeatures` to construct feature vectors from the `SinglePacketFeatures`, `PayloadCountFeatures`, and `TimeFeatures` of two packets. The difference between this experiment and Experiment 2 is that here we include `TimeFeatures` in the sliding window.

Figure 5.8 is the per-packet ROC curve resulting from using this feature set for anomaly detection. Compared to Figure 5.5, this ROC curve is more gradual, increasing to a 70% true positive rate, with a 20% false positive rate, followed by a 90% true positive rate, with a 60% false positive rate.

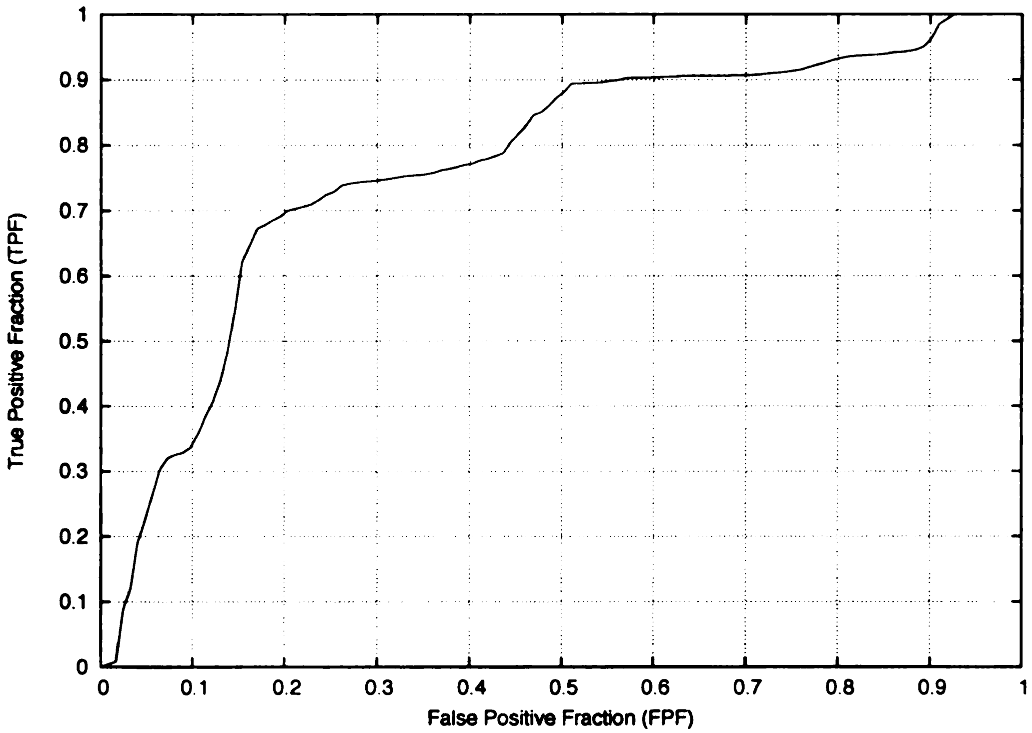


Figure 5.8: Per-packet ROC curve for Experiment 3.

Compared to Figure 5.6, the per-attack ROC curve in Figure 5.9 increases more quickly, detecting 100% of attacks with a false positive rate of 18%.

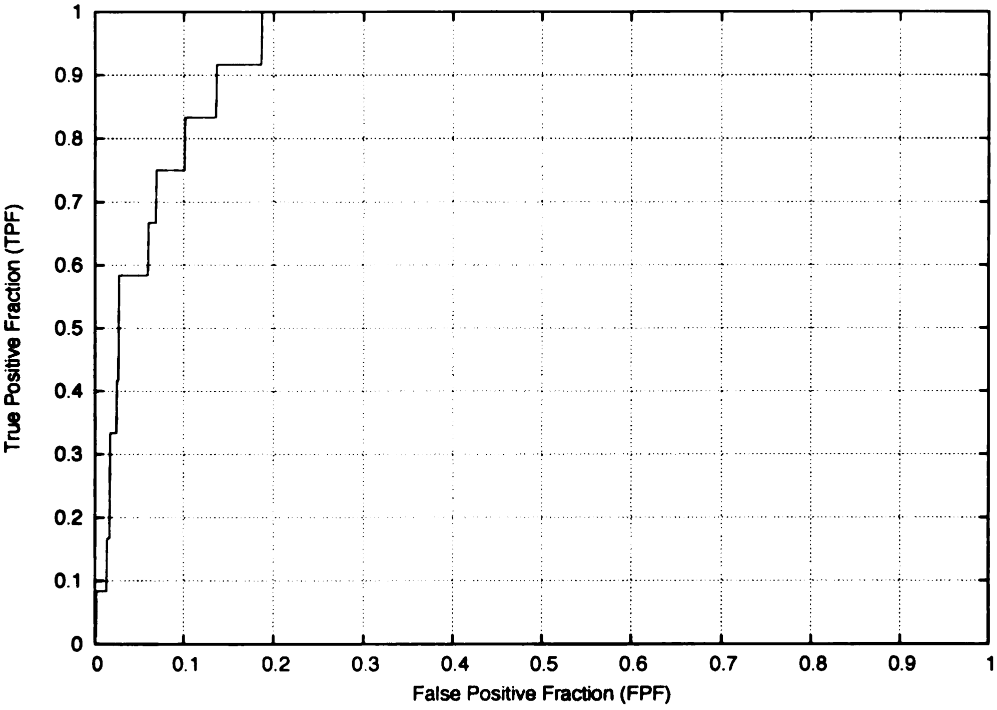


Figure 5.9: Per-attack ROC curve for Experiment 3.

Figure 5.10 shows an improvement in the performance of the 1-second window when compared to Experiment 2, achieving a 100% true positive rate, with an 11.8% false positive rate. However, performance of the 20-second window is slightly worse, detecting 100% of attacks at a 2% false positive rate as opposed to a 1.5% false positive rate. When utilizing a 60-second window, this feature set detects all attacks with a false positive rate of 0.7%, corresponding to 554 false alerts.

As in Experiment 2, the true positive rates achieved by the per-packet and per-attack ROC analyses are crippled by a high false positive rate, just as time-based ROC analysis is successful at achieving a high true positive rate with few false alerts.

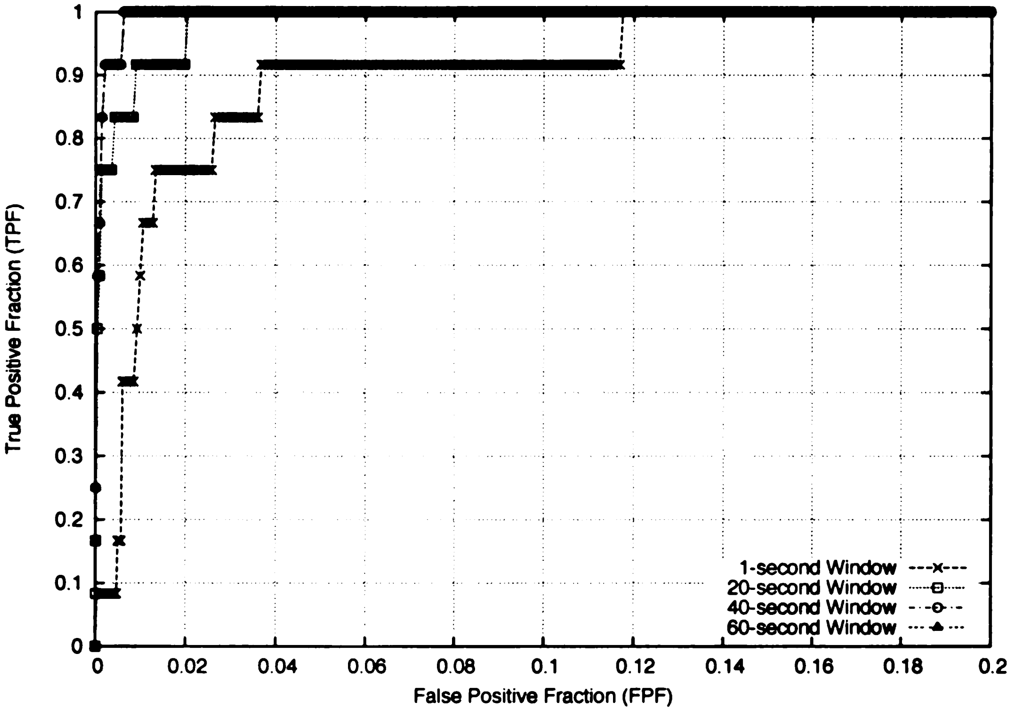


Figure 5.10: Time-agglomerative ROC curve for Experiment 3.

Experiment 4. The fourth anomaly detection experiment performed with MESONet used the same feature set as Experiment 4 in Section 4.4. For this experiment, we use SlidingWindowFeatures to construct feature vectors from the SinglePacketFeatures and PayloadCountFeatures of two packets.

Figure 5.11 is the per-packet ROC curve resulting from using this feature set for anomaly detection. This ROC curve is very similar to that in Figure 5.8, achieving a roughly equivalent 69% true positive rate, with a 20% false positive rate, followed by a 91% true positive rate, with a 60% false positive rate.

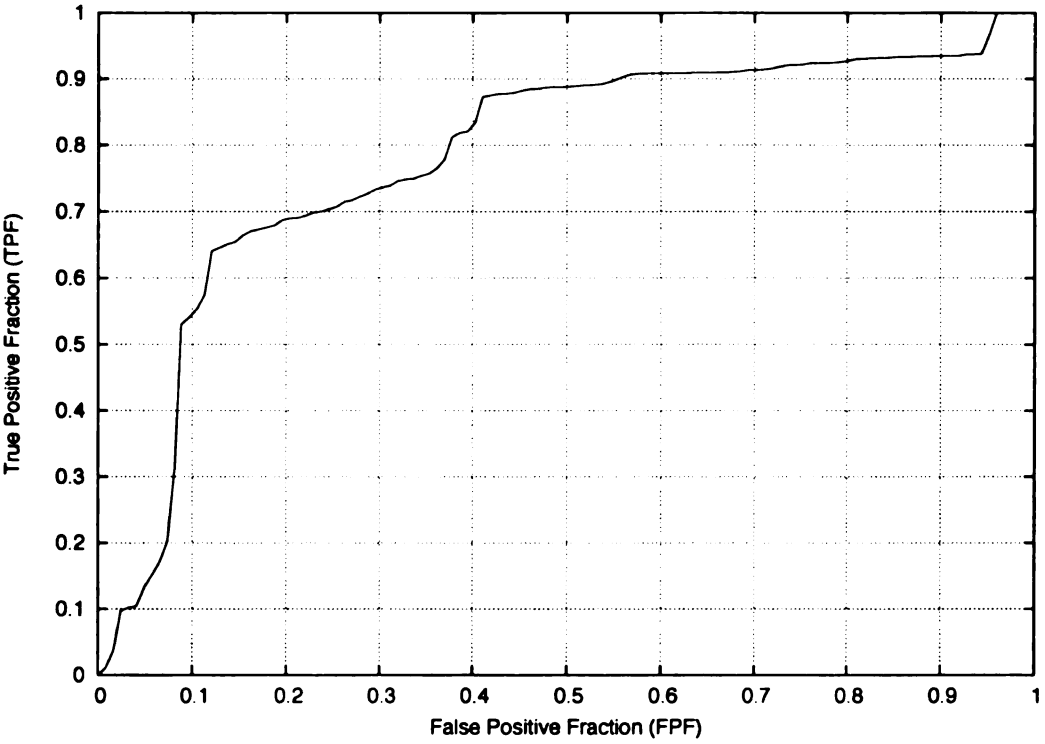


Figure 5.11: Per-packet ROC curve for Experiment 4.

The per-attack ROC curve in Figure 5.12 also achieves a 100% true positive rate, with an 18% false positive rate, similar to Figure 5.9. However, this ROC curve achieves a 58% true positive rate at a false positive rate of 1%, compared to the 2.5% from Figure 5.9.

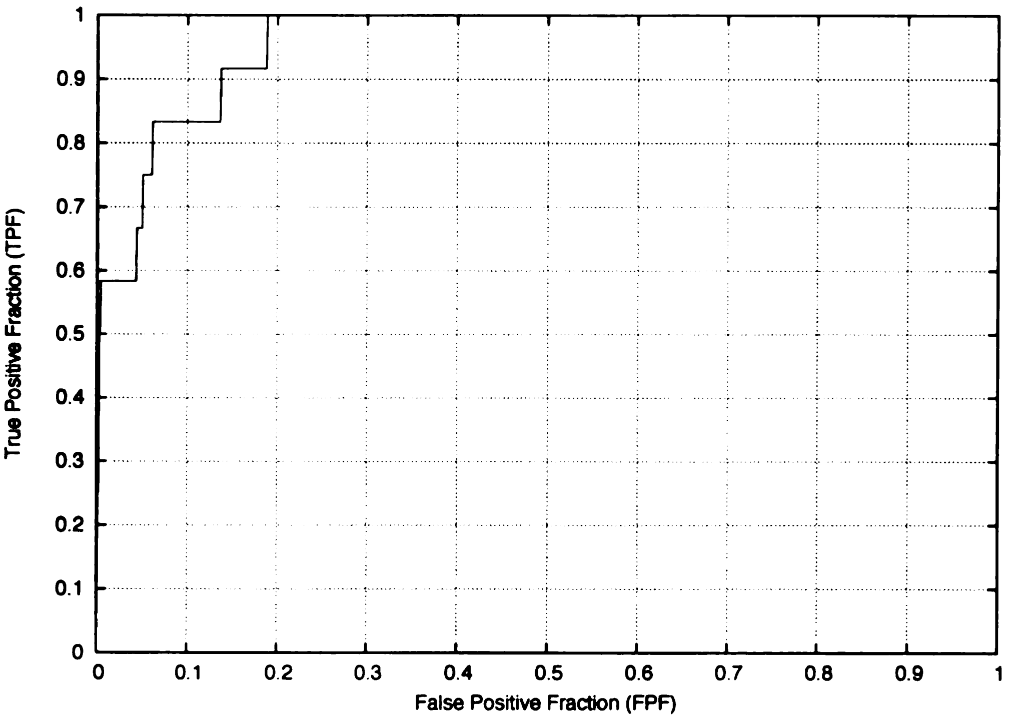


Figure 5.12: Per-attack ROC curve for Experiment 4.

Figure 5.13 shows that this feature set is capable of detecting 100% of attacks within a 1-second window, at a false positive rate of 14%, slightly worse than that in Experiment 3. However, when using a 40-second or 60-second window, 100% of attacks are detected with the low false positive rate of 0.2%, corresponding to 158 false alerts. This is the best performing time-agglomerative ROC analysis conducted for anomaly detection, and reaches an acceptable number of false alerts.

This experiment is the last using a misuse detection feature set. All three have shown that per-packet and per-attack ROC analyses are ineffectual, while even a 1-second time-agglomerative ROC analysis performs with an acceptable false positive rate when detecting 100% of network attacks. Overall, larger time windows during

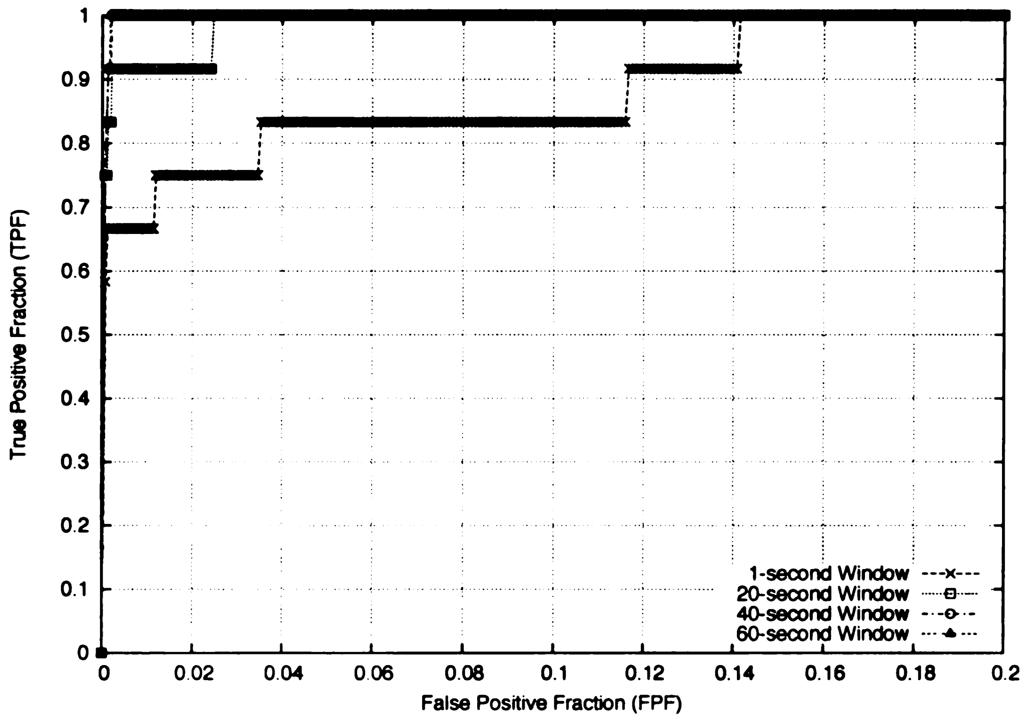


Figure 5.13: Time-agglomerative ROC curve for Experiment 4.

time-agglomerative ROC analysis have resulted in higher true positive and lower false positive rates. This suggests that there is a time-based relationship between network anomalies and network attacks, specifically when using the best performing feature sets from misuse detection.

5.5 Derivative Features

In this section we describe experiments using feature sets that include grouping, derivatives, packet number, time, single packet features, and in one case, payload count features. Complete feature set configuration for all experiments conducted in this chapter can be found in Appendix C.

Experiment 5. The fifth anomaly detection experiment performed with MESONet used a different feature set from those used for misuse detection in Chapter 4. For this experiment, we use a destination-grouped two-packet sliding window of `SinglePacketFeatures` and `PayloadCountFeatures`, appended to `PacketNumberFeatures` and the derivative of `TimeFeatures`. The main difference between this experiment and others is the inclusion of derivative features (through the use of the `DerivativeFeatures` class) and `PacketNumberFeatures`. The rationale for including these features is that some network attacks have unusual timing characteristics (for example, DoS attacks), or send unusually high numbers of certain packets over a short period of time (for example, portsweeps) [26].

Overall, the per-packet ROC curve for this feature set, shown in Figure 5.15 is only marginally better than that in Experiments 2 through 4, achieving a 90% true positive rate, with a 50% false positive rate. However, an interesting characteristic of this ROC curve is that it reaches a 67% true positive rate very quickly, with only a 5% false positive rate. This characteristic shows that feature vectors with large nearest-cluster distances are very likely to be attacks, but that middling nearest-cluster distances are not an accurate measure of attacks.

Similar to the per-packet ROC curve, the per-attack ROC curve shown in Figure 5.15 also reaches a 100% true positive rate quickly, with only an 8% false positive rate. Figure 5.15 shows that when using this feature set, large nearest-cluster distances cover all network attacks.

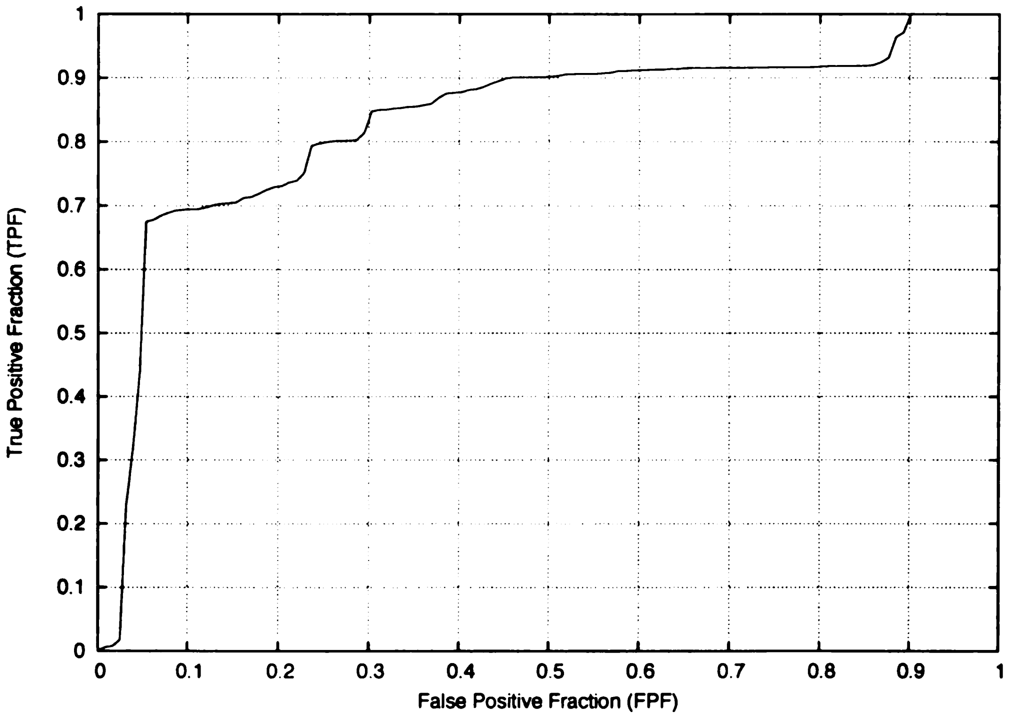


Figure 5.14: Per-packet ROC curve for Experiment 5.

With a 1-second window, the time-agglomerative ROC curve shown in Figure 5.16 reaches a 100% true positive rate with a 3% false positive rate. Larger time windows also achieve a 100% true positive rate, however with a 6% false positive rate.

As with both the per-packet and per-attack ROC curves, the time-agglomerative curve achieves a high true positive rate faster than experiments from Section 5.4. However, for all time-agglomerative ROC curves, Experiment 2 through 4 achieve the same true positive rate (100%) at a lower false positive rate.

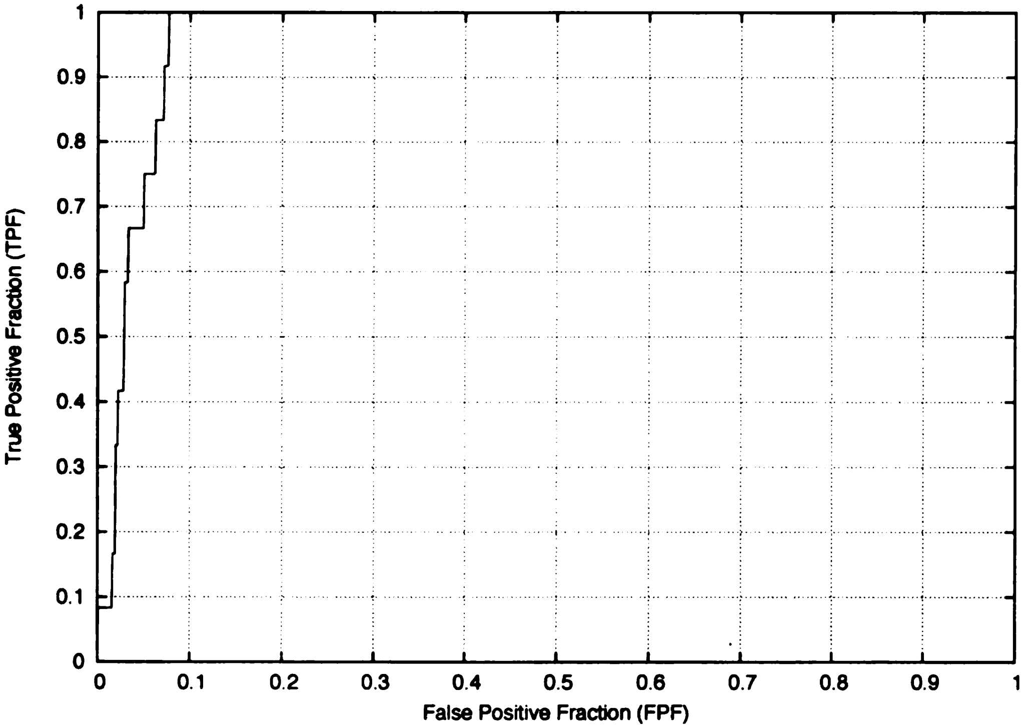


Figure 5.15: Per-attack ROC curve for Experiment 5.

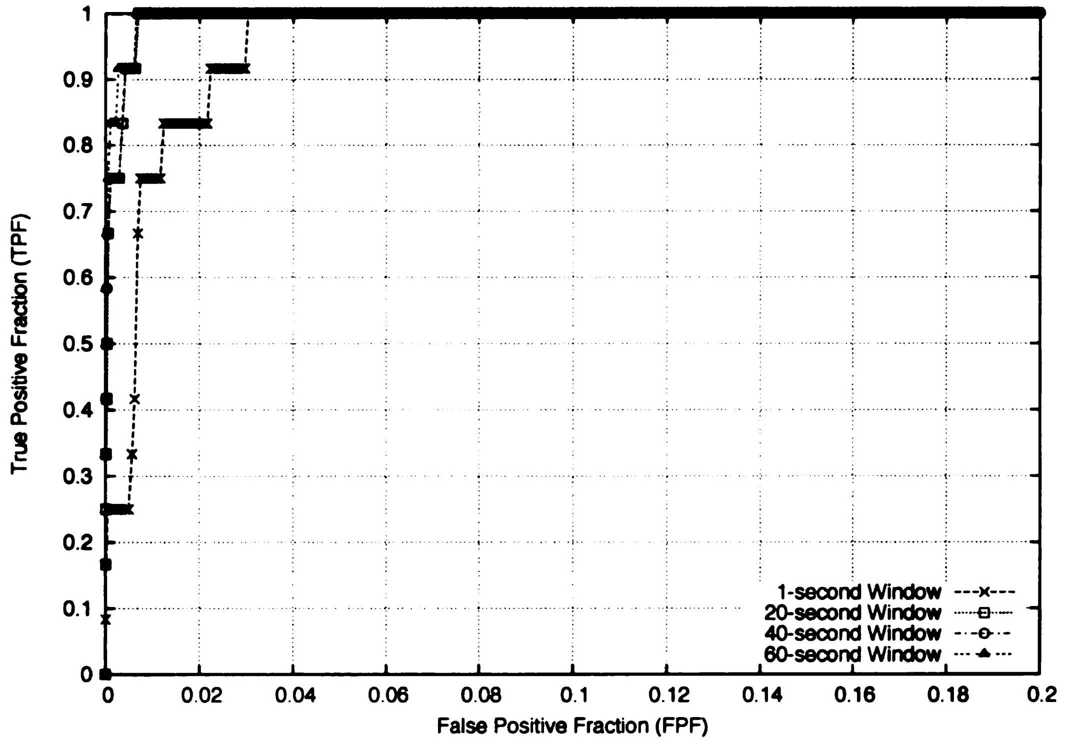


Figure 5.16: Time-agglomerative ROC curve for Experiment 5.

Experiment 6. The sixth anomaly detection experiment performed with MESONet removes the `PayloadCountFeatures` from the feature set of Experiment 5. For this experiment, we use a destination-grouped two-packet sliding window of `SinglePacketFeatures`, appended to `PacketNumberFeatures` and the derivative of `TimeFeatures`. The rationale for this experiment is that payload is the most variable aspect of network traffic, and hence is most likely to generate false positives.

The per-packet ROC curve for this feature set, shown in Figure 5.17 is nearly identical to that in Experiment 5, achieving a 70% true positive rate, with a 10% false positive rate, increasing to a 90% true positive rate, with a 50% false positive rate.

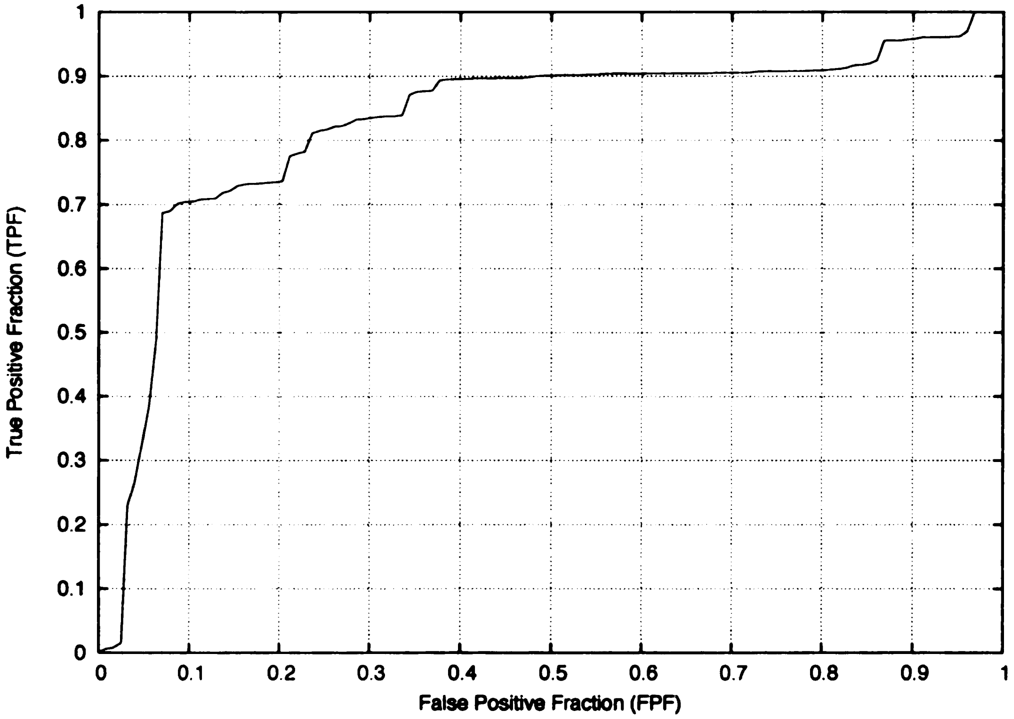


Figure 5.17: Per-packet ROC curve for Experiment 6.

Although the per-attack ROC curve for this feature set, shown in Figure 5.18, also achieves a 100% true positive rate, the false positive rate of 12% is slightly higher than that in Experiment 5. Figure 5.18 also shows that this feature set is "slower" at detecting all the attacks present in Tracefile D, which suggests that the feature set used in Experiment 5 is better at correlating large nearest-cluster distances to attack packets.

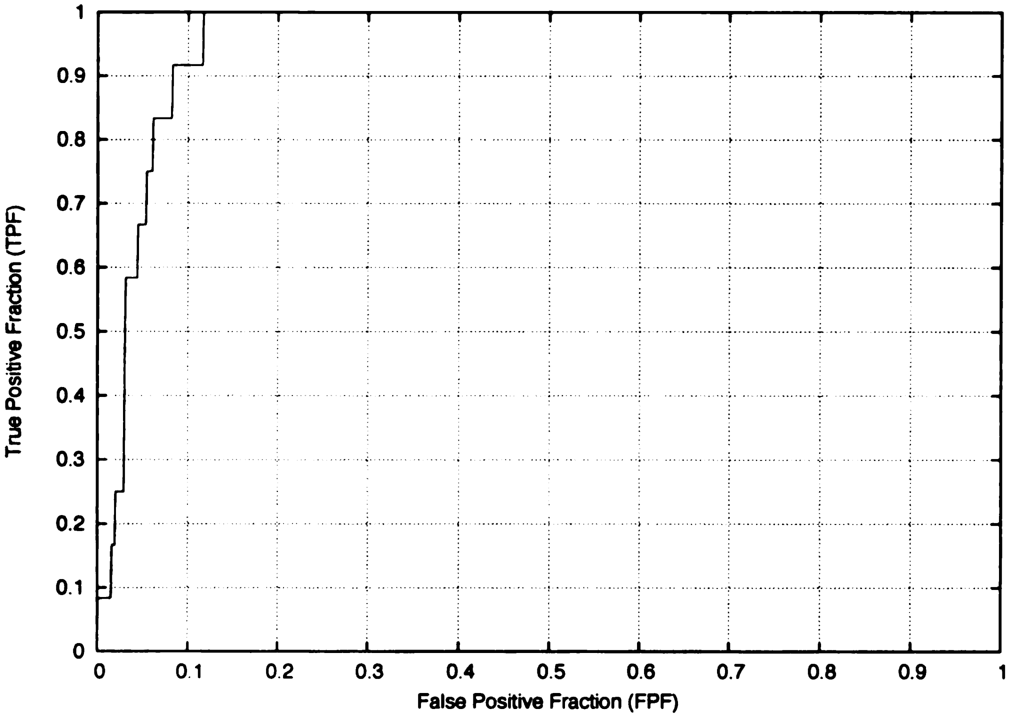


Figure 5.18: Per-attack ROC curve for Experiment 6.

Finally, the time-agglomerative ROC curve in Figure 5.19 shows a strong time-based correlation between large nearest-cluster distance and attack packets, achieving a 100% true positive rate, with a 2% false positive rate when using a 1-second window, and with a 0.6% false positive rate when using a 20-second or larger window. Compared to Experiment 5, Figure 5.19 shows a stronger correlation between large nearest-cluster distances and attack packets.

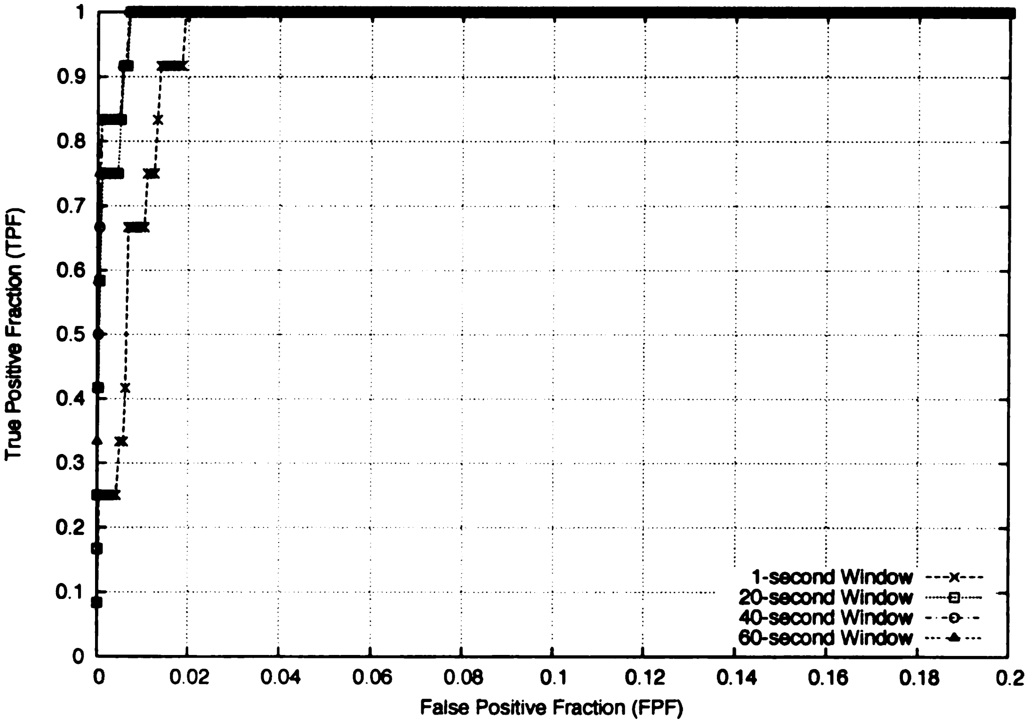


Figure 5.19: Time-agglomerative ROC curve for Experiment 6.

5.6 Summary of Results

This chapter has presented the results of 6 different experiments using MESONet for anomaly detection. Different feature sets were used for each experiment, and we tested the best performing feature sets from Chapter 4 as well as feature sets specific to anomaly detection.

Table 5.2 is a summary of the feature classes used for each experiment described in this section. It lists the true positive rate at a 10% false positive rate for both per-packet and per-attack ROC curves, and the true positive rate for 20-second time-agglomerative ROC curves at 1% false positive rate.

In conclusion, the success of time-agglomerative ROC curves suggest that anomaly detection systems have potential for detecting the presence of, if not the specific, attack. Although merely alerting that an attack is occurring is less useful than pinpointing the attack packets, it is a necessary step for the wide deployment of intrusion detection systems, especially if they are to extend beyond network-based systems.

Experiment	Packet TPR (%)	Attack TPR (%)	30-second TPR (%)	Feature Classes
2	44	75	91	SlidingWindowFeatures, SinglePacketFeatures, PayloadCountFeatures, TimeFeatures
3	34	75	91	SlidingWindowFeatures, SinglePacketFeatures, PayloadCountFeatures, TimeFeatures
4	55	84	91	SlidingWindowFeatures, SinglePacketFeatures, PayloadCountFeatures
5	69	100	100	HostFeatures, SlidingWindowFeautres, SinglePacketFeatures, PayloadCountFeatures, PacketNumberFeatures, TimeFeatures, DerivativeFeatures
6	70	91	100	HostFeatures, SlidingWindowFeautres, SinglePacketFeatures, PacketNumberFeatures, TimeFeatures, DerivativeFeatures

Table 5.2: Summary of anomaly detection validation tests: per-packet and per-attack True Positive Rate (TPR) at 10% false positive rate, 20-second time-agglomerative TPR at 1% false positive rate.

Chapter 6

Conclusions and Future Work

In this thesis we explored the application of developmental learning techniques to network intrusion detection. We used a MESOTree-based developmental learning algorithm, in conjunction with runtime selectable features, to construct a unified framework for misuse and anomaly detection systems. This framework, called MESONet, encapsulates network traffic capturing, feature extraction, training, and misuse and anomaly testing.

We have shown that machine learning techniques can be successfully applied in a systems domain, specifically network intrusion detection. As systems become increasingly complex, machine learning approaches become increasingly important, as they offer an approach that is capable of dealing with this complexity.

We have evaluated the performance of MESONet through experimentation, conducting a series of misuse and anomaly detection experiments on a subset of the 1999 DARPA IDS Evaluation datasets. Performance tests shows that while MESONet is capable of misuse detection with high accuracy, it suffers from low attack detection rate. On the other hand, depending on the feature set used, MESONet achieves high accuracy when on anomaly detection, achieving a 100% true positive rate with a 0.2% false positive rate when using a 40-second detection window.

Possible future work in network intrusion detection includes incorporating different types of features into MESONet, including non-stationary models [35] and additional traffic statistics [41]. A problem facing all network intrusion detection research, however, is the lack of available real-world network traffic that includes a ground truth.

Possible future work not related to network intrusion detection includes the application of these techniques to general system security. For example, features regarding system behavior, including process state, hardware activity, and audit logs, could be extracted from a running operating system kernel. A framework similar to MESONet could be created for these features, after which developmental learning techniques could be used to classify system behavior. A difficulty associated with this research includes the lack of ground truth for training data.

APPENDICES

Appendix A

Feature Development

Composable feature sets do little good for rapid experimentation if developing new features is onerous, therefore MESONet provides a common interface for developing new features. Figure A.1 is a code listing of the Java interface that must be implemented by any class that extracts features from network packets. The **toFloatArray(...)** methods are called with the current captured packet, which must be either a TCP, UDP, or ICMP packet, and its arrival time. The **init(...)** method is called at object instantiation time, and is passed the enclosing XML element from the configuration file for initialization purposes.

As an example of the ease by which new features can be developed for MESONet Figure A.2 is a listing of the hypothetical **TCPPercentage** class. This simplistic class calculates the percentage of traffic that uses the TCP protocol, on a packet-by-packet basis. Figure A.3 is the relevant portion from a configuration file that would use this new feature. If this feature were to be used with destination grouping (by using **HostFeatures**, for example), then it would calculate the percentage of TCP traffic destined for the same host. No additional work is needed to incorporate this feature into MESONet.

```
import org.jdom.*;
import net.sourceforge.jpcap.util.*;
import net.sourceforge.jpcap.net.*;

/*
 * Interface for all feature extractors in MESONet.
 */
public interface FeatureStrategy {
    // Initialization
    // featureStrategyTag is the enclosing XML
    // element from the configuration file.
    public void init( Element featureStrategyTag );

    // One of these are called for each packet that is captured
    // by MESONet. They return an array of floats, that MUST
    // be the same size for every packet.
    public float[] toFloatArray( TCPPacket pkt, Timeval time );
    public float[] toFloatArray( UDPPacket pkt, Timeval time );
    public float[] toFloatArray( ICMPPacket pkt, Timeval time );
}
```

Figure A.1: Java interface for feature extraction.

```
import org.jdom.*;
import net.sourceforge.jpcap.net.*;
import net.sourceforge.jpcap.util.*;

/*
* Example implementation of a feature extractor.
* This class calculates the percentage of all traffic that is
* TCP, and returns a 1-element float array containing this
* percentage.
*/
public class TCPPercentage implements FeatureStrategy {
    private float count=0;
    private float tcp=0;

    // No initialization.
    public void init( Element featureStrategyTag ) {
    }

    // Adjust the count of TCP traffic.
    public float[] toFloatArray( TCPPacket pkt, Timeval time ) {
        ++tcp; ++count;
        return new float[] { tcp / count };
    }

    // Adjust the count of non-TCP traffic.
    public float[] toFloatArray( UDPPacket pkt, Timeval time ) {
        ++count;
        return new float[] { tcp / count };
    }
    public float[] toFloatArray( ICMPPacket pkt, Timeval time ) {
        ++count;
        return new float[] { tcp / count };
    }
}
```

Figure A.2: Listing of all code for the TCPPercentage class.

```
<featureStrategy classname="TCPPercentage"/>
```

Figure A.3: Feature set using the `TCPPercentage` class.

Appendix B

Misuse Detection Feature Set Configurations

This appendix lists the complete feature set configurations for all misuse detection experiments conducted in Chapter 4.

Figure B.1 lists the feature set configuration used for Experiment 1, consisting of only the SinglePacketFeatures class.

```
<featureStrategy classname="SinglePacketFeatures"/>
```

Figure B.1: Feature set configuration for Experiment 1.

Figure B.2 lists the feature set configuration used for Experiment 2, which augments SinglePacketFeatures with the PayloadCountFeatures class via the use of the CompoundFeatureStrategy.

```
<featureStrategy classname="CompoundFeatureStrategy">
  <arg name="subStrategy">
    <featureStrategy classname="SinglePacketFeatures"/>
  </arg>
  <arg name="subStrategy">
    <featureStrategy classname="PayloadCountFeatures"/>
  </arg>
</featureStrategy>
```

Figure B.2: Feature set configuration for Experiment 2.

Figure B.3 lists the feature set configuration used for Experiment 3, which augments **SinglePacketFeatures** with the **PayloadValueFeatures** class. It should be noted **PayloadValueFeatures** violates the ordering principle described in Section 3.2.

```
<featureStrategy classname="CompoundFeatureStrategy">
  <arg name="subStrategy">
    <featureStrategy classname="SinglePacketFeatures"/>
  </arg>
  <arg name="subStrategy">
    <featureStrategy classname="PayloadValueFeatures">
      <arg name="payloadLength" value="100"/>
    </featureStrategy>
  </arg>
</featureStrategy>
```

Figure B.3: Feature set configuration for Experiment 3.

Figure B.4 lists the feature set configuration used for Experiment 4, which uses **SlidingWindowFeatures** to construct feature vectors from the **SinglePacketFeatures** and **PayloadCountFeatures** of two packets.

```
<featureStrategy classname="SlidingWindowFeatures">
  <arg name="windowSize" value="2"/>
  <arg name="subStrategy">
    <featureStrategy classname="CompoundFeatureStrategy">
      <arg name="subStrategy">
        <featureStrategy classname="SinglePacketFeatures"/>
      </arg>
      <arg name="subStrategy">
        <featureStrategy classname="PayloadCountFeatures"/>
      </arg>
    </featureStrategy>
  </arg>
</featureStrategy>
```

Figure B.4: Feature set configuration for Experiment 4.

Figure B.5 lists the feature set configuration used for Experiment 5, which uses SlidingWindowFeatures to construct feature vectors from the SinglePacketFeatures, PayloadCountFeatures, and TimeFeatures of two packets.

```
<featureStrategy classname="SlidingWindowFeatures">
  <arg name="windowSize" value="2"/>
  <arg name="subStrategy">
    <featureStrategy classname="CompoundFeatureStrategy">
      <arg name="subStrategy">
        <featureStrategy classname="SinglePacketFeatures"/>
      </arg>
      <arg name="subStrategy">
        <featureStrategy classname="PayloadCountFeatures"/>
      </arg>
      <arg name="subStrategy">
        <featureStrategy classname="TimeFeatures">
          <arg name="windowSize" value="10"/>
        </featureStrategy>
      </arg>
    </featureStrategy>
  </arg>
</featureStrategy>
```

Figure B.5: Feature set configuration for Experiment 5.

Figure B.6 lists the feature set configuration used for Experiment 6, which appends TimeFeatures to a two-packet sliding window consisting of SinglePacketFeatures and PayloadCountFeatures.

Figure B.7 lists the feature set configuration used for Experiment 7, which uses a three-packet sliding window of SinglePacketFeatures, PayloadCountFeatures, and TimeFeatures.

Figure B.8 lists the feature set configuration used for Experiment 8, which uses SlidingWindowFeatures to construct feature vectors from the SinglePacketFeatures, PayloadCountFeatures, TimeFeatures, and

```
<featureStrategy classname="CompoundFeatureStrategy">
  <arg name="subStrategy">
    <featureStrategy classname="SlidingWindowFeatures">
      <arg name="windowSize" value="2"/>
      <arg name="subStrategy">
        <featureStrategy classname="CompoundFeatureStrategy">
          <arg name="subStrategy">
            <featureStrategy classname="SinglePacketFeatures"/>
          </arg>
          <arg name="subStrategy">
            <featureStrategy classname="PayloadCountFeatures"/>
          </arg>
        </featureStrategy>
      </arg>
    </featureStrategy>
  </arg>
  <arg name="subStrategy">
    <featureStrategy classname="TimeFeatures">
      <arg name="windowSize" value="10"/>
    </featureStrategy>
  </arg>
</featureStrategy>
```

Figure B.6: Feature set configuration for Experiment 6.

PacketNumberFeatures of two packets.

Figure B.9 lists the feature set configuration used for Experiment 9, which appends TimeFeatures and PacketNumberFeatures to a two-packet sliding window consisting of SinglePacketFeatures and PayloadCountFeatures.

Figure B.10 lists the feature set configuration used for Experiment 10, which appends TimeFeatures and PacketNumberFeatures to a two-packet sliding window consisting of SinglePacketFeatures and PayloadCountFeatures, and additionally applies destination grouping through the use of HostFeatures.

Figure B.11 lists the feature set configuration used for Experiment 11, which appends TimeFeatures to a two-packet sliding window consisting of SinglePacketFeatures and PayloadCountFeatures, and additionally applies des-

```
<featureStrategy classname="SlidingWindowFeatures">
  <arg name="windowSize" value="3"/>
  <arg name="subStrategy">
    <featureStrategy classname="CompoundFeatureStrategy">
      <arg name="subStrategy">
        <featureStrategy classname="SinglePacketFeatures"/>
      </arg>
      <arg name="subStrategy">
        <featureStrategy classname="PayloadCountFeatures"/>
      </arg>
      <arg name="subStrategy">
        <featureStrategy classname="TimeFeatures">
          <arg name="windowSize" value="10"/>
        </featureStrategy>
      </arg>
    </featureStrategy>
  </arg>
</featureStrategy>
```

Figure B.7: Feature set configuration for Experiment 7.

tination grouping through the use of HostFeatures.

```
<featureStrategy classname="SlidingWindowFeatures">
  <arg name="windowSize" value="2"/>
  <arg name="subStrategy">
    <featureStrategy classname="CompoundFeatureStrategy">
      <arg name="subStrategy">
        <featureStrategy classname="SinglePacketFeatures"/>
      </arg>
      <arg name="subStrategy">
        <featureStrategy classname="PayloadCountFeatures"/>
      </arg>
      <arg name="subStrategy">
        <featureStrategy classname="TimeFeatures">
          <arg name="windowSize" value="10"/>
        </featureStrategy>
      </arg>
      <arg name="subStrategy">
        <featureStrategy classname="PacketNumberFeatures"/>
      </arg>
    </featureStrategy>
  </arg>
</featureStrategy>
```

Figure B.8: Feature set configuration for Experiment 8.

```
<featureStrategy classname="CompoundFeatureStrategy">
  <arg name="subStrategy">
    <featureStrategy classname="SlidingWindowFeatures">
      <arg name="windowSize" value="2"/>
      <arg name="subStrategy">
        <featureStrategy classname="CompoundFeatureStrategy">
          <arg name="subStrategy">
            <featureStrategy classname="SinglePacketFeatures"/>
          </arg>
          <arg name="subStrategy">
            <featureStrategy classname="PayloadCountFeatures"/>
          </arg>
        </featureStrategy>
      </arg>
    </featureStrategy>
  </arg>
  <arg name="subStrategy">
    <featureStrategy classname="CompoundFeatureStrategy">
      <arg name="subStrategy">
        <featureStrategy classname="TimeFeatures">
          <arg name="windowSize" value="10"/>
        </featureStrategy>
      </arg>
      <arg name="subStrategy">
        <featureStrategy classname="PacketNumberFeatures"/>
      </arg>
    </featureStrategy>
  </arg>
</featureStrategy>
```

Figure B.9: Feature set configuration for Experiment 9.

```
<featureStrategy classname="HostFeatures">
  <arg name="sortby" value="dst"/>
  <arg name="subStrategy">
    <featureStrategy classname="CompoundFeatureStrategy">
      <arg name="subStrategy">
        <featureStrategy classname="SlidingWindowFeatures">
          <arg name="windowSize" value="2"/>
          <arg name="subStrategy">
            <featureStrategy classname="CompoundFeatureStrategy">
              <arg name="subStrategy">
                <featureStrategy classname="SinglePacketFeatures"/>
              </arg>
              <arg name="subStrategy">
                <featureStrategy classname="PayloadCountFeatures"/>
              </arg>
            </featureStrategy>
          </arg>
        </featureStrategy>
      </arg>
      <arg name="subStrategy">
        <featureStrategy classname="CompoundFeatureStrategy">
          <arg name="subStrategy">
            <featureStrategy classname="TimeFeatures">
              <arg name="windowSize" value="10"/>
            </featureStrategy>
          </arg>
          <arg name="subStrategy">
            <featureStrategy classname="PacketNumberFeatures"/>
          </arg>
        </featureStrategy>
      </arg>
    </featureStrategy>
  </arg>
</featureStrategy>
```

Figure B.10: Feature set configuration for Experiment 10.

```
<featureStrategy classname="HostFeatures">
  <arg name="sortby" value="dst"/>
  <arg name="subStrategy">
    <featureStrategy classname="CompoundFeatureStrategy">
      <arg name="subStrategy">
        <featureStrategy classname="SlidingWindowFeatures">
          <arg name="windowSize" value="2"/>
          <arg name="subStrategy">
            <featureStrategy classname="CompoundFeatureStrategy">
              <arg name="subStrategy">
                <featureStrategy classname="SinglePacketFeatures"/>
              </arg>
              <arg name="subStrategy">
                <featureStrategy classname="PayloadCountFeatures"/>
              </arg>
            </featureStrategy>
          </arg>
        </featureStrategy>
      </arg>
      <arg name="subStrategy">
        <featureStrategy classname="TimeFeatures">
          <arg name="windowSize" value="10"/>
        </featureStrategy>
      </arg>
    </featureStrategy>
  </arg>
</featureStrategy>
```

Figure B.11: Feature set configuration for Experiment 11.

Appendix C

Anomaly Detection Feature Set Configurations

This appendix lists the complete feature set configurations for all anomaly detection experiments conducted in Chapter 5.

Figure C.1 lists the feature set configuration used for Experiment 2, which appends **TimeFeatures** to a two-packet sliding window consisting of **SinglePacketFeatures** and **PayloadCountFeatures**.

Figure C.2 lists the feature set configuration used for Experiment 3, which uses **SlidingWindowFeatures** to construct feature vectors from the **SinglePacketFeatures**, **PayloadCountFeatures**, and **TimeFeatures** of two packets.

Figure C.3 lists the feature set configuration used for Experiment 4, which uses **SlidingWindowFeatures** to construct feature vectors from the **SinglePacketFeatures** and **PayloadCountFeatures** of two packets.

Figure C.4 lists the feature set configuration used for Experiment 5, which uses a destination-grouped two-packet sliding window of **SinglePacketFeatures** and **PayloadCountFeatures**, appended to **PacketNumberFeatures** and the derivative of **TimeFeatures**.

Figure C.5 lists the feature set configuration used for Experiment 6, which

```
<featureStrategy classname="CompoundFeatureStrategy">
  <arg name="subStrategy">
    <featureStrategy classname="SlidingWindowFeatures">
      <arg name="windowSize" value="2"/>
      <arg name="subStrategy">
        <featureStrategy classname="CompoundFeatureStrategy">
          <arg name="subStrategy">
            <featureStrategy classname="SinglePacketFeatures"/>
          </arg>
          <arg name="subStrategy">
            <featureStrategy classname="PayloadCountFeatures"/>
          </arg>
        </featureStrategy>
      </arg>
    </featureStrategy>
  </arg>
  <arg name="subStrategy">
    <featureStrategy classname="TimeFeatures">
      <arg name="windowSize" value="10"/>
    </featureStrategy>
  </arg>
</featureStrategy>
```

Figure C.1: Feature set configuration for Experiment 2.

uses a destination-grouped two-packet sliding window of **SinglePacketFeatures**, appended to **PacketNumberFeatures** and the derivative of **TimeFeatures**.

```
<featureStrategy classname="SlidingWindowFeatures">
  <arg name="windowSize" value="2"/>
  <arg name="subStrategy">
    <featureStrategy classname="CompoundFeatureStrategy">
      <arg name="subStrategy">
        <featureStrategy classname="SinglePacketFeatures"/>
      </arg>
      <arg name="subStrategy">
        <featureStrategy classname="PayloadCountFeatures"/>
      </arg>
      <arg name="subStrategy">
        <featureStrategy classname="TimeFeatures">
          <arg name="windowSize" value="10"/>
        </featureStrategy>
      </arg>
    </featureStrategy>
  </arg>
</featureStrategy>
```

Figure C.2: Feature set configuration for Experiment 3.

```
<featureStrategy classname="SlidingWindowFeatures">
  <arg name="windowSize" value="2"/>
  <arg name="subStrategy">
    <featureStrategy classname="CompoundFeatureStrategy">
      <arg name="subStrategy">
        <featureStrategy classname="SinglePacketFeatures"/>
      </arg>
      <arg name="subStrategy">
        <featureStrategy classname="PayloadCountFeatures"/>
      </arg>
    </featureStrategy>
  </arg>
</featureStrategy>
```

Figure C.3: Feature set configuration for Experiment 4.

```
<featureStrategy classname="HostFeatures">
  <arg name="sortby" value="dst"/>
  <arg name="subStrategy">
    <featureStrategy classname="CompoundFeatures">
      <arg name="subStrategy">
        <featureStrategy classname="DerivativeFeatures">
          <arg name="granularity" value="1000"/>
          <arg name="windowSize" value="5"/>
          <arg name="subStrategy">
            <featureStrategy classname="TimeFeatures">
              <arg name="windowSize" value="10"/>
            </featureStrategy>
          </arg>
        </featureStrategy>
      </arg>
      <arg name="subStrategy">
        <featureStrategy classname="PacketNumberFeatures"/>
      </arg>
      <arg name="subStrategy">
        <featureStrategy classname="SlidingWindowFeatures">
          <arg name="windowSize" value="2"/>
          <arg name="subStrategy">
            <featureStrategy classname="CompoundFeatureStrategy">
              <arg name="subStrategy">
                <featureStrategy classname="SinglePacketFeatures"/>
              </arg>
              <arg name="subStrategy">
                <featureStrategy classname="PayloadCountFeatures"/>
              </arg>
            </featureStrategy>
          </arg>
        </featureStrategy>
      </arg>
    </featureStrategy>
  </arg>
</featureStrategy>
```

Figure C.4: Feature set configuration for Experiment 5.

```
<featureStrategy classname="HostFeatures">
  <arg name="sortby" value="dst"/>
  <arg name="subStrategy">
    <featureStrategy classname="CompoundFeatures">
      <arg name="subStrategy">
        <featureStrategy classname="DerivativeFeatures">
          <arg name="granularity" value="1000"/>
          <arg name="windowSize" value="5"/>
          <arg name="subStrategy">
            <featureStrategy classname="TimeFeatures">
              <arg name="windowSize" value="10"/>
            </featureStrategy>
          </arg>
        </featureStrategy>
      </arg>
      <arg name="subStrategy">
        <featureStrategy classname="PacketNumberFeatures"/>
      </arg>
      <arg name="subStrategy">
        <featureStrategy classname="SlidingWindowFeatures">
          <arg name="windowSize" value="2"/>
          <arg name="subStrategy">
            <featureStrategy classname="SinglePacketFeatures"/>
          </arg>
        </featureStrategy>
      </arg>
    </featureStrategy>
  </arg>
</featureStrategy>
```

Figure C.5: Feature set configuration for Experiment 6.

Bibliography

[1] 1999 KDD Cup. http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html.

[2] The shmoo group, capture the flag contest, DEFCON-9. Website, July 2001. http://www.cs.ucsb.edu/˜dhm/defcon-tcpdumps.

[3] tcpdump/libpcap, 2001. http://www.tcpdump.org.

[4] UCLA CSD packet traces, D-WARD project. Website, December 2002. http://lever.cs.ucla.edu/ddos/traces.

[5] Charu C. Aggarwal, Cecilia Procopiuc, Joel L. Wolf, Philip S. Yu, and Jong Soo Park. Fast algorithms for projected clustering. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 61–72, Philadelphia, Pennsylvania, USA, June 1999.

[6] J. P. Anderson. Computer security threat monitoring and surveillance. Technical report, NIST, April 1980.

[7] M. Asaka, T. Onabuta, T. Inoue, S. Okazawa, and S. Goto. A new intrusion detection method based on discriminant analysis. In *Institute of Electronics, Information, and Communication Engineers (IEICE)*, number 577 in Transactions, 2001.

[8] Stefan Axelsson. The base-rate fallacy and its implications for the difficulty of intrusion detection. In *ACM Conference on Computer and Communications Security*, pages 1–7, 1999.

[9] Stefan Axelsson. Intrusion detection systems: A survey and taxonomy. Technical Report 99-15, Chalmers Univ., March 2000.

[10] Bruce Blumberg, Marc Downie, Yuri Ivanov, Matt Berlin, Michael Patrick Johnson, and Bill Tomlinson. Integrated learning for interactive synthetic characters. In *Computer Graphics, Proceedings of SIGGRAPH'02*, 2002.

[11] J. Cannady. Artificial neural networks for misuse detection. In *Proceedings of the 1998 National Information Systems Security Conference (NISSC'98) October 5-8 1998. Arlington, VA.*, pages 443–456, 1998.

[12] Patrick Charles, John Guthrie, Justin Haddad, Steve Bitteker, and David B. Knoester. JPCap: Network packet capture facility for java, May 2001. http://sourceforge.net/projects/jpcap.

[13] William W. Cohen. Fast effective rule induction. In *Proc. of the 12th International Conference on Machine Learning*, pages 115–123. Morgan Kaufmann, July 1995.

[14] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification, Second Edition.* John Wiley and Sons, Incorporated, New York, New York, USA, 2001.

[15] Computer Economics. Virus attack costs on the rise – again. Website, 2002. http://www.computereconomics.com.

[16] James P. Egan. *Signal Detection Theory and ROC Analysis.* Series in Cognition and Perception. Academic Press, June 1975.

[17] S. Floyd and V. Paxson. Difficulties in simulating the internet. In *IEEE/ACM Transactions on Networking (TON)*, volume 9, August 2001.

[18] Stephanie Forrest, Steven A. Hofmeyr, Anil Somayaji, and Thomas A. Longstaff. A sense of self for Unix processes. In *Proceedinges of the 1996 IEEE Symposium on Research in Security and Privacy*, pages 120–128. IEEE Computer Society Press, 1996.

[19] A. Ghosh, J. Wanken, and F. Charron. Detecting anomalous and unknown intrusions against programs. In *Proceedings of the 1998 Annual Computer Security Applications Conference (ACSAC'98), December 1998.*, pages 259–267. Los Alamitos, CA, USA : IEEE Comput. Soc, 1998, 1998.

[20] Anup K. Ghosh and Aaron Schwartzbard. A study in using neural networks for anomaly and misuse detection. In *Proceedings of the 8th USENIX Security Symposium*, pages 141–152, August 1999.

[21] Anup K. Ghosh, Aaron Schwartzbard, and Michael Schatz. Learning program behavior profiles for intrusion detection. In *Proceedings 1st USENIX Workshop on Intrusion Detection and Network Monitoring*, pages 51–62, April 1999.

[22] Wey-Shiuan Hwang and Juyang Weng. Hierarchical discriminant regression. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1277–1293, November 2000.

[23] K. Julisch. Dealing with false positives in intrusion detection. In *3rd Intl. Workshop on Recent Advances in Intrusion Detection RAID*, Lecture Notes in Computer Science. Springer-Verlag, 2000.

[24] Eric P. Kasten and Philip K. McKinley. A framework-based classifier for online learning. Technical Report MSU-CSE-04-1, Department of Computer Science and Engineering, Michigan State University, East Lansing, Michigan, USA, January 2004.

[25] Richard A. Kemmerer. NSTAT: A model-based real-time network intrusion detection system. Technical Report TRCS97-18, University of California, Santa Barbara, 1998.

[26] Kris Kendall. A database of computer attacks for the evaluation of intrusion detection systems, 1998. Master's thesis.

[27] Sandeep Kumar and Eugene H. Spafford. A Pattern Matching Model for Misuse Intrusion Detection. In *Proceedings of the 17th National Computer Security Conference*, pages 11–21, 1994.

[28] T. Lane and C. E. Brodley. An application of machine learning to anomaly detection. In *Proc. 20th NIST-NCSC National Information Systems Security Conference*, pages 366–380, 1997.

[29] Terran Lane and Carla E. Brodley. Temporal sequence learning and data reduction for anomaly detection. *ACM Transactions on Information and System Security*, 2(3):295–331, 1999.

[30] Wenke Lee, Salvatore J. Stolfo, and Kui W. Mok. A data mining framework for building intrusion detection models. In *IEEE Symposium on Security and Privacy*, pages 120–132, 1999.

[31] Lin, Wang, and Jajodia. Abstraction-based misuse detection: High-level specifications and adaptable strategies. In *PCSFW: Proceedings of The 11th Computer Security Foundations Workshop*. IEEE Computer Society Press, 1998.

[32] Richard Lippmann, David Fried, Isaac Graf, Joshua Haines, Kristopher Kendall, David McClung, Dan Weber, Seth Webster, Dan Wyschogrod, Robert Cunningham, and Marc Zissman. Evaluating intrusion detection systems: The 1998 DARPA off-line intrusion detection evaluation. In *Proceedings of the DARPA Information Survivability Conference and Exposition*, Los Alamitos, CA, 2000. IEEE Computer Society Press.

[33] Richard Lippmann, Joshua W. Haines, David J. Fried, Jonathan Korba, and Kumar Das. The 1999 darpa off-line intrusion detection evaluation. *Comput. Networks*, 34(4):579–595, 2000.

[34] M. V. Mahoney and P. K. Chan. An analysis of the 1999 DARPA/Lincoln Laboratory evaluation for network anomaly detection. In *Proc. of Recent Advances in Intrusion Detection (RAID)*, pages 220–237, 2003.

[35] Matthew V. Mahoney and Philip K. Chan. Learning nonstationary models of normal network traffic for detecting novel attacks.

[36] J. McHugh. Testing intrusion detection systems: A critique of the 1998 and 1999 darpa off-line intrusion detection system evaluation as performed by lincoln laboratory. In *ACM Transactions on Information and System Security*, November 2000.

[37] Sreerama K. Murthy. Automatic construction of decision trees from data: A multi-disciplinary survey. *Data Mining and Knowledge Discovery*, 2(4):345–389, 1998.

[38] Darren Mutz. Mucus tcpdumps. Website, November 2003. http://www.cs.ucsb.edu/~dhm/mucus-tcpdumps.

[39] NFR Inc. Network flight recorder, 1997. http://www.nfr.com.

[40] V. Paxon and S. Floyd. Why we don't know how to simulate the internet. In *In Proceedings of the 29th Conference on Winter Simulation (WSC)*, 1997.

[41] Vern Paxson. Bro: a system for detecting network intruders in real-time. *Computer Networks (Amsterdam, Netherlands: 1999)*, 31(23–24):2435–2463, 1999.

[42] P. A. Porras and P. G. Neumann. EMERALD: Event monitoring enabling responses to anomalous live disturbances. In *Proc. 20th NIST-NCSC National Information Systems Security Conference*, pages 353–365, 1997.

[43] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach.* Prentice-Hall, 1995.

[44] R. Sekar, Y. Guang, S. Verma, and T. Shanbhag. A high-performance network intrusion detection system. In *ACM Conference on Computer and Communications Security*, pages 8–17, 1999.

[45] Mei-Ling Shyu, Shu-Ching Chen, Kanoksri Sarinnapakorn, and LiWu Chang. A novel anomaly detection scheme based on principal component classifier. In *Proceedings of the IEEE Foundations and New Directions of Data Mining Workshop*, pages 172–179, 2003. Held in in conjunction with the Third IEEE International Conference on Data Mining ICDM'03.

[46] Arlindo Silva, Ana Neves, and Ernesto Costa. Genetically programming networks to evolve memory mechanisms. In Wolfgang Banzhaf, Jason Daida, Agoston E. Eiben, Max H. Garzon, Vasant Honavar, Mark Jakiela, and Robert E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 2, page 1448, Orlando, Florida, USA, 13-17 1999. Morgan Kaufmann.

[47] G. Vigna, S. Eckmann, and R. Kemmerer. The stat tool suite. In *DARPA Information Survivability Conference and Exposition (DISCEX)*. IEEE Computer Society Press, 2000.

[48] Giovanni Vigna and Richard A. Kemmerer. NetSTAT: A network-based intrusion detection approach. In *Annual Computer Security Applications Conference ACSAC*, 1998.

[49] J. Weng and W. Hwang. Online image classification using ihdr. In *International Journal on Document Analysis and Recognition*, volume 5, pages 118–125, 2003.

[50] Juyang Weng. A theory for mentally developing robots. In *Proceedings 2nd International Conference on Development and Learning*. IEEE Computer Society Press, June 2002.

[51] Juyang Weng, Colin H. Evans, Wey S. Hwang, and Yong-Beom Lee. The developmental approach to artificial intelligence: Concepts, developmental algorithms and experimental results. Technical Report MSU-CPS-98-25, Michigan State University, July 1998.