Intelligent Inference Methods for Automated Network Diagnostics

CHATHURANGA WIDANAPATHIRANA

SUBMITTED IN TOTAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Department of Electrical and Computer Systems Engineering
FACULTY OF ENGINEERING
MONASH UNIVERSITY
AUSTRALIA

AUGUST 2015

COPYRIGHT NOTICES

Notice 1

Notice 2

DECLARATION

In accordance with Monash University Doctorate Regulation 17.2 Doctor of Philosophy and Research Master's Regulations, the following declarations are made:

I hereby declare that this thesis contains no material which has been accepted for the award of any other degree or diploma at any university or equivalent institution and that, to the best of my knowledge and belief, this thesis contains no material previously published or written by another person, except where due reference is made in the text of the thesis.

The core theme of the thesis is Intelligent Inference Methods for Automated Network Diagnostics. The ideas, development and writing up of all the work in the thesis were the principal responsibility of myself, the candidate, working within the Department of Electrical and Computer Systems Engineering under the supervision of Dr Y. Ahmet Şekercioğlu.

Signed: 
Chathu h Widanapathirana

Date: August 2015

ABSTRACT

In today's complex networks, timely identification and resolution of performance problems is extremely challenging. Current diagnostic practices to identify the root causes of such problems primarily rely on human intervention and investigation. Fully automated and scalable systems, which are capable of identifying complex problems are needed to provide rapid and accurate diagnosis. The study presented in this thesis creates the necessary scientific basis for the automatic diagnosis of network performance faults using novel intelligent inference techniques based on machine learning. We propose three new techniques for characterisation of network soft failures, and by using them, create the Intelligent Automated Network Diagnostic (IAND) system.

First, we propose Transmission Control Protocol (TCP) trace characterisation techniques that use aggregated TCP statistics. Faulty network components embed unique artefacts in TCP packet streams by altering the normal protocol behaviour. Our technique captures such artefacts and generates a set of unique fault signatures. We first introduce Normalised Statistical Signatures (NSSs) with 460 features, a novel representation of network soft failures to provide the basis for diagnosis. Since not all 460 features contribute equally to the identification of a particular fault, we then introduce improved forms of NSSs called EigenNSS and FisherNSS with reduced complexity and greater class separability. Evaluations show that we can achieve dimensionality reduction of over 95% and detection accuracies up to 95% while achieving micro-second diagnosis times with these signatures.

Second, given NSSs have features that are dependent on link properties, we introduce a technique called Link Adaptive Signature Estimation (LASE) using regression-based predictors to artificially generate NSSs for a large number of link parameter combinations. Using LASE, the system can be trained to suit the exact networking environment, however dynamic, with a minimal set of sample data. For extensive performance evaluation, we collected 1.2 million sample traces for 17 types of device failures on 8 TCP variants over various types of networks using a combination of fault injection and link emulation techniques.

Third, to automate fault identification, we propose a modular inference technique that learns from the patterns embedded in the signatures, and create Fault Classifier Modules (FCMs). FCMs use support vector machines to uniquely identify individual faults and are designed using soft class boundaries to provide generalised fault detection capability. The use of a modular design and generic algorithm that can be trained and tuned based on the specific faults, offers scalability and is a key differentiator from the existing systems that use specific algorithms to detect each fault. Experimental evaluations show that FCMs can achieve detection accuracies of between 90% – 98%.

The signatures and classifiers are used as the building blocks to create the IAND system with its two main sub-systems: IAND-*k* and IAND-*h*. The IAND-*k* is a modular diagnostic system for automatic detection of previously known problems using FCMs. The IAND-*k* system is applied for accurately detecting faulty links and diagnosing problems in end-user devices in a wide range of network types (IAND-*k*UD, IAND-*k*CC). Extensive evaluation of the systems demonstrated high overall detection accuracies up to 96.6% with low false positives and over 90% accuracy even in the most difficult scenarios. Here, the FCMs use supervised machine learning methods and can only detect previously known problems. To extend the diagnostic capability to detect previously unknown problems, we propose IAND-*h*, a hybrid classifier system that uses a combination of unsupervised machine learning-based clustering and supervised machine learning-based classification. The evaluation of the system shows that previously unknown faults can be detected with over 92% accuracy. The IAND-*h* system also offers real-time detection capability with diagnosis times between 4 μs and 66 μs.

The techniques and systems proposed during this research contribute to the state of the art of network diagnostics and focus on scalability, automation and modularity with evaluation results demonstrating a high degree of accuracy.

PUBLICATIONS

During the course of this project, a number of peer-reviewed articles have been published. They are listed here for reference.

Peer-Reviewed Journal Articles

- C. H. Widanapathirana, X. Ang, J. C. Li, M. V. Ivanovich, P. G. Fitzpatrick and Y. A. Şekercioğlu, "A Hybrid Classifier Using Reduced Signatures for Automated Soft-Failure Diagnosis in Network End-User Devices", *Journal of Networks*, Vol 9, No 12, pp. 3275-3289, Dec 2014

- C. H. Widanapathirana, J. C. Li, M. V. Ivanovich, P. G. Fitzpatrick and Y. A. Şekercioğlu, "Adaptive Statistical Signatures of Network Soft-Failures in User Devices", *The Computer Journal*, Vol 57, No 8, pp.1262-1278, July 2013

- C. H. Widanapathirana, Y. A. Şekercioğlu, P. G. Fitzpatrick, M. V. Ivanovich and J. C. Li, "Automated Inference System for End-To-End Diagnosis of Network Performance Issues in Client-Terminal Devices", *International Journal of Computer Networks & Communications (IJCNC)*, Vol 4, No 3, pp. 37-56, May 2012

- C. H. Widanapathirana, Y. A. Şekercioğlu and B-M. Goi, "Hybrid FPMS: A New Fairness Protocol Management Scheme for Community Wireless Mesh Networks", *KSII Transactions on Internet and Information Systems*, Vol 5, Issue 11, pp. 1909-1928, Apr 2011 (Additional publication outside the scope of this thesis)

Peer-Reviewed Conference Articles

- C. H. Widanapathirana, J. C. Li, M. V. Ivanovich, P. G. Fitzpatrick, and Y. A. Şekercioğlu, "Automated Diagnosis of Known and Unknown Soft-Failure in User Devices Using Transformed Signatures and Single Classifier Architecture", *The 38th IEEE Conference on Local Computer Networks (LCN'13)*, Sydney, Australia, pp. 614-621, Oct 2013.

- C. H. Widanapathirana, J. C. Li, M. V. Ivanovich, P. G. Fitzpatrick, and Y. A. Şekercioğlu, "Adaptive Signatures of Soft-Failures in End-User Devices Using Aggregated TCP Statistics", *IFIP/IEEE International Symposium on Integrated Network Management (IM'13)*, Ghent, Belgium, pp. 752-755, May 2013.

- C. H. Widanapathirana, Y. A. Şekercioğlu, P. G. Fitzpatrick, M. V. Ivanovich and J. C. Li, "Diagnosing Client Faults Using SVM-based Intelligent Inference from TCP Packet Traces", *6th International Conference on Broadband Communications & Biomedical Applications (IB2COM'11)*, Melbourne, Australia, pp. 68-73, Dec 2011, (finalist for the best paper award).
- C. H. Widanapathirana, Y. A. Şekercioğlu, P. G. Fitzpatrick, M. V. Ivanovich and J. C. Li, "Intelligent Automated Diagnosis of Client Device Bottlenecks in Private Clouds", *4th IEEE/ACM International Conference on Utility and Cloud Computing (UCC'11)*, Melbourne, Australia, pp. 261-266, Dec 2011.

ACKNOWLEDGEMENTS

I am deeply indebted to Dr Ahmet Şekercioğlu for his guidance and advice from the beginning of my research at Monash until today. This thesis would not have been possible without his patience, willingness to help and challenge. He gave me the opportunity to think freely and determine the direction of my research, yet kept a watchful eye supporting me, making sure I was on the right path.

I would also like to thank my associate supervisors, Dr Jonathan Li, Dr Milosh Ivanovich and Dr Paul Fitzpatrick, who have taken valuable time out of their busy lives to critically look at my work, read my writing and advise me on the direction and improvements. As English is my second language, Dr Li's help with publication writing was immensely helpful, and he has always been available for a conversation no matter what the issue was. Dr Ivanovich has always provided me with valuable feedback, and has been a great source of knowledge and wisdom. His enthusiasm was infectious and his encouragement was sometimes really needed.

The staff at Monash University go great lengths to make a postgraduate student's life easy and the transition smooth. I would like to thank all staff at Monash, especially at Electrical and Computer Systems Engineering (ECSE) and Monash Graduate School for helping me with many hundreds of administrative tasks over the years, supporting me with every request I made and providing me with an environment where I could focus on my research. I also like to acknowledge the help given by Dr Alex McKnight with proof reading the thesis and improving my writing.

This note would not be complete without acknowledging Open Universities Australia, where I have been given a full-time job even before my graduation, and has given me great opportunities to innovate, build novel systems and apply some of my research to create real applications. All my managers have encouraged me to write my thesis when sometimes it seemed impossible to find the time and energy.

One of the most valuable gifts my life at ECSE has offered me is a group of friends which remains close even after graduating and taking different paths. Their support over the years in all the ups and downs that come with postgraduate and personal life has been an immense comfort. Thank you Chethiya, Asanka, Sampath, Gamage, Chanka, Randika, Uditha and all the others for making my life at Monash one of the best chapters of my life. Those conversations full of laughter,

overnight workdays, countless excursions and the infamous parties are all fond memories. Thank you for your continued support and I hope we can continue our friendship in the years to come.

I cannot thank my parents enough, for their endless love, support and encouragement. This journey would not have been possible without the sacrifices and heart-aches they had to endure. Their unconditional love and that deeply sincere "Skype" call was what kept me going when it was most difficult. Then there is that special person - my best friend, my life partner, my beautiful wife Mashi, who supported me no matter how hard things were. Thank you so much for all the sacrifices, thank you for sharing this journey through thick and thin and thank you for enduring all the hardships that came with it. She was my light when I could not see the light at the end of the tunnel, and this thesis would not have been possible if it were not for her support and encouragement.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

ADSL	Asymmetric Digital Subscriber Line
AFET	Average Feature Estimation Time
ANN	Artificial Neural Networks
BDP	Bandwidth-Delay Product
BER	Bit Error Rate
BIC	Binary Increase Congestion control
BN	Bayesian Network
CFD	Client Fault Diagnostic
CFD-M	CFD Matrix
CFD-P	CFD Parallel
D-SACK	Duplicate Selective Acknowledgement
DoS	Denial of Service
ED	Euclidean Distance
FCM	Fault Classifier Module
FEF	Feature Estimator Function
FNR	False-Negative Rate
FPR	False-Positive Rate
GRNN	General Regression Neural Networks
HSTCP	High-speed TCP
IAND	Intelligent Automated Network Diagnostic
IAND-*k*	Intelligent Automated Network Diagnostic for known problems
IAND-*k*CC	Intelligent Automated Network Diagnostic for Cloud Computing
IAND-*k*UD	Intelligent Automated Network Diagnostic for End-User Devices
IAND-*h*	Intelligent Automated Network Diagnostic hybrid classifier
IDS	Intrusion-Detection Systems
ISP	Internet Service Provider
ITIL	Information Technology Infrastructure Library
KIS	Kernel Instrumentation Set
KNN	K-Nearest Neighbour
LAN	Local Area Network
LASE	Link Adaptive Signature Estimation
LFD	Link Fault Diagnosis
LPD	Link Problem Detection
LSO	Least Square Optimisation
ML	Machine Learning
ML	Machine Learning

NAT	Network Address Translation
NB	Naive Bayes
NDT	Network Diagnostic Tool
NIC	Network Interface Card
NMS	Network Management System
NSS	Normalised Statistical Signature
OS	Operating System
PC	Personal Computer
PCA	Principal Component Analysis
POLY	Polynomial Regression
QoE	Quality of Experience
QP	Quadratic Programming
RBF	Radial Basis Function
RFC	Request For Comment
RTT	Round Trip Time
SaaS	Software-as-a-Service
SACK	Selective Acknowledgement
SMO	Sequential Minimal Optimisation
SNMP	Simple Network Management Protocol
SV	Support Vectors
SVD	Singular Value Decomposition
SVM	Support Vector Machine
SVR	Support Vector Regression
TBIT	TCP Behaviour Inference Tool
TCP	Transmission Control Protocol
TNR	True-Negative Rate
TPR	True-Positive Rate
TT	Train Time
UD	End-User Device

CHAPTER ONE

INTRODUCTION

Computer networks have become a vital and ubiquitous part of our lives. In recent years, most technological developments in the networking field have predominantly focused on improving connection medium, connection speeds, new architectures, managing congestion and network-dependent applications. As applications become more bandwidth-intensive and networks become faster, users' demand for high-speed delivery of information has increased in tandem with decreased tolerance for performance issues and "down-time". In short, users expect always-on connections without interruption. With the complexity of today's networks, it is becoming impossible to rely on traditional methods to identify and resolve performance problems and bottlenecks in end-user devices, access links, and server systems. Rapid resolution of users' problems, especially connection performance issues, will have a major impact on a network service provider's service quality in an extremely competitive and rapidly-changing environment.

1.1 Changing face of network services and consumer habits

Historically, computer software and user data resided in the end-user devices. Networks were predominantly used only to share batches of information between

devices, surf the web (intermittent communication between end-user and web server) or to communicate with highly specialised web services. 10-15 years ago, loss of connectivity in a personal computing device or in an enterprise workstation did not mean any tangible loss of productivity, or for the most part, usability of the device. Network performance issues only really affected a handful of very specialised applications and high-performance computing services.

However, cloud computing is a revolution that is defining the Information Technology in the second decade of the 21st century. The rising popularity of cloud computing services, especially Software-as-a-Service (SaaS) [1], suggests that most common user applications will soon be moved to remote servers (See Figure 1.1). Next-generation operating systems (OSs) such as Google Chrome OS further extend the cloud computing concept to run only web applications

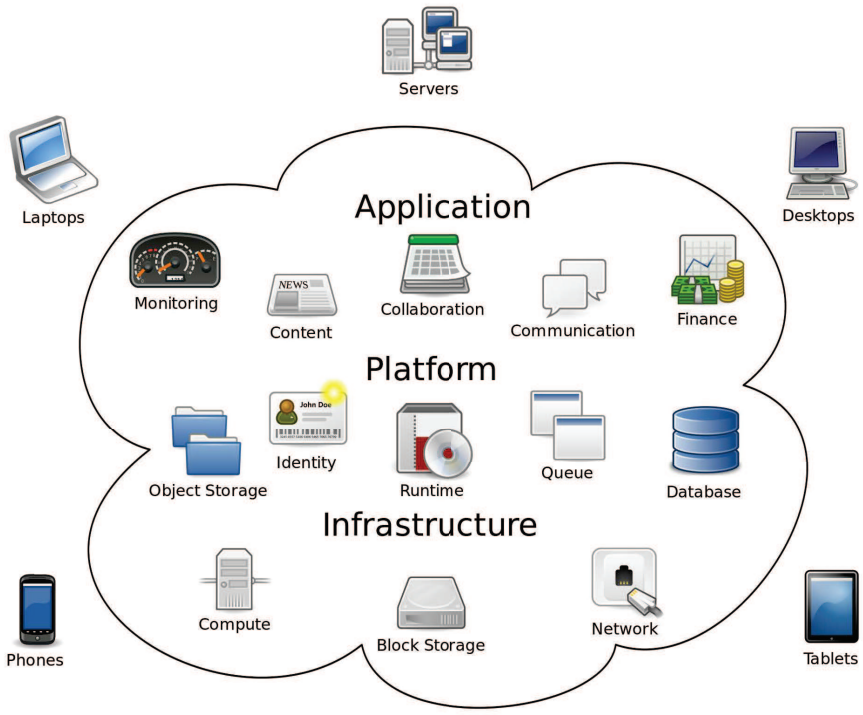


Figure 1.1: Cloud computing involves deploying groups of servers and software that allow centralised data storage and on-line, on-demand access to applications and resources. With cloud services, especially the Software-as-a-Service computing model, the consumption habits of end-users have drastically changed.

on user devices, and OSs themselves are heavily dependent on reliable network connections [2]. Whilst the transition from traditional computing to a fully cloud-based ecosystem is rapidly unfolding, average user still uses a mixture of on-device and SaaS type applications on a day-to-day basis. In addition, with new types of services such as high-definition video streaming, on-line gaming, file sharing/archiving, word processing and other media-heavy applications, users consume increasing amounts of data through their devices. This phenomenon is not limited to ordinary consumers, but is also apparent in enterprises, as companies move all their applications to the cloud at a rapidly increasing pace [3].

With the rapid adoption of cloud applications by the consumers, always available network connections and consistently fast communication speeds are becoming critically important. The networking research community has now converged on the common understanding that performance unpredictability and data-transfer bottlenecks are going to be significant obstacles to satisfactory cloud computing experience [1, 4].

1.2 Network failures and challenges

Performance management is a complex yet crucial part of present-day networks. Users worldwide expect and rely on network connectivity for day-to-day and even critical activities.

Network failures have been commonly divided into two main categories [5, 6, 7, 8]: “hard failures“ and “soft failures“. A hard failure is characterised by the inability of the network to deliver any bandwidth at all. Possible causes include power failures, physical discontinuities in network cables, or failures of major network components (e.g., routers or gateways). Detection of hard failures is not difficult, since if not noticed immediately by operators, users are sure to inform network personnel within minutes of the failure. Soft failures are less well-defined,

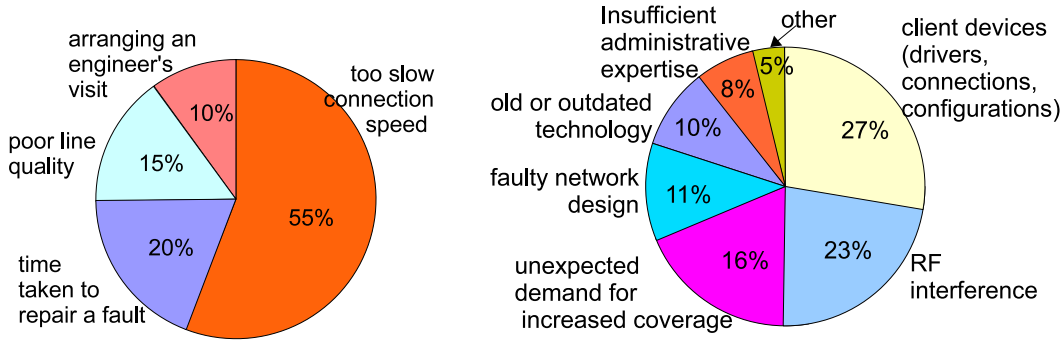
but the view of most of the research literature is that soft failures are characterised by degraded performance, or the partial loss of network bandwidth. The fact that soft failures may be seen as performance degradations suggests that to detect the onset of such an event, performance parameters could be measured with the goal of detecting sudden, anomalous shifts in performance. One difficulty with this idea is that performance levels, degradations, and time-scales of the shifts need definitions. Due to the constantly-changing character of a network environment, network performance shifts, and by association, faults are by their very nature ill-defined. A complicating factor is that network protocols make allowances for various error conditions. For example, a missed packet will be retransmitted. The retransmission consumes bandwidth that would have been available if the original packet had not been missed. Another example is collisions. Most would agree that the occurrence of one collision is not an error; the Ethernet, in fact, relies on collision phenomena for its normal operation [6]. Are two collisions in a period of one second considered an error? Twenty collisions? Two thousand collisions? Where and how does one draw the line? Because the network protocols accommodate the presence of at least some errors as a standard part of network operation, the definition of a network fault is necessarily fuzzy.

In this work, we define a "healthy" user device (UD), as a device capable of delivering typical network performance in the prevailing local network circumstances to an end-user. Severe degradation of network quality from the typical baseline is defined as a soft failure. Resolution of such soft failures is critically important for maintaining a user's quality of experience (QoE) [9]. End-user devices are defined as devices used directly by an end-user to communicate, for example, a personal computer, mobile phone, laptop or a tablet.

The definition of soft failure suggests that a failure in any part of the communication system (in any protocol layer) that causes the user to consistently experience a significantly lower network quality than what is "typical" can be

considered as a root cause of the failure. As an example, a user signing up for a fibre connection with 20 Mb/s typical download and upload speed can reasonably expect the experienced data transfer rate to fluctuate within a few megabits (10-20 Mb/s). However, if the user is constantly experiencing a 3 Mb/s bandwidth, a reasonable conclusion is the existence of a soft failure.

Slow network connections and connection interruptions experienced by users are a major issue, and therefore, two of the most common customer complaints received by ISPs. For example, according to the Ofcom quality of service review for 2009 [10], in the UK broadband market, nearly half the customer technical complaints were because of sluggish network connections or poor line quality (Figure 1.2a). Most of these problems occur either at the access link or the user's device. A survey by Cisco Systems shows that as much as 30% of wireless network performance problems are directly or partly caused by misconfiguration or faults at the user devices (Figure 1.2b) [11]. Almost all out of the box computers and operating systems are supplied with conservative default network parameters. Although these systems offer extensive customisation capabilities, the knowledge required for reconfiguration is a problem in practice. Recent studies have found that network data rates reached by novice users are only one-third



(a) Main types of fault/repair complaints received by UK ISPs [10].

(b) Major issues contributing to wireless network performance problems [11].

Figure 1.2: Recent studies have found that slow network connections are a major problem, caused mainly by faulty access links or faulty user devices.

of what expert users typically achieve, a phenomenon commonly referred to as the "wizard gap" [12]. In addition, a number of new industry studies cite that a primary challenge in managing cloud services is the lack of tools to identify the sources of performance bottlenecks and rapid troubleshooting [3, 13]. Furthermore, automation is a mandatory requirement for achieving highly-scalable cloud services, which presents a significant research challenge for the creation of comprehensive diagnostic solutions [13].

1.3 Network diagnosis

In general, detecting network faults and identifying the root causes is referred to as network diagnosis. Diagnosis of the root causes of network performance problems requires a methodical approach: first, to isolate the faulty segment of the network and second, to identify the exact cause of the problem.

Root causes of the network performance problems experienced by end-users can be traced to service provider servers, backbone networks, access networks, and UDs [14, 15]. Finding and fixing a network performance or configuration problem is a strenuous and complex task. It requires time, energy, and expertise to locate problems, and it is difficult for users to know whom to contact when the source and destination computers are located in different administrative domains. The main question network operators hear today is "my application is slow, what is wrong with the network? " To answer this question, the general solution is to install some specialised packet-sniffing hardware/software, spend a long time gathering data, and then find someone with the engineering expertise required to analyse the application's traffic. In enterprise environments, IT personnel have traditionally relied on network monitoring systems (NMSs) to detect the signs of network problems. However, due to the complexity of such systems and the difficulty of formalising the scope of the diagnosis task itself, diagnosis

has historically been a largely manual process requiring significant and expensive human input. This complexity means that most performance problems, in contrast to connectivity problems, remain an unsolved mystery.

The main emphasis of past research has been on analysing problems in core networks, access networks, and servers, since such issues can potentially affect a large number of users [16, 17, 18]. In contrast, performance bottlenecks at UDs have received little attention, mainly because of the complexities that arise with the large variety of devices [19]. Furthermore, the assumption that problems at the UDs typically impact only a few users has discouraged researchers from creating UD-specific diagnostic applications. Nevertheless, studies have found that many network problems occur at the UDs, and their cumulative impact is far more significant than faults in the core networks [20].

There are varying opinions on the primary cause of soft failures. Parameter misconfiguration in various protocol layers has been found to be a common issue causing performance bottlenecks. Often this is the result of the overly conservative default networking parameters supplied with almost all out-of-the-box operating systems [12]. Configuration changes that occur in the background when installing new applications can also cause soft failures unbeknown to the user. In addition, hardware problems [21], network interface card (NIC) driver issues [22], kernel level software problems, mismatches between system settings and the link [23], and protocol implementation errors [24] have commonly been found to cause soft failures.

Most common performance issues are often simple to correct, but harder to diagnose. As a result, most customer connection issues persist and stay as, not fully resolved, even after several attempts to correct them [10]. Most users experience severely degraded network performance, and computer networks are under-utilised because of unresolved network issues [12, 25]. The Internet2 performance initiative found that the median bandwidth in their 10 Gb/s backbone

in April 2010 was only about 3.05 Mb/s [26]. The diversity of user devices and access links, concerns about allowing remote access to diagnose their devices and the ever-increasing set of protocol parameters worsen the diagnostic difficulties.

1.3.1 TCP as a window to network behaviour

The content and behaviour of various types of protocol packet streams are constantly affected by the characteristics of end-to-end path elements. The transport layer, present in all communication systems, is tasked with end-to-end packet delivery. The most common transport layer protocol, Transmission Control Protocol (TCP) [27], dominates the Internet and private networks responsible for more than 90% of data transfers [28]. TCP sits towards the top of TCP/IP protocol stack and masks lower layers' behaviour to minimise their effects on the application layer. This unique position of the TCP in the protocol stack offers a window into the behaviour across all layers, which are otherwise impossible to observe from any other single vantage point [12]. The effects embedded into TCP packet streams, or "artefacts", can be used as an excellent source of information to remotely diagnose a performance bottleneck.

Analysis and inference of TCP packet traces is a sophisticated approach used for diagnosing extremely complicated network problems in highly specialised cases. Inferring from a packet trace is a cognitive process and mainly depends on the investigator's experience. Although this technique is comprehensive, most ISPs overlook the advantages because of the complexities of the process and the resource requirements.

1.3.2 Machine learning as a diagnostic tool

Machine learning is a scientific discipline that is concerned with the design and development of algorithms that allow computers to evolve behaviours based on

training data. Machine-learning techniques borrow heavily from statistical techniques such as probability theory, distribution analysis, and entropy theory. Machine learning specifically relies on training and cross-validation, which involves partitioning a sample of data into complementary subsets, performing the analysis on one subset, called the training set, and validating the analysis on the other subset, called the validation set or testing set. Cross-validation can provide an estimate of model accuracy.

Diagnosis algorithms that rely on machine learning can be grouped into two broad categories:

1. unsupervised learning, which identifies patterns in unlabelled data typically by clustering and detects unexpected outlier data points that might be indicators of failures, and

2. supervised learning, which uses labelled data of successful and failed states to learn which metrics are most correlated with failed states, or to identify signatures of recurrent problems from a database of known problems.

Signature-driven machine learning allows system administrators to identify repeated problems from a database of known problems. Signature-based approaches have wide applicability, because studies have shown that typically a half to as much as 90% of network failures are due to recurrent problems. Most research has centred on how to represent and retrieve signatures of recognised problems from the database of known problems. However, in such situations, unsupervised learning techniques are more applicable, because we do not know about new faults a priori and therefore will not have the labelled data sets suitable for supervised learning techniques.

Machine learning-based inference of TCP traces has been used in several security applications. A number of studies have used machine learning in root cause diagnosis of enterprise networks [29, 30, 31, 32, 33], access links [34, 35], networks

[36], and computer systems [37, 31], using inference of information such as user requests, event logs, and system calls. However, most of these methods have limited functionality or suffer from restrictions in scalability across platforms, access to the devices, and diagnostic capability. Our literature survey did not uncover any comprehensive, scalable, intelligent inference methods of TCP packet traces for automated network diagnosis.

1.4 Thesis motivation and aims

As discussed previously, network soft failures have a major impact on how end-users experience the quality of the services on offer today. Our literature survey found that over the years, significant attention has been given to the creation of systems and methodologies for diagnosing problems in core networks, access networks, backbones and servers. However, UDs pose the greatest challenge in soft-failure diagnosis, and this problem has received little attention for the creation of scalable, automated work flows.

In summary, this research is motivated by three main factors:

1. **Lack of comprehensive UD network performance diagnostic methodology:** Present attempts to automate the inference/diagnostic process follow a bottom-up design approach, where a particular algorithm is used to look for a specific anomaly in the packet trace. However, these tools fall short in their ability to generalise to new faults and to cope with the ever-increasing variety of user devices, and the large number of TCP variants.

2. **The potential of TCP traces for inferring root causes:** TCP traces have been extremely useful in inferring root causes of complicated network problems across many protocol layers without having direct access to the faulty section of the network. Analysis of TCP traces for failure diagnosis is among the most complex and sophisticated approaches in the network industry, yet

largely remains a manual process limited to the skills of a select few. The resources required for physical trace collection, analysis, and inference do not support the economics of a commercial network.

3. **Recent advances in machine learning techniques:** The machine learning concept, which has gained momentum in recent years, provides a foundation to learn from the known to automatically identify the unknown. However, the possibility of using machine learning-based inference of TCP traces to diagnose network performance problems of UD has not been previously explored. In addition, most studies focus on supervised machine learning where only the known issues can be identified. The ability to scale an automated system requires a more sophisticated approach to combine unsupervised learning with supervised learning to leverage the capabilities of both techniques.

Following on from these motivating factors, the aim of this research is to develop and analyse novel inference techniques based on machine learning to diagnose network soft failures using TCP traces. Using these inference techniques as the foundation, the research creates the basic building blocks of diagnostic systems which will perform automated fault diagnosis in various types networks. By evaluating the systems presented in this thesis using data from controlled networks and real-world live networks, the research demonstrates the capability and potential of such systems to automatically diagnose network soft failures and identify root causes.

1.5 Thesis contributions

The work presented in this thesis propose inference techniques based on machine learning that learn from hidden, embedded artefacts in the TCP traces due to network faults. These techniques encompass methods for characterising TCP traces,

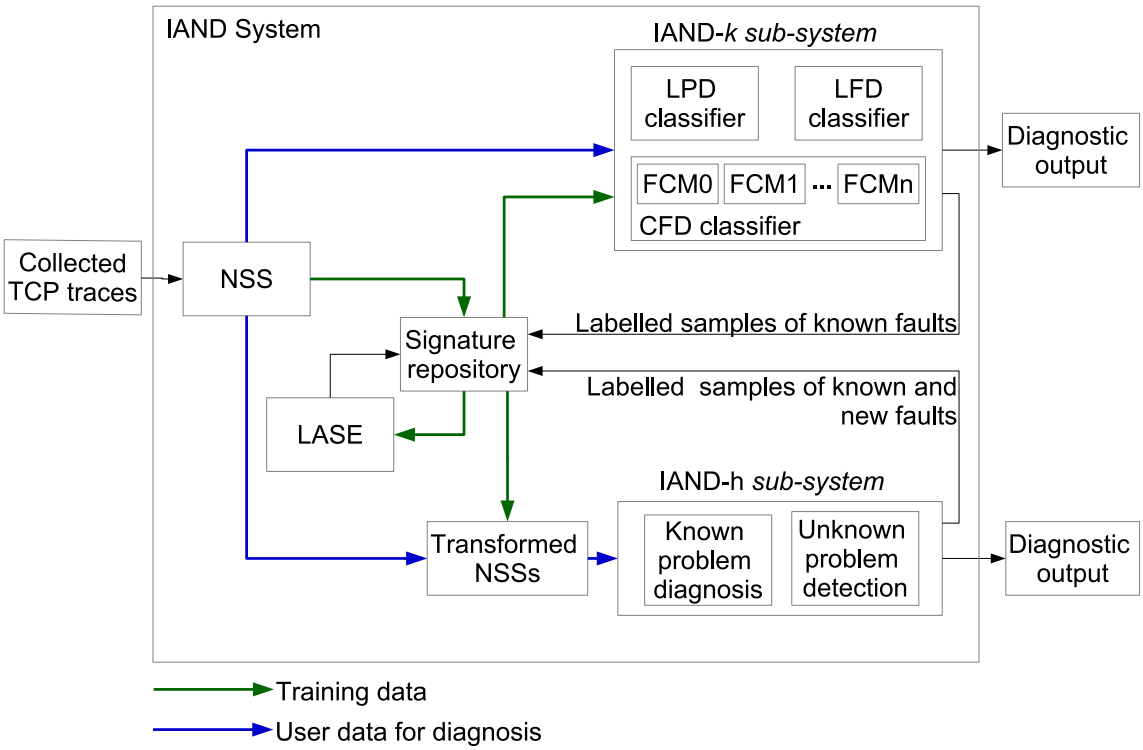


Figure 1.3: Intelligent Automated Network Diagnostic (IAND) system overview. The system consists of two sub-systems, IAND-*k* and IAND-*h*, each of which have unique and complementary capabilities and characteristics. This thesis specifically focuses on using the IAND system for diagnosing performance problems in UDs and detecting faulty links.

creating a standardised signature, reducing signature complexity, and pattern classification techniques to uniquely identify the signatures.

Using the inference techniques as the foundation, Intelligent Automated Network Diagnostic (IAND) System is proposed for automated diagnosis of network faults. As seen in Figure 1.3, the IAND system consists of two main complementary sub-systems, IAND-*k* for modular and easily scalable diagnosis of known faults, and IAND-*h*, a hybrid classifier for both known and unknown problem diagnosis with real-time detection capability. Both systems have two phases: 1. training phase uses data from previously identified faults to create detection models and 2. diagnosis phase, the trained systems are operational and unknown samples are sent through the system for a diagnosis. Table 1.1 shows a compari-

	IAND-*k*	IAND-*h*
Architecture	Using binary SVMs and a database of known problems for automated diagnosis	Uses multi-class classifier and a database of known problems for automated diagnosis
Scalability	Highly scalable with modular architecture	System re-training is required to scale
Accuracy	Previously unknown problems may lead to false positives, but fault specific "module tuning" leads to more accurate diagnosis when using NSSs	Can accurately recognise and group previously unknown problems. Diagnostic capability automatically improves over time. Accuracy improves when using FisherNSSs.
Diagnosis speed	Slower than IAND-*h*	Provides results in near real-time when using FisherNSSs
Extendibility	Automated process adds new samples to the known signatures repository. No new faults are added automatically	Can automatically extend the signature repository over time with new faults

Table 1.1: Comparison between IAND-*k* and IAND-*h* Characteristics

son between characteristics of IAND-*k* and IAND-*h* sub-systems. The system is deployed in a network as shown in Figure 1.4.

The research presented in this thesis specifically focuses on the creation of a system for detecting faulty access links and diagnosing the root causes of performance problems of UDs in highly-dynamic network environments. Diagnosing the root-causes of link problems is out of scope in this research and will be addressed in future work. However, the proposed techniques aim to follow a top-down design approach, which can be adopted for various types of networks to diagnose a wide range of root causes of performance problems, not limited to the examples given throughout the thesis. The system proposed in this research focuses on scalability, automation and modularity, as we believe these attributes will be the cornerstones of next-generation diagnostic systems.

The key contributions of the research are summarised as follows:

- The proposal of the concept of Normalised Statistical Signature (NSS), a min-

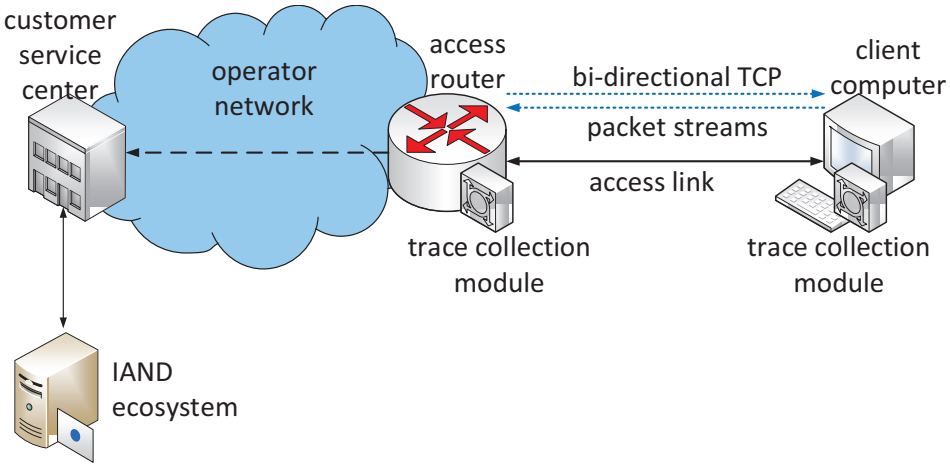


Figure 1.4: Deployment of the diagnostic system in a network environment.

imally invasive, robust method to remotely collect and characterise network faults using TCP trace statistics.

- The creation of a link-adaptive signature estimation (LASE) technique to dynamically generate signatures and avoid complexities that arise in rapidly changing network environments.

- The proposal on IAND system for automated diagnosis of network faults, specifically focusing on end-user device soft failures.

- The creation of fault classifier modules (FCMs) using support vector machine (SVM)-based pattern classification for identifying individual network faults. FCMs include a feature selection process that identifies those features with the most correlation to a particular fault.

- The creation and evaluation of the IAND-*k*, a modular and scalable diagnostic system for automatic diagnosis of end-user device problems (IAND-*k*UD). The IAND-*k* system is capable of automating the diagnosis of previously known problems.

- The creation and evaluation of the IAND-*k* for cloud computing (IAND-*k*CC) system for diagnosing user device bottlenecks in cloud environments.

- The proposal of EigenNSS and FisherNSS, two forms of signatures derived from the original NSS for complexity reduction and improved separability.

- The proposal and evaluation of IAND-*h*, a single hybrid classifier architecture with the capability of diagnosing both known and unknown faults with real-time detection capability.

- The creation of one of the largest, accurately labelled active TCP trace datasets in the research field with 1.2 million samples.

1.6 Thesis outline

This section provides an outline of the thesis content.

The thesis contains seven chapters starting with an **Introduction** to network failures and UD performance problems. The first chapter also provides an overview of the motivation for the study, the research aims and contributions.

Chapter 2, **Automated Network Diagnostics: An Overview** provides an in-depth analysis of background research and related concepts. We define some key concepts and then discuss existing diagnostic practices, anomaly detection techniques, network characterisation and complexity reduction methods. Next we discuss how anomaly detection techniques have been applied in diagnosis and various machine learning techniques before moving to a detailed analysis of why and how TCP has been utilised for network diagnosis.

Chapter 3 discusses **Statistical Signatures of Network Soft-Failures** and introduces the concept of normalised statistical signatures (NSSs) to characterise a TCP trace. This chapter then discusses the fault emulation and data collection methodology we have followed and introduces the data sets that have been used throughout this research. This chapter also introduces link-adaptive signature estimation (LASE), a method to easily generate NSSs to overcome the challenge of

dynamic and unstable links.

Chapter 4 introduces the **Fault inference and classifier** techniques using the NSSs and machine learning techniques and formulates the basic building blocks for the creation of more complex systems.

Chapter 5 proposes a proof of concept **automated inference system, IAND-*k*** using the fault inference modules introduced in the previous chapter for diagnosing UD problems and link problems. We specifically focus on link problem detection and exact UD fault identification. We analyse the performance of the system in real-world network environments with the datasets introduced in Chapter 3 and discuss the strengths and limitations of the system. We also propose an application of the IAND-*k* system in private cloud computing environments (IAND-*k*CC) and demonstrate the capability of the system to automate diagnosis.

Chapter 6 presents methods that extend the diagnosis of both **known and unknown soft failures** using IAND-*h*, a hybrid architecture that combines supervised and unsupervised machine learning methods. The chapter also introduces two additional signatures derived from NSSs, **EigenNSS and FisherNSS**, that reduce the complexity of NSSs and produce better overall performance.

The last chapter, **Conclusion**, summarises the body of work presented in the thesis and its novel contributions. We reflect upon our application of machine learning techniques to use TCP traces and the development of a scalable, automated diagnosis system for identifying UD problems and faulty links. We also discuss further avenues of research, and future directions and possibilities that arise from this work.

CHAPTER TWO

AUTOMATED NETWORK DIAGNOSTICS: AN OVERVIEW

This chapter provides a background to network diagnostic practices, common methodologies, and tools found in the research literature. Figure 2.1 shows an overview of the related work with a focus on combining machine learning (ML) with TCP trace analysis-based inference. These works span several decades dating back to the introduction and widespread popularity of TCP. There have been numerous applications and tools built for various aspects of networking using TCP, ML and combinations of the two. As indicated in the figure, there is a clear research gap in harnessing the benefits of both TCP trace analysis and machine learning for network soft-failure diagnosis. Investigation of the existing methods reveals the limitations of the current state-of-the art in creating a scalable and automated diagnostic system. This research gap provides the motivation for the work presented in this thesis.

2.1 Defining detection and diagnosis

In general, detecting network faults and identifying the root causes is referred to as network failure diagnosis.

Detection and diagnosis are not sharply separated in common speech. By the

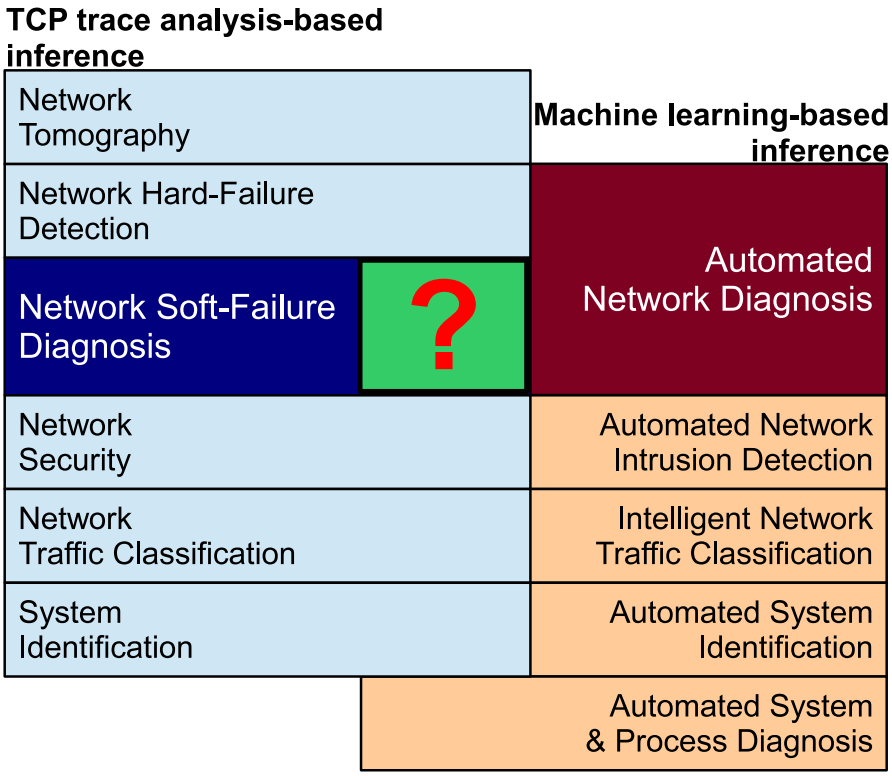


Figure 2.1: Overview of the related work found in the literature and the research gap.

phrase"detecting a problem" one often means two things: first, the confirmation that there is a problem at all and second, the verification of the nature or type of the problem itself. An example is to say that the Internet is slow because the ISP is throttling the connection after exceeding the package usage limits. The correct terminology in this example would be to say that an unusual behaviour has been detected (i.e., the Internet is slower than what is usually expected) and the diagnosis is that, for example, the ISP is throttling the connection speed due to exceeded usage limits. Detecting that the Internet is slow does not necessarily mean that there is a problem with the connection itself. Nevertheless, by simply looking at the symptoms at a high level, it is impossible to tell if there is a serious problem with the router, or the user must wait until the new billing cycle for the ISP to re-establish guaranteed speeds. Therefore, if unusual behaviour is detected, a more thorough diagnosis has to be conducted in order to find out if there is actually a

problem and the root cause of it. Since the terms “detection“ and “diagnosis“ often carry an implicit duality and appear to be over-used in common speech, they must be precisely defined before being used in an engineering system.

Detection basically means to identify symptoms indicative of something unusual in the network performance. In the context of the proposed framework, the detection happens at the user level when the user is experiencing less than normal network quality. This may be a slower download or an interrupted video stream. In most cases, users use widely-available on-line bandwidth and latency estimation tools [38, 39] to confirm their suspicions.

Upon detecting a suspected issue, diagnosis means to investigate the root cause of the detected symptoms. The output of the diagnosis may be that there is, in fact, no problem at all. However, assuming that there was an underlying condition causing the slow network, the output of the diagnosis will be the most likely cause. Usually, after the diagnosis of the root cause is completed, certain corrective actions must be performed in order to resolve the problem. Resolution of the faults and the mechanisms to automate that process have their own associated complexities, and this is an entirely different area of research. The broad range of challenges in the resolution process warrants a detailed investigation and a comprehensive automated solution. This is beyond the scope of the present research.

2.2 Detection and diagnosis of network failures

Failure diagnosis is one of the major challenges that home users and network administrators face today. The problem is complex because there are various components, which collaborate to realise a particular service, and these components belong to different functional domains and physical locations. Troubleshooting network problems can be a frustrating, expensive, and time-consuming experience. Today’s networks are so complicated that the point of failure may be virtu-

ally anywhere. Worse still, the network could contain multiple points of failure, resulting in confusing symptoms that are hard to diagnose.

End-to-end delivery of a network service can be loosely divided into a number of segments: the core and access networks, user devices and applications. The core and access networks always belong to the ISP or organisation and user-devices and applications in most cases belong to the end-user. Diagnosis of core and access network problems has received much attention in the research community. The main emphasis of past research has been on diagnosing problems in core, access network components and back-end servers, since such issues can potentially affect a large number of users [16, 17, 18]. In contrast, performance bottlenecks at UDs have received little attention, mainly because of the complexities that arise with the large variety of devices [19]. Furthermore, the assumption that problems at the UDs typically impact only a small number of users has discouraged researchers from creating UD-specific diagnostic applications. Nevertheless, recent studies have found that many network problems occur at the UDs, and their accumulated impact is far more significant than faults in the core networks [20].

2.2.1 Traditional practices - Hard failure diagnosis

Hard failures which result in total loss of connectivity are usually easier to detect. Core and access networks are constantly checked with network monitoring systems (NMSs) [40, 41] to detect failures and performance issues. ISPs spend millions of dollars and employ hundreds of engineers to investigate and resolve network issues in the ISPs' administrative domains. Resolution of a hard failure is normally assisted by NMSs and systematic checking of network and pre-configured alarms to pinpoint the exact point of failure. If the point of failure is at the user's device itself, there is no other option but to communicate with the user through other channels to identify and resolve the device failure. This process is

comparatively straight-forward and difficult to replace with automated systems, particularly when diagnosing hard failures in UDs.

2.2.2 Traditional practices - Soft failure diagnosis

In comparison, soft failures, which manifest as degraded network performance or the loss of network bandwidth, are harder to detect and diagnose [5, 6, 7, 8]. Diagnosing soft failures often requires experienced network engineers and time-consuming investigations to methodically eliminate each segment of the network as a potential cause until the exact problem is detected. These manual investigations of possible network bottlenecks are expensive, and in many cases, the costs are only justified in a core network or an access network affecting hundreds or thousands of users. Although problems in UDs have been found to be a major bottleneck, very little attention has been given to resolving these issues. Traditionally, resolving soft failures is a function of customer support or IT support, and involves an escalation process starting from a call centre operative who conduct limited initial testing. The process escalates depending on the complexity, and typically concludes with senior engineers spending valuable time investigating and resolving the issue [42].

In IT services environments, the problem management process (defined, for example, in Information Technology Infrastructure Library - ITIL [43]) describes the steps by which problems are reported, diagnosed, and solved. A typical sequence is for a problem ticket to be opened by a call to the customer help-desk, or by an alert generated by a monitoring system. This is followed by some basic diagnosis by first-level support personnel based on, for example, well-documented procedures. Simple issues such as incorrect network interface settings can generally be handled here without progressing further. If the problem needs additional investigation, it is passed to second-or third-level personnel, who are typically system administrators or service engineers with more advanced skills and knowl-

edge. They often start with vague or incomplete descriptions of problems (e.g., "Application is running very slowly," or "Web pages are not loading fast enough") which require significant investigation before the cause and solutions are found. In the context of high-level support, administrators often consult monitoring tools that provide some specific system indicators, and then log in to the server to collect additional detailed information using system utilities. In day-to-day problem management, this investigative process is often the most time-consuming and expensive task for engineers. It is difficult to automate, and requires field experience and expert knowledge. Unlike the first-level support protocols, there is hardly any well-documented procedure one can refer to during this advanced problem-management process. Engineers usually rely on their own knowledge and experience to diagnose the root cause of the problem. Because of the complexity of the problems, it is a significant challenge and overhead to create useful documentation about this process which can be used by others. Particularly in an environment where support teams are globally distributed, the creation and sharing of this knowledge is a major challenge. Such challenges could lead to human error and consequently large-scale mis-diagnosis when appropriate training and skills are not in place.

Providing the highest quality of support at the first point of contact is prohibitively expensive and has become a major source of customer dissatisfaction. In enterprise and office environments, prolonged delays in soft-failure resolution result in losses in productivity, under-utilised network resources, and employee frustration [25].

2.3 Anomaly detection

With the increasing complexity of the networks and ever-increasing user expectations, the traditional practices are becoming obsolete, expensive, and counter-

productive. The automatic identification of anomalies in the behaviour of network infrastructures as a means to automate network diagnosis is an area that has recently aroused significant interest in both the commercial and research communities. Detecting the existence of an anomaly and its nature is the most complex task in network soft failure diagnosis. A comprehensive diagnostic application requires functionality that goes beyond simply detecting anomalies to identify the exact anomaly and the associated root causes. Once detected, the unique identification of the faults requires extending the anomaly detection to pattern identification. However, in the research literature, these two concepts are tightly interwoven and in most cases, difficult to differentiate. A review of the existing landscape of anomaly detection techniques provides the best starting point in identifying the best avenues to the development of a comprehensive automated network diagnostic application. Furthermore, the anomaly detection techniques discussed in this section are the foundation of most inference-based applications shown in Figure 2.1.

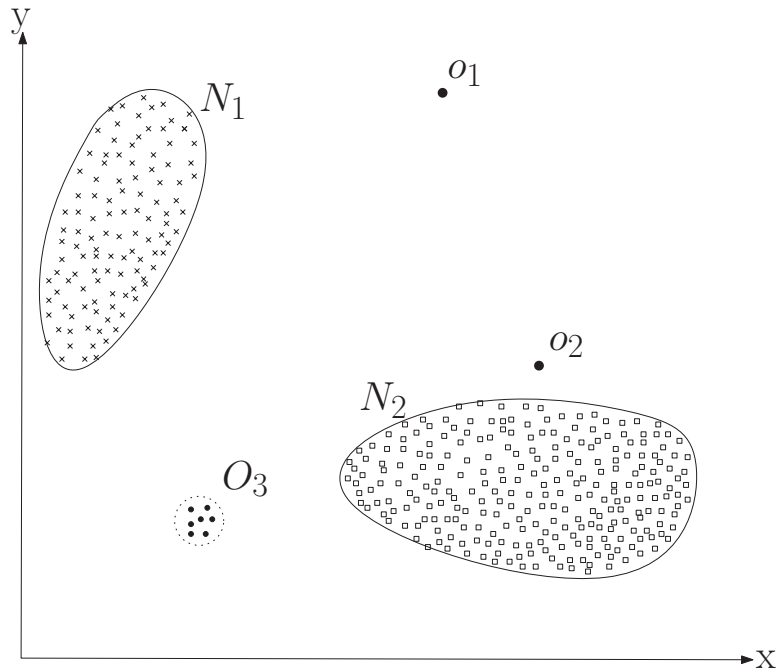


Figure 2.2: A simple example of anomalies in a 2-dimensional data set.

Anomalies are defined as patterns in data that do not conform to a well-defined notion of normal behaviour [44]. Hence, the problem of anomaly detection in any system implies the existence of a concept of normality. The notion of "normal" is usually provided by a formal model that expresses the relations between the fundamental variables involved in the system dynamics. Consequently, an event is catalogued as anomalous because its degree of deviation in relation to the profile of characteristic behaviour of the system, specified by the model of normality, is sufficiently high. Figure 2.2 illustrates anomalies in a simple 2-dimensional data set. The data set has two normal regions, N1 and N2, since most observations lie in these two regions. Points that are sufficiently far away from the regions, e.g., points o1 and o2, and points in region O3, are anomalies.

2.3.1 Network attributes for defining models

Like many other systems, a network is a complicated assembly of components with complex relationships that can be studied from several points of view. Defining ordinary behaviour in the context of an entire network could lead to failure. When the problem of anomaly detection is posed within this context, it is necessary to determine which network attributes, or facets, the normal behaviour model refers to. Thus, detectors can be classified according to the particular aspect being modelled. Network attributes used to create anomaly detection models can be classified in accordance with the organisation provided by the scheme shown in Figure 2.3.

Network traffic flow

Traditionally, traffic has been the unique network-related aspect addressed by detection systems. The method that obtains models of normality by analysing network traffic is termed "flow analysis" (see Figure 2.3), and is characterised by the study of the temporal evolution of several measures related to traffic flows. As

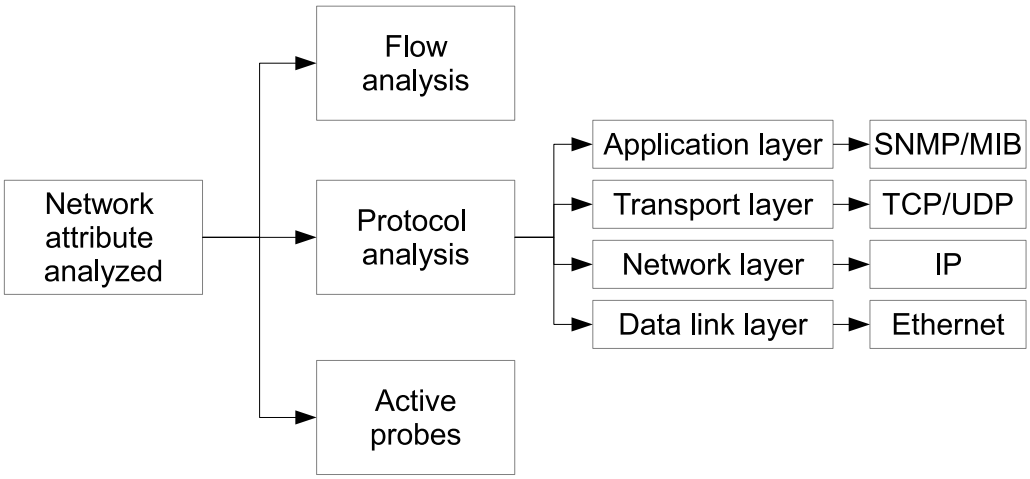


Figure 2.3: Network attributes used to create models in anomaly detection.

example, Cleveland et al. [45] identified packet flows by capturing the IP headers of a select set of packets at different points in the network, while Caceres et al. [46] and Aukia et al. [47] used link-level information captured at the routers to determine flow levels and utilisation. These types of flow level characteristics have been extensively used in anomaly detection, especially in network security and traffic classification. For example, Cabrera et al. [48] used the total number of connections initiated during a given time interval as a measure to classify traffic flows in detecting denial of service (DoS)-type network attacks. Zhang et al. [49] introduced a multi-dimensional box plot method for short-time scale traffic (MBST) which aimed to detect many types of hidden network attacks by observing how traffic features changed over time. In traffic classification, many authors, including Hong et al. [50], John et al. [51], and Dahmouni et al. [52], have used network traffic flow analysis as a method for identifying traffic types and anomalous behaviour.

Flow analysis usually requires sophisticated network sampling techniques for packet filtering as well as specialised hardware at the network devices to do IP packet lookup and capture. Although the data obtained from the traffic flow method is a rich source of information for network diagnosis, the hardware requirements for this measurement method make it difficult to use in practice.

Protocol analysis

Network protocols are one of the fundamental pieces of networking, as they are the foundation of all information interchange throughout the network. Each protocol is carefully designed to support a specific facet of the communication process, so that devices and applications use it according to an established set of formal rules. Nevertheless, protocol specifications usually suffer ambiguities that enable use in several ways that go beyond those for which they were developed, resulting in severe security implications. In addition to this, some implementations are not fully in conformity with the recommendations (for example, a list of common implementation problems in TCP is provided in RFC-2525 [53]).

In order to detect protocol anomalies, several approaches have focused on the modelling of protocol usages [54, 55, 56]. From this point of view, it is possible to classify systems according to the network layer modelled:

1. Data link (Ethernet, token ring, etc.)
2. Network (IP in most cases)
3. Transport/Control (TCP, UDP, RTP, ICMP, etc.)
4. Application (HTTP, DNS, Telnet, FTP, SSH, POP, SMTP, etc.)

Network management protocols provide information about network traffic statistics. These protocols support variables that correspond to traffic counts at the device level. This information from the network devices can be passively monitored. The information obtained may not directly provide a traffic performance metric, but can be used to characterise network behaviour and, therefore, can be used for network anomaly detection. Using this type of information requires the cooperation of the service provider's network management software and more importantly, may require privileged access to user devices. However, these protocols provide a wealth of information that is available at fine granularity. Sim-

ple network management protocol (SNMP) is a such protocol which provides a mechanism to communicate between the manager and the agent. The SNMP manager can collect management data that is provided by the SNMP agent, but does not have the ability to process this data. The SNMP server maintains a database of management variables called the management information base (MIB) variables [57]. Every network device has a set of MIB variables that are specific to its functionality. MIB variables are defined based on the type of device as well as on the protocol level at which it operates. For example, bridges, that are data-link layer devices, contain variables that measure link-level traffic information. Routers that are network-layer devices, contain variables that provide network-layer information. These variables contain information pertaining to the different functions performed by the network devices. Although this is a valuable resource for network management, we are only beginning to understand how this information can be used in problems such as failure and anomaly detection [16].

Network Probes

Network probes are specialised tools such as ping and traceroute [18] that can be used to obtain specific network parameters such as end-to-end delay and packet loss. Probing tools provide an instantaneous measure of network behaviour. These methods do not require the cooperation of the network service provider. However, it is possible that service providers could choose not to allow ping traffic through their firewall. Furthermore, the specialised IP packets used by these tools need not follow the identical trajectory or receive the same treatment by network devices as do the regular IP packets. This method also assumes the existence of symmetric paths between given source-destination pairs. On the Internet, this assumption cannot be guaranteed. Therefore, performance metrics derived from such tools can provide only a coarse-grained view of the network. As a result, the data obtained from probing mechanisms may be of limited value for the purpose

of anomaly detection.

In addition to these attributes, other network features such as topology, systems logs [58], user reports, and mixtures [59, 36] of the above may be features in a network diagnostic system.

2.3.2 Analysis scale

Although it is not always well identified, the notion of scale of analysis is implicit in every anomaly detection method proposed to date. In order to achieve our objective of obtaining accurate models of the normality of a network, deep comprehension of the phenomena involved in its dynamics is required. Nevertheless, there are several points of view, or dimensions, from which to carry out such a study.

In the research literature on network anomaly detection, three analysis levels can be clearly defined. These analysis levels dictate the kind of detectable anomalies and the types of application in which they could be used.

1. Microscale - Methods based on the analysis of low-level features. According to the feature modelled, examples of low scales of analysis are:
    - Analysis of individual packets, in the case of protocol analysis.
    - Traffic analysis during short periods of time (e.g. < 1s).
    - Traffic analysis destined to a specific service within a host.
2. Mesoscale - Methods based on the analysis of medium-level features. According to the feature modelled, examples of medium scales of analysis are:
    - Analysis of connections or packet streams.
    - Traffic analysis from several seconds to minutes.
    - Analysis of traffic destined to a specific host within the network

3. Macroscale - Methods based on the analysis of high-level features. According to the feature modelled, examples of high scales of analysis are:

    - Simultaneous analysis of several connections and event correlation within the whole network.
    - Traffic analysis during hours, days, months, and so on.
    - Traffic analysis across all the hosts within the whole network.

The notion of scale can be identified when traffic analysis or protocol analysis is used for anomaly detection. For an example, the inspection of individual packets can be viewed as a low-scale analysis, in contrast to the study of packet streams (medium scale), or even the simultaneous evaluation of different connections within the whole network (high scale).

The importance of the scale of analysis in anomaly detection methods lies in the fact that some anomalies are only observable at certain scales. In a diagnostic application, it is also important to consider the inherent requirements of identifying large-scale problems versus user-level problems.

2.4 Characterising network behaviour for fault diagnosis

The previous section discussed the network attributes and scales that can be used for anomaly detection. This section discusses how those attributes have been utilised to characterise the behaviour of the network towards creating a "signature" of the anomaly. These signatures can be used to uniquely identify the anomaly and provide the basis of our proposed system.

There are numerous network applications that create signatures for automated identification. These applications include: (i) online traffic classification and flow

identification [60, 61, 62, 52, 63, 64, 65, 66, 67], (ii) traffic monitoring [68, 69], (iii) intrusion and attack detection [70, 49, 71, 72], (iv) source identification and system fingerprinting [73, 61, 74, 75], (v) connectivity failure detection [76, 77, 33, 42], and (vi) soft-failure detection [36, 16, 78, 79, 80]. Table 2.1 has a critical comparison of such signatures and their limitations for UD diagnosis.

Network signatures generated from flow-based characteristics have been commonly used in online traffic classification and flow identification. Roughan et al. [81] used traffic flow data such as flow duration, bytes per second and packet size collected through passive measurement of traffic flows for QoS mapping in IP traffic. In contrast, Nguyen et al. [63] proposed an IP traffic classification technique based on short sub-flow features such as inter-packet arrival interval, inter-packet length variation and IP packet length instead of full flow characteristics. But et al. [67] used derived flow characteristics such as ratio between BitTorrent packets to total packets within a flow, ratio of small status packets to total packets, ratio of data packets to total packets and payload size standard deviation to identify BitTorrent traffic. In an another application, Branch et al. [66] have used mean packet length, autocorrelation and the ratio of data transmitted in either direction of a bi-directional flow to identify variable rate VoIP traffic flows.

Taku et al. [82], Hajji [68], and Thottan et al. [16] have used signatures created using IP flow related MIB variable data, number of various types of packets, number of TCP connection requests, port numbers, packet size sequence and traffic throughput in network fault detection and Zhang et al. [49] in intrusion-detection systems. These applications mainly aim to raise alarms when the overall traffic flow pattern in the network shows abnormalities. Flow-based signatures created in above studies have been designed for passive monitoring applications and fundamentally unsuitable for a UD diagnosis application that runs on-demand when a user is experiencing a network performance problem. The signatures are limited due to the small number of features that have been selected with the knowledge

of the specific behaviour being detected. A scalable characterization of network faults using a single signature however requires a much broader feature set as different faults can impact different subsets of features which we have no way of pre-determining.

Another common approach when creating signatures for diagnostic purposes is to use the system logs from the device itself, or the "reports" compiled by the user. Both Aggarwal et al. [36] and Reidemeister et al. [77], for example, created signatures incorporating internal system logs, while Sihyung et al. [20] used user-reports to build the fault signatures. The use of the system logs not only raises privacy concerns in a public network, but also requires the network operators to gain privileged access to the UDs. Although practical for expert users, reports from average users with no specific network knowledge can be unreliable. Although system logs and user reports can provide valuable information to enrich the fault signatures, we believe the challenges with privacy, reliability, and scalability far outweigh the benefits such signatures offer to a diagnosis system.

Communication protocols are another common source of information to create network signatures. As shown in Table 2.1, popular protocols such as IP, TCP, UDP, and HTTP are often used as they are supported by most devices. While Dahmouni et al. [52], Manikopoulos et al. [70], and Wolfgang [74] extract features from multiple protocols, some studies such as by Gomes [73] and Chen et al. [72] have limited their features to a single protocol. These proposed signatures have been created to detect very specific network phenomena, and the features have been chosen with prior knowledge of how each event affects the features. Consequently, the signatures are limited to a handful of features. In the case of UD soft-failure diagnosis, the signatures must be able to effectively and uniquely characterise not only large numbers of faults, but also any new types of faults. Hence, such signatures with limited feature sets can affect the ability to capture the information needed to generalise the signatures.

Application	Studies	Source of information	Features	Data collection	Limitation for UD diagnosis
Traffic classification	Roughan et al. [81]	Traffic flow data	Flow duration, bytes per flow, packets per flow, rms/ave. packet size	Passive monitoring of live online traffic	Limited feature set, on-demand monitoring of traffic flows from user is difficult, portability across networks.
	Dahmouni et al. [52]	TCP and UDP/IP header data of successive flows	First-order Markov chain parameters of control packet sequence	Passive or active monitoring of online traffic	Dependency on TCP flags, creating and collecting multiple flows is difficult on-demand.
	Nguyen et el. [63]	Short sub-flow features	sub-flow inter-packet arrival interval, inter-packet length variations and IP packet length minimum, maximum, mean and standard deviation	Passive monitoring of live online traffic	Limited feature set to capture wide range of network anomalies. Feature set is geared to characterize traffic types as opposes to normal vs abnormal behaviour. optimal sliding classification window size (N) might change from fault to fault.
	But et al. [67]	BitTorrent, FTP flows	Ratio between BitTorrent packets to total packets, small status packets to total packets, data packets to total packets and payload size standard deviation	Passive monitoring of live online traffic	Specific to detect BitTorrent flows. Derived four features are not scalable to detect wide range of faults.

	Branch et al. [66]	VoIP traffic flows	Mean packet length, autocorrelation and the ratio of data transmitted in either direction	Passive or active monitoring of live online traffic	Specific to detect VoIP flows. Feature set is geared to characterize traffic types as opposes to normal vs abnormal behaviour.
Network attack / intrusion detection	Zhang et al. [49]	Traffic flow data, IP protocol data	Source and destination IP, port. Packet and flow sequence.	Passive monitoring of live online traffic, anomaly injection	Limited feature set, features are not fault-dependent, passive monitoring is not suitable.
	Manikopoulos et al. [70]	Protocol traffic data	IP and UDP packets, length, data bytes, rate.	Passive monitoring of test-bed data	Limited feature set, passive monitoring.
	Chen et al. [72]	TCP protocol data	Tokens extracted from TCP payload using decision rules.	Passive monitoring of live data	Decision rules are not scalable for use in automated systems.
Source identification and system fingerprinting	Gomes, J. [73]	IP protocol data	Packet length, MTU, derived entropy	Passive monitoring of live data	Limited feature set.
	Wolfgang, M. [74]	TCP, ICMP protocol data	Responses received from the devices to the probes sent.	Active probing of the devices	Limited feature set.
Connectivity and soft failure	Lee et al. [20]	Passive TCP-HTTP data, active ping probes, hardware status, user reports	TCP SYN- SYN/ACK pairs, HTTP request-error pairs, problem duration, NIC on-off, user descriptions	Passive and active monitoring, user reports	Access to UDs and user reports is limited, limited feature set.
	Taku et al. [82]	Traffic flow data	Traffic flow level eigenvector	Simulation data	Limited feature set, not proven in real networks.

Aggarwal et al. [36]	TCP and UDP protocol data, Host ID data, Router/modem information	Internal router configuration parameters, TCP and UDP traffic features,	Emulated faults in network testbed	Requires special background process at the client, requires privileged device information,
Reidemeister et al. [77]	Historic log files	Clusters extracted from log files contents	Fault injection in to a test-bed	Requires a specific log format in multiple devices, has to be custom-built for a particular network, requires device access.
Thottan et al. [16]	IP flow related MIB variable data	IP In Received, IP In Delivered, IP Out Requests	Passive monitoring in multiple networks	Designed for active monitoring of network failures, device-side MIB information needs privileged access, insufficient feature set.

Table 2.1: Comparison of commonly found types of network signatures used in various applications

The source of information for the signature, attributes or features, collection methods, and portability of signatures across networks vary with the particular application of the signature. Consequently, many of the signatures in the literature do not offer a comprehensive method for characterising UD soft failures. Table 2.1 shows a summary of different types of network signatures and their limitations in the context of UD, soft-failure characterisation, especially for utilisation in an automated, scalable system. Furthermore, most of these studies train and evaluate their systems using duplicate data sets or data collected at the same network location with little diversity. This overlooks the effects that the network can have on the signature. Therefore, we propose a more comprehensive signature to characterise UD failures and analyse how different network properties affect such a signature.

2.5 Complexity reduction

This sub-section presents a summary of the different types of dimensionality reduction techniques used by other researchers, and their limitations. Network signatures generated from flow-based characteristics for traffic classification such as those proposed by Zhang et. al. [49] contain large amounts of data as they are collected from one of China's seven major backbone networks. The complexity of their dataset was reduced by simply capturing packet headers, which surprisingly still contain excessive amounts of data for a whole day. For further complexity reduction, each day is divided into 24 hours, and data is only captured in the first minute of each hour. Due to the on-demand requirement, this reduction method is deemed unsuitable for our application.

Clustering algorithms are commonly used in reducing the complexity of a large data set. Vaarandi et al.proposed a density-based approach for clustering [83] to reduce the amount of data (signatures from system log files) required

by a support person to evaluate the behaviour of the system. In density-based clustering, clusters are usually defined as areas of higher density than the remaining data set. The algorithm has three steps; (i) data summation, (ii) building cluster candidates, using the summary information collected, and (iii) cluster selection based on the candidates. This method not only clusters data into regions based on frequent patterns from the log files, but also extracts the static parameters that are unique to the system.

In other domains, Fisher's Linear Discriminant (FLD) [84] is used to reduce the dimensionality of image data sets. This method is also very versatile in overcoming inconsistencies and variances in an image (e.g. lighting variations in facial recognition). FLD provides linear class separation in huge data sets, which helps to simplify the classification process.

2.6 Methods of anomaly detection and diagnosis

In this section, we review the most commonly-used network anomaly detection methods and automated diagnosis techniques. Using the models created on the basis of previously discussed attributes, these methods provide the basis for detecting anomalous from normal behaviour and identifying the anomalies.

Figure 2.4 shows the most common methods of anomaly detection, such as expert rule-based approaches, signal processing and statistical analysis methods, and machine learning-based techniques. Many existing network-related applications use these techniques to detect abnormalities from the expected behaviour, although slight modifications in these techniques are often needed when adopting them to uniquely identify the anomaly.

2.6.1 Expert rule (specifications)-based systems

Early work in the area of network fault or anomaly detection was based on expert systems. In expert systems, an exhaustive database containing the rules or specifications of behaviour of the faulty system is used to determine if a fault occurred [85, 86, 87, 88]. Rule-based systems are too slow for real-time applications and are dependent on prior knowledge about the fault conditions on the network [89]. The identification of faults in this approach depends on symptoms that are specific to a particular manifestation of a fault. Examples of these symptoms include excessive utilisation of one packet type, the number of open TCP connections, and total throughput exceeded. These rule-based systems rely heavily on the expertise of the network manager and do not adapt well to the evolving network environment. Thus, it is possible that entirely new faults may escape detection.

Specifications or rules are provided by using any kind of formal tool or through manual coding. For example, finite state machines [90] model alarm sequences that occur during and prior to fault events. A probabilistic finite state machine

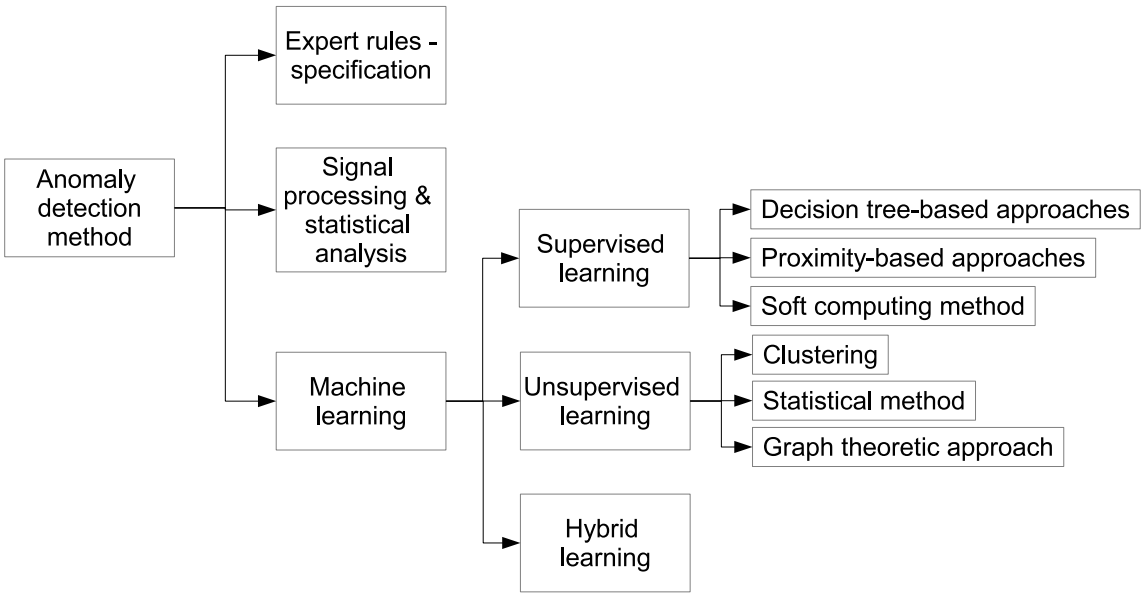


Figure 2.4: Methods of network anomaly detection.

model is built for a known network fault using history data. State machines are designed with the intention of not just detecting an anomaly but also possibly identifying and diagnosing the problem. The sequence of alarms obtained from the different points in the network is modelled as the states of a finite state machine. The alarms are assumed to contain information such as the device name as well as the symptom and time of occurrence. The transitions between the states are measured using prior events [91, 92, 93].

In [87], the authors describe an expert system model using fuzzy cognitive maps to overcome this limitation. Cognitive maps can be used to obtain an intelligent modelling of the propagation and interaction of network faults. Case-based reasoning is an extension of rule-based systems [88]. It differs from fuzzy cognitive maps in that, in addition to only rules, a picture of previous fault scenarios is used to make the decisions. A picture here refers to the circumstances or events that led to the fault. In order to adapt the case-based reasoning scheme to the changing network environment, adaptive learning techniques are used to obtain the functional dependence of relevant criteria, such as network load, collision rate, etc., to previous trouble tickets [94]. The trouble-ticketing system is used to perform two functions: prepare for problem diagnostics through filtering; and infer the root cause of the problem. Using case-based reasoning for describing fault scenarios also suffers from its heavy dependence on past information. Furthermore, the identification of relevant criteria for the different faults will, in turn, require a set of rules to be developed. In addition, using any functional approximation scheme, such as back propagation, causes an increase in computation time and complexity. The number of functions to be learned also increases with the number of faults studied.

2.6.2 Statistical analysis

As the network evolves, each of the methods described above requires significant recalibration or retraining. However, using statistical approaches, it is possible to continuously track the behaviour of the network. Statistical analysis has been used to detect both anomalies corresponding to network failures [95], and network intrusions [96]. Interestingly, both cases make use of the standard sequential change point detection approach. The flooding detection system, proposed in [96], uses measured network data that describes TCP operations to detect SYN flooding attacks. In [95] the author used SNMP MIB variables with varying statistical characteristics to design a failure detection system that looks for sequential changes in time-series data using the non-parametric cumulative sum (CUSUM) method [97]. In [16] the author introduced a statistical signal processing technique based on abrupt change detection in MIB variables by assuming traffic variables are quasi-stationary. At the same time, the performance of most statistical analysis-based methods requires methods to quantify the bursty behaviour of the variables to increase the optimality of statistical methods.

2.7 Machine learning

Machine learning (ML) is a branch of artificial intelligence, and concerns the construction and study of systems that can learn from data. In network anomaly detection and fault diagnosis, constructing models of the normal behaviour of a system consists of observing the system while it is working under normal conditions, and applying machine-learning techniques in order to obtain a model that: (i) explains, by means of a simple representation, the amount of data observed; and (ii) is able to extrapolate to other, non-observed situations. ML can also be used to create models of the anomalies using existing anomaly data, which could then be used to uniquely identify the anomaly in an automated system. Many

different types of models exist for ML and Figure 2.5 shows an overview of some of the common techniques.

2.7.1 Supervised machine learning

Supervised learning is the ML task of inferring a function from labelled training data. The training data consist of a set of training examples. In supervised learning, each example is a pair consisting of an input object (typically a vector) and a desired output value (also called the supervisory signal). A supervised learning algorithm analyses the training data and produces an inferred function, which can be used for mapping new examples. An optimal scenario will allow for the algorithm to correctly determine the class labels for unseen instances. This requires the learning algorithm to generalise from the training data to unseen situations in a "reasonable" way.

Statistical methods

Most basic methods of supervised ML are based on statistical methods for constructing probabilistic data models through the use of applied statistics and probability theory. Statistical techniques fit a statistical model (usually for normal behaviour) to the given data and then apply a statistical inference test to determine if an unseen instance belongs to this model or not. Instances that have a low probability to be generated from the learnt model, based on the applied test statistic, are declared as anomalies. For example, detecting outliers based on regression analysis, especially reverse search, and direct search [98] can be considered a relatively straight-forward statistical technique. More advanced regression methods based on support vector regression and particle swarm optimisation algorithms are given in [99] for pattern analysis of network intrusion detection. [100] and [48] also use statistical model-based anomaly detection to identify network intrusions. The first study uses a user-activity monitor as the source of information,

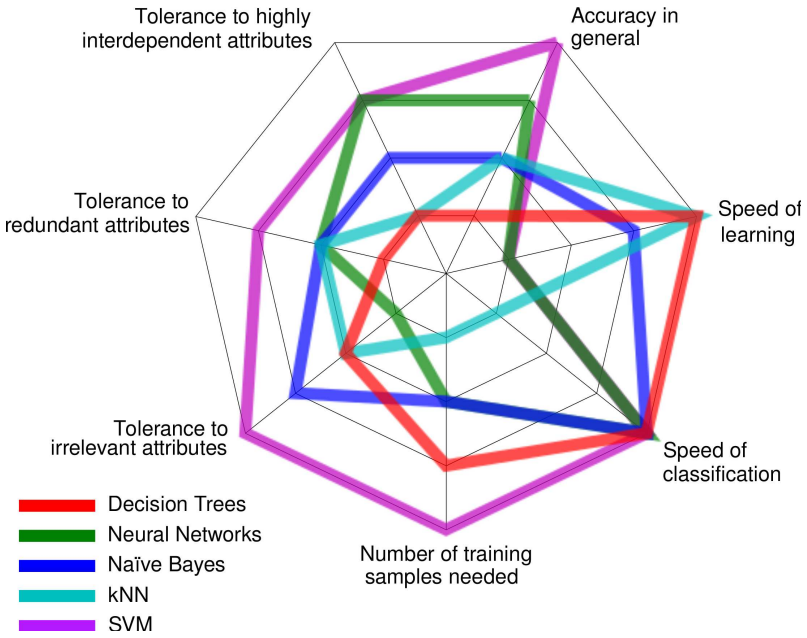


Figure 2.5: Comparison of machine learning algorithms.

while the second study uses a network traffic model.

Classification-based methods

Classification [101, 102] is used to construct a model (classifier) from a set of labelled data instances (training), and then classify a test instance into one of the classes using the learnt model (testing). Classification-based anomaly detection techniques operate in a similar two-phase fashion. The training phase learns a classifier using the available labelled training data. The testing phase classifies a test instance as normal or anomalous using the classifier. Classification-based anomaly detection techniques operate under the general assumption that normal and anomalous classes can be learnt in the given feature space.

Based on the labels available for the training phase, classification-based anomaly detection techniques can be grouped into two broad categories: multi-class and one-class anomaly detection techniques. Multi-class classification-based anomaly detection techniques assume that the training data contain labelled instances belonging to multiple normal classes [103, 104]. Such anomaly detection techniques

construct a classifier to distinguish between each normal class against the rest of the classes. One-class classification-based anomaly detection techniques assume that all training instances have only one class label. Such techniques construct a discriminative boundary around the normal instances using a one-class classification algorithm, e.g., one-class support vector machines (SVMs) [105], one-class kernel Fisher discriminants [106].

Decision tree-based approaches

These approaches begin with a set of cases or examples, and create a tree data structure that can be used to classify new cases. Each case is described by a set of attributes (or features) that can have numeric or symbolic values. Associated with each training case is a label representing the name of a class. Each internal node of the tree contains a test, the result of which is used to decide what branch to follow from that node. For example, a test might ask "is $x > 4$ for attribute x ? ". If the test is true, then the case processes down the left branch, and if not it follows the right branch. The leaf nodes contain class labels instead of tests. In classification mode, when a test case (which has no label) reaches a leaf node, a decision tree method such as C4.5 [107] classifies it using the label stored there. While decision-trees [108] are not always the most competitive classifiers in terms of prediction, they enjoy the crucial advantage of yielding human-interpretable results, which is important if the method is to be adopted by actual network operators.

Decision trees were commonly used in network-related machine learning in the early days. In [29] and [109], the authors used a decision-tree learning approach based on C4.5 and Min Entropy for diagnosing failures in large Internet sites. Other researchers [36] also used C4.5 and proposed "NetPrints", a system that leverage shared knowledge in a population of users to diagnose and resolve misconfiguration in home networks, while [110] proposed a TCP throughput prediction system. These studies showed decision trees are successful when the feature set is limited. However, their drawbacks are their time complexity and they

tend to produce false positives when working with larger feature sets and noisy samples.

Perceptron-based techniques

These well-known classification algorithms are based on the notion of perceptron introduced in [111]. A single layered perceptron can be briefly described as follows: If x_1 through x_n are input feature values and w_1 through w_n are connection weights/prediction vectors (typically real numbers in the interval [-1, 1]), then perceptron computes the sum of weighted inputs: $\sum_i x_i w_i$ and output goes through an adjustable threshold: if the sum is above the threshold, the output is 1; otherwise, it is 0. These are called "single-layered perceptrons"

Perceptrons can only classify linearly-separable sets of instances. If a straight line or plane can be drawn to separate the input instances into their correct categories, input instances are linearly separable and the perceptron will find the solution. If the instances are not linearly separable, learning will never reach a point where all instances are classified appropriately. Multi-layered perceptrons (artificial neural networks) have been created to try to solve this problem [112]. [113, 114] provide an extensive survey of existing work in artificial neural networks (ANNs) and our discussion is limited to the basics of ANN and and their use in network-related applications.

A multi-layer ANN consists of a large number of units (neurons) joined together in a pattern of connections. Units in a net are usually segregated into three classes: input units, which receive information to be processed; output units, where the results of the processing are found; and units in between known as hidden units. Generally, correctly determining the size of the hidden layer is a problem, because an under-estimate of the number of neurons can lead to poor approximation and generalisation capabilities, while excessive nodes can result in over-fitting and eventually make the search for the global optimum more difficult [115].

ANN depends upon three fundamental aspects: the input and activation functions of the unit, the network architecture, and the weight of each input connection. Given that the first two aspects are fixed, the behaviour of the ANN is defined by the current values of the weights. The weights of the net to be trained are initially set to random values, and then instances of the training set are repeatedly exposed to the net. The values for the input of an instance are placed on the input units, and the output of the net is compared with the desired output for this instance. All the weights in the net are then adjusted slightly in the direction that would bring the output values of the net closer to the values for the desired output. There are several algorithms with which a network can be trained [116]. However, the most well-known and widely-used learning algorithm to estimate the values of the weights is the back propagation (BP) algorithm [117].

ANNs have been applied to anomaly detection in multi-class as well as one-class settings. A basic multi-class anomaly detection technique using ANN operates in two steps. First, an ANN is trained on the normal training data to construct the different normal classes. Second, each test instance is provided as an input to the ANN. If the network accepts the test input, it is normal, and if the network rejects a test input, it is an anomaly. In addition, ANNs can also be trained using pre-identified anomaly data with correct labelling. The trained ANN can then be used to identify the exact anomaly in a subsequent setting to provide the root cause diagnosis. ANN has been commonly used in intrusion-detection systems (IDSs) where anomalous network behaviour is detected [118, 119]. In [120], the author proposed an ATM network controller that uses ANN for learning the relations between the offered traffic and service quality. ANN can also be used to detect fingerprints hidden in network traffic, as demonstrated in [121], where ANN was used for remote OS identification.

A radial basis function (RBF) network is a special form of three-layer feedback ANN that uses radial basis functions at each of the hidden units as acti-

vation functions. Each of the output units implements a weighted sum of hidden unit outputs. RBF networks have also been widely applied in many engineering fields [122] and have often been used in network IDS-related applications [123, 124, 125].

Support Vector Machines

Support vector machines (SVMs) are one of the newest supervised machine learning techniques [126]. An excellent survey of SVMs can be found in [127], and a more recent book is by Christianini et al. [128]. Therefore, in the present study, apart from a brief description of SVMs, we will refer to some more recent studies and the landmarks published before these.

SVMs revolve around the notion of a "margin" either side of a hyperplane that separates two data classes. Maximising the margin, and thereby creating the largest possible distance between the separating hyperplane and the instances on either side of it, has been proven to reduce the upper bound on the expected generalisation error. In the case of linearly-separable data, once the optimum separating hyperplane is found, data points that lie on its margin are known as support vector points and the solution is represented as a linear combination of only these points. Other data points are ignored. Therefore, the model complexity of an SVM is unaffected by the number of features encountered in the training data (the number of support vectors selected by the SVM learning algorithm is usually small). For this reason, SVMs are well suited to dealing with learning tasks where the number of features is large with respect to the number of training instances.

2.7.2 Hard-Margin linear SVM

Let n m-dimensional training inputs $\mathbf{x}_i$ ($i = 1, ..., n$) belong to Class 1 or 2 and the associated labels be $y_i = +1$ for Class 1 and $y_i = -1$ for Class 2. If the training data

are linearly separable, then a pair (**w**,b) exists such that

$$\mathbf{w}^T\mathbf{x}_i + b \geq +1 \text{ for } y_i = +1, \tag{2.1a}$$

$$\mathbf{w}^T\mathbf{x}_i + b \geq -1 \text{ for } y_i = -1, \tag{2.1b}$$

with the decision function given as

$$D(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\mathbf{x}_i + b \tag{2.2}$$

for $i = 1, ..., n$ where, **w** is the m-dimensional weight vector, b is the bias term.

The hyperplane

$$D(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\mathbf{x}_i + b = c \text{ for } -1 < c < +1 \tag{2.3}$$

forms a separating hyperplane that separates $\mathbf{x}_i$ ($i = 1, ..., n$) into Class 1 and 2. When c =0, the separating hyperplane is in the middle of the two hyperplanes with c = 1 and -1. The distance between the separating hyperplane and the training datum nearest to the hyperplane is called the *margin*. As shown in Figure 2.6, there is an infinite number of decision functions that satisfy (2.1). The hyperplane with the maximum margin is called the *optimal separating hyperplane* (see Figure 2.6).

The optimal separating hyperplane is obtained by minimising

$$Q(\mathbf{w}) = \frac{1}{2}\mathbf{w}^{\mathrm{T}}\mathbf{w} \tag{2.4}$$

with respect to **w** and *b* subject to the constraints

$$y_i(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i + b) \geq 1 \text{ for } i = 1, ..., n \tag{2.5}$$

which is a *quadratic programming (QP) problem* [129]. For linearly separable data,

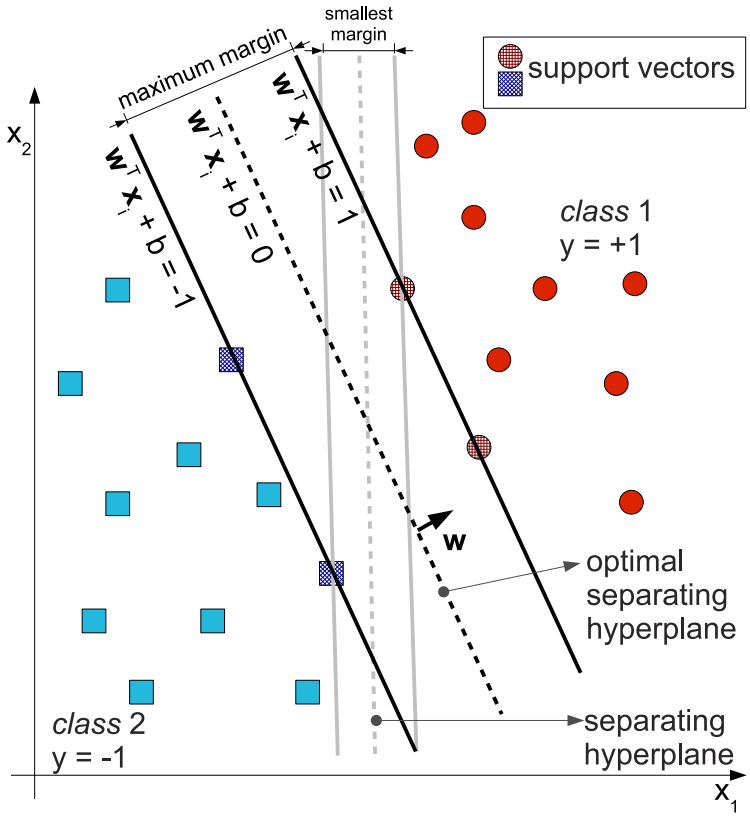


Figure 2.6: Hard-margin SVM

the data points that lie on its margins are known as support vectors.

Even though the maximum margin allows the SVM to select among multiple candidate hyperplanes, for many datasets, the SVM may not be able to find any separating hyperplane at all because the data contains misclassified instances. This problem can be addressed by using a soft margin that accepts some misclassification of the training instances [130].

Most real-world problems involve non-separable data for which no hyperplane exists that successfully separates the positive from negative instances in the training set. One solution to the inseparability problem is to map the data onto a higher-dimensional space and define a separating hyperplane there. This higher-dimensional space is called the "transformed feature space", as opposed to the input space occupied by the training instances (see Figure 2.7). With an appropriately chosen transformed feature space of sufficient dimensionality, any

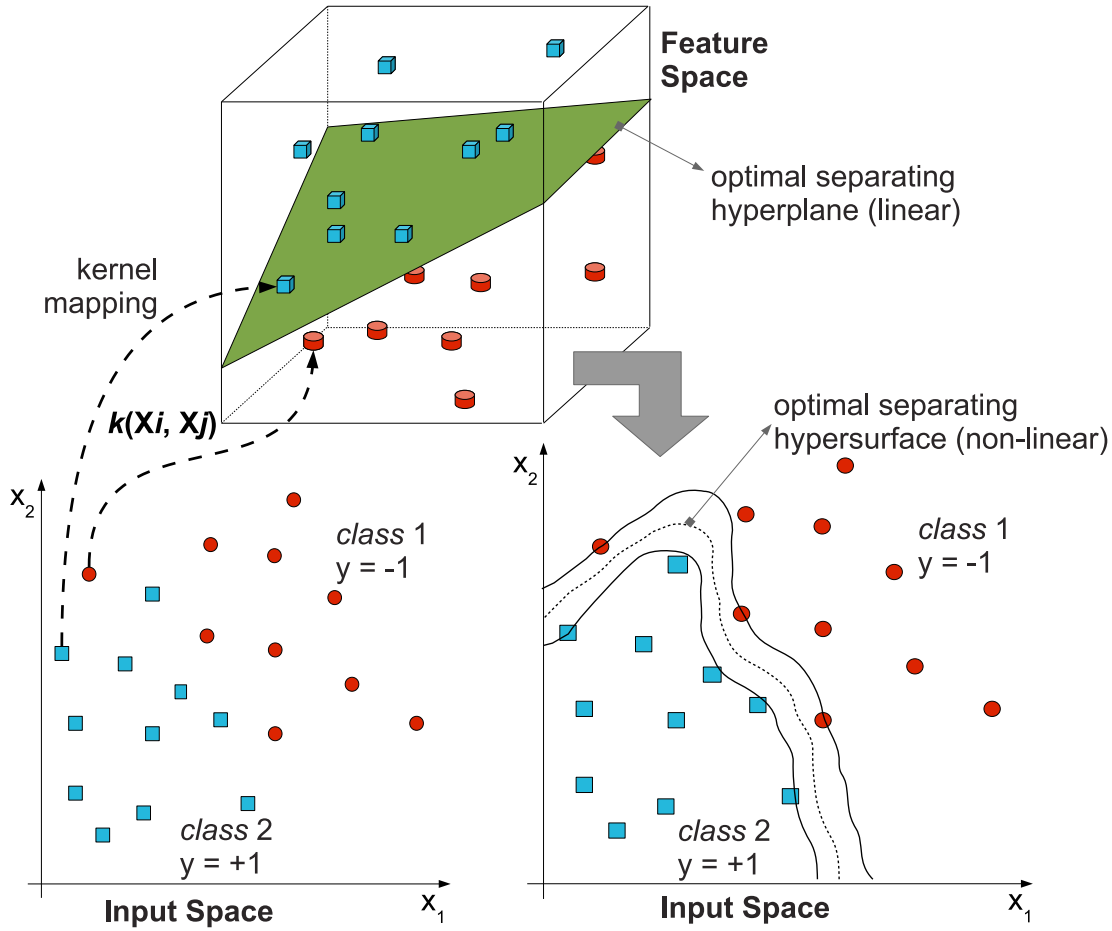


Figure 2.7: Kernel mapping from input space to feature space for creating a complex separating hyper surface between the classes

consistent training set can be made separable. A linear separation in transformed feature space corresponds to a non-linear separation in the original input space. The mapping from the input space to the transformed feature space is performed by "kernel functions", which are a special class of function that allow inner products to be calculated directly in feature space, without performing the mapping described above [131]. Once a hyperplane has been created, the kernel function is used to map new points into the feature space for classification. The selection of an appropriate kernel function is important, since the kernel function defines the transformed feature space in which the training set instances will be classified [132].

SVMs have been applied to anomaly detection in one-class settings with ker-

nels such as RBF kernels. This technique has been commonly used in IDS [133, 134, 135]. Network traffic classification is another application of SVMs. In Mantia et al. [136] and Este et al. [137], the authors performed TCP traffic classification using SVM, and Miaza et al. [138] used SVM-based learning for encrypted traffic classification. SVMs have also been used in automated computer system diagnosis using system traces [37], and for TCP throughput prediction [139] by support vector regression (SVR) [140].

Statistical classification - Bayesian Networks

In contrast to ANNs and SVMs, statistical approaches are characterised by having an explicit underlying probability model, which provides a probability that an instance belongs in each class, rather than simply a classification. Bayesian networks (BNs) are the most well-known representative of statistical learning algorithms [141]. A BN is a graphical model that encodes probabilistic relationships among variables of interest. The interesting feature of BNs, compared to decision trees, ANNs, or SVMs, is certainly the possibility of taking into account prior information about a given problem, in terms of the structural relationships among its features [142].

Naive Bayesian (NB) networks are very simple BNs which are composed of directed acyclic graphs with only one parent (representing the unobserved node) and several children (corresponding to the observed nodes), with a strong assumption of independence among child nodes in the context of their parent [143]. The assumption of independence among child nodes is clearly almost always wrong, and for this reason, naive Bayes classifiers are usually less accurate than other more sophisticated learning algorithms such as ANNs. The major advantage of the naive Bayes classifier is its short computational time for training [144].

Several researchers have adapted ideas from Bayesian statistics to create models for anomaly detection. For example, Peng et al. [145] and Valdes et al. [146] developed an anomaly detection system that employed BNs to perform intrusion

detection for cyber security using NB networks. BNs have also been used as a method for packet loss detection in TCP [147] and in [148] the authors introduced BARD, a method of network management using BNs. BN-based classifiers have also been used in network traffic classification, with Hong et al. [50] using BN for TCP flow classification and Agrawal et al. in [149] using BN to limit the bandwidth of spam flows. BNs can also be used as predictor algorithms (regression). Tariq et al. [150] introduced the What-If Scenario Evaluator (WISE), a BN-based tool that predicts the effects of possible configuration and deployment changes in content distribution networks. In [151], the authors demonstrate how NB classifiers can be used for automated diagnosis in mobile UMTS networks, and in [152], a comparison of BN diagnosis systems shows that a Bayesian classifier with continuous variables outperforms discrete variables in limited training set scenarios.

2.7.3 Unsupervised machine learning

In machine learning, the problem of unsupervised learning is that of trying to find hidden structure in unlabelled data. Since the examples given to the learner are unlabelled, there is no error or reward signal to evaluate a potential solution. This distinguishes unsupervised learning from supervised learning. Unsupervised classification, or clustering, has the advantage of grouping instances with similar properties and so simplifies the data [153]. Rather than forcing a set of classes onto the data set, an unsupervised classification scheme will reflect natural clusters in the data. This can be useful for finding novel or interesting features in the data that would not otherwise be found.

Statistical methods

These approaches have been developed from the view-point of statistical learning theory. They attempt to detect outliers in an on-line process through the unsupervised learning of a probabilistic model of the information source. A score is given

to an input based on the learned model, with a high score indicating a high possibility of being a statistical outlier. An off-line process of outlier detection uses batch detection, in which outliers can be detected only after seeing the entire data set. The on-line setting is more realistic than the off-line one when dealing with the vast amount of data in network monitoring. An example of such a method is "SmartSifter" (SS) [154].

Proximity-based approaches (Nearest neighbour)

The concept of nearest neighbour analysis has been used in several network anomaly-detection techniques. Such techniques are based on the key assumption that normal data instances occur in dense neighbourhoods, while anomalies occur far from their closest neighbours.

One of the most basic nearest neighbour techniques, k-Nearest Neighbour (kNN) is based on the principle that the instances within a dataset will generally exist in close proximity to other instances that have similar properties [155]. If the instances are tagged with a class label, then the value of the label of an unclassified instance can be determined by observing the class of its nearest neighbours. The kNN locates the k nearest instances to the query instance and determines its class by identifying the single most frequent class label. In general, instances can be considered as points within an n-dimensional instance space, where each of the n-dimensions corresponds to one of the n-features that are used to describe an instance. The absolute position of the instances within this space is not as significant as the relative distance between instances.

This relative distance is determined by using a distance metric. Ideally, the distance metric must minimise the distance between two similarly classified instances, while maximising the distance between instances of different classes. Distance (or similarity) between two data instances can be computed in different ways. For continuous attributes, Euclidean distance is a popular choice, but

other measures can be used [101]. For categorical attributes, a simple matching coefficient is often used, but more complex distance measures can be used [156]. For multivariate data instances, distance or similarity is usually computed for each attribute and then combined [101].

Clustering-based approaches

Cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in the same group (called a cluster) are more similar (in some sense, or another) to each other than to those in other group's (clusters) [157, 158, 159]. Cluster analysis itself is not one specific algorithm, but the general task to be solved. It can be achieved by various algorithms that differ significantly in their notion of what constitutes a cluster and how to efficiently find them. Popular notions of clusters include groups with small distances between the cluster members, dense areas of data space, intervals, or particular statistical distributions. Some of the main categories of clustering algorithms are as follows:

Connectivity-based clustering (Hierarchical clustering)

Connectivity-based clustering, also known as hierarchical clustering, is based on the core idea of objects being more related to nearby objects than to objects farther away [160]. Hierarchical clustering aims to obtain a hierarchy of clusters, called a dendrogram, that shows how the clusters are related to each other. These methods proceed either by iteratively merging small clusters into larger ones (agglomerative algorithms, by far the most common) or by splitting large clusters (divisive algorithms). When computing the distances, apart from the usual choice of distance functions, the user also needs to decide on the linkage criterion to use (since a cluster consists of multiple objects, there are multiple candidates to compute the distance to).

Centroid-based clustering

In centroid-based clustering, clusters are represented by a central vector, which may not necessarily be a member of the data set. When the number of clusters is fixed to k, k-means clustering gives a formal definition as an optimisation problem: find the k cluster centres and assign the objects to the nearest cluster centre, such that the squared distances from the cluster are minimised [161, 162]. The optimisation problem itself is known to be non-deterministic polynomial-time hard (NP-hard), and thus the common approach is to search only for approximate solutions. A particularly well-known approximative method is Lloyd's algorithm [163], often referred to as the "k-means algorithm". However, it only finds a local optimum, and is commonly run multiple times with different random initialisations.

Distribution-based clustering

The clustering model most closely related to statistics is based on distribution models. Clusters can then easily be defined as objects belonging most likely to the same distribution [164, 165]. While the theoretical foundation of these methods is excellent, they suffer from one key problem known as over-fitting, unless constraints are placed on the model complexity. A more complex model will usually always be able to explain the data better, which makes choosing the appropriate model complexity inherently difficult.

Density-based clustering

In density-based clustering, clusters are defined as areas of higher density than the remainder of the data set. Objects in these sparse areas - that are required to separate clusters - are usually considered to be noise and border points. The most popular density-based clustering method is "DBSCAN" [166]. In contrast to many newer methods; it features a well-defined cluster model called "density-reachability". Similar to linkage-based clustering, it is based on connecting points within certain distance thresholds. A

cluster consists of all density-connected objects (which can form a cluster of an arbitrary shape, in contrast to many other methods), plus all objects that are within these objects' range. Another interesting property of DBSCAN is that its complexity is fairly low.

Semi-supervised clustering

In addition to the similarity information used by unsupervised clustering, in many cases, a small amount of knowledge is available concerning either pairwise (must-link or cannot-link) constraints between data items or class labels for some items. Instead of simply using this knowledge for the external validation of the results of clustering, one can imagine letting it guide or adjust the clustering process, i.e. provide a limited form of supervision. The resulting approach is called semi-supervised clustering [167, 168].

2.8 Transmission Control Protocol (TCP)

As discussed in previous sections, network protocol-related attributes have been widely used to characterise network behaviour and to detect anomalies. TCP is by far the most common transport protocol in the Internet and has been regularly used in anomaly detection applications for a number of years. This section aims to provide an overview of the TCP protocol and how it offers a window into network behaviour.

Traffic measurements in several large campus networks indicated that over 90% of network traffic is transported by the TCP [169]. The original definition of TCP appeared in RFC-793 in 1981 [27]. Since then, many researchers have identified problems and weaknesses of the protocol, and proposed solutions. RFC-4614, "A Roadmap for Transmission Control Protocol (TCP) Specification Documents" provides a comprehensive review of all specification changes that have transformed the TCP to its current status [170]. Although the implementation de-

tails can differ, the core functionality of the protocol remains unchanged since its introduction.

TCP provides a reliable connection-oriented data service in packet-switched networks. This is achieved by employing acknowledgements (ACKs), sequence numbers, and timers. TCP employs window-based congestion control mechanisms to adjust its congestion window size (*cwnd*). *cwnd* is the maximum amount of data that the sender can transmit before receiving an ACK. The receiver also advertises a limit (*rwnd*) on the amount of outstanding data. Data transmission is always governed by the window size $W_m = \min(cwnd, rwnd)$.

TCP Reno [171] is one of the most widely-adopted TCP schemes. It has four transmission phases: slow start, congestion avoidance, fast recovery, and fast retransmit. TCP maintains two variables; *cwnd*, which is initially set to be 1 maximum segment size (MSS), and the slow start threshold (*ssthresh*). At the beginning of a TCP connection, the sender enters the slow start phase, in which *cwnd* is increased by 1 MSS for every ACK received; thus, the TCP sender's *cwnd* grows exponentially in round-trip times (RTTs). When *cwnd* reaches *ssthresh*, the TCP sender enters the congestion avoidance phase. Reno employs a sliding-window-based flow control mechanism, allowing the sender to advance the transmission window linearly by one segment upon reception of an ACK, which indicates the last in-order packet received successfully by the receiver. When packet loss occurs at a congested link due to buffer overflow at the intermediate router, either the sender receives duplicate ACKs (DUPACKs), or the sender's retransmission time-out (RTO) timer expires. These events activate TCP's fast retransmit and recovery, by which the sender reduces the size of its *cwnd* to half and linearly increases *cwnd* as in congestion avoidance, resulting in a lower transmission rate to relieve the link congestion [172].

2.8.1 TCP variants

The initial idea and definition of TCP was introduced in RFC-793 [27] in 1988. Since then, many researchers have closely studied every aspect of TCP and proposed solutions for numerous problems [173, 174]. In recent years, it has become clear that it can perform very poorly in networks with high bandwidth-delay product (BDP) path gigabit networks [175]. The problem stems from the fact that the standard TCP AIMD congestion control algorithm increases the congestion window too slowly. TCP was primarily designed for wired networks where the random bit error rate (BER) is negligible, and congestion is the main cause of packet loss. Consequently, traditional TCP schemes suffer from severe performance degradation in a mixed wired and wireless environment due to inherently high BER [176].

An update and supplement to TCP was issued in RFC-1122 [177]. This introduced many new concepts to make TCP much more robust and efficient. Since its introduction, TCP has been modified numerous times to suit specific applications and to improve the performance with growing networking needs. Jacobson [173] introduced congestion control and flow control to TCP. This has resulted in several important TCP versions with different congestion control variations that are used by popular operating systems and networking platforms. Among the most popular are TCP Tahoe [27], TCP Reno [27], TCP Vegas [178], TCP New Reno [179], and TCP SACK [180].

TCP implementations in the real world have to follow both RFC-793 [27] as well as RFC-1122 [177] to guarantee their interoperability. Although RFCs 793 and 1122 give a detailed description of TCP implementation, two TCP implementations that conform to the specifications can differ slightly because an implementer has some freedom to choose the software design and the parameters to interpret the protocol standards. With the introduction of other TCP variants, TCP implementations have become subjective to each individual platform, thus

giving hundreds of unique flavours.

The following list categories some of the most commonly-used TCP variants.

Reactive Congestion Control - The standard Reno scheme employs reactive flow control. The congestion window is adjusted based on the collective feedback of ACKs and DUPACKs generated at the receiver.

- TCP New Reno [179] - Modifies the fast recovery mechanism of Reno to cope with multiple losses from a single window. TCP New Reno is used by default in Redhat 7.2, FreeBSD 4.3-4.5 [181]
- TCP BIC [182] - Uses a binary search algorithm where the window grows to the mid-point between the last window size (i.e., maximum) where TCP has a packet loss and the last window size (i.e., minimum) where it does not have a loss for one RTT period. TCP BIC is used by default in Linux kernels 2.6.8 through 2.6.18 [183].
- TCP CUBIC [184] - Simplifies the window adjustment algorithm of BIC-TCP by replacing the concave and convex window growth portions of BIC-TCP by a cubic function (which contains both concave and convex portions). TCP CUBIC is used by default in Linux kernels from version 2.6.19 to 3.1 [183].
- High-speed TCP (HSTCP) [185] - Uses a generalised AIMD where the linear increase factor and multiplicative decrease factor are adjusted by a convex function of the current congestion window size. HSTCP is included in Linux as a selectable option in the modular TCP congestion control framework since Linux kernel 2.6.16 [183]

Proactive Congestion Control - In proactive congestion control, the sender attempts to adjust the congestion window pro-actively to an optimal rate, according to the information collected via feedback, which can be translated to an indication of the network condition. Delay-based congestion control is a

common proactive congestion control mechanism which uses packet delay to identify congestion as opposed to packet losses being used by loss-based congestion control.

- TCP-Vegas [178] - Estimates the backlogged packets in the buffer of the bottleneck link. Vegas selects the minimal RTT as a reference to derive the optimal throughput the network can accommodate. TCP Vegas is available in both Unix kernel and Free-BSD as an optional module.
- TCP-Westwood [186] - A rate-based end-to-end approach in which the sender estimates the available network bandwidth dynamically by measuring and averaging the rate of returning ACKs. TCP Westwood is included in Linux as a selectable option in the modular TCP congestion control framework since Linux kernel 2.6.16 [183]
- TCP-Hybla [187] - Increases the congestion window size more aggressively to compensate throughput drop due to RTT increase.
- Delay-gradient congestion control (CDG) [188, 189] - Created by Hayes et al., CDG is a sender-side delay-gradient TCP congestion control algorithm. CDG has been designed to overcome the key limitation of other congestion control mechanisms - their need to establish accurate path RTT measures to set a delay-threshold. CDG has an improved tolerance to non-congestion related packet losses and improves the coexistence and fairness with loss-based TCP.

In addition to TCP variants mentioned above, there are many other TCP variants created for specific applications or operating systems. For example, Windows operating system, since Windows Vista uses Compound TCP (CTCP) designed to aggressively adjust the sender's congestion window to optimise TCP for connections with large bandwidth-delay products while trying not to harm fairness [190]. TCP Westwood+ implemented in Linux kernel [191] are further

examples of improved TCP variants.

2.9 TCP trace analysis-based behaviour inference

TCP trace analysis dates back several decades to the introduction of TCP as a mainstream transport protocol. TCP sits in the middle of the TCP/IP protocol stack and masks lower layers' behaviour to minimise their effects on the application layer. This unique position of the TCP in the protocol stack offers a window into the behaviour of both lower and higher layers, which are otherwise impossible to observe from any other single vantage point [12]. The effects embedded into TCP packet streams, or "artefacts", can be used as an excellent source of information to remotely diagnose a performance bottleneck.

TCP trace analysis tools, techniques and applications can be categorised into two groups: visualisation methods and inference methods. Table 2.2 summarises the work on trace analysis tools available in literature.

Trace visualisation tools are often used by network professionals to analyse the packet stream and associated details to identify the embedded clues about client and link behaviour [192, 193, 194]. Although very valuable in manual diagnosis processes, these tools are only capable of assisting the fault diagnostics by organising and summarising the trace data into an easily comprehensible format, rather than directly determining the root causes. Whilst we extensively used tcptrace, xplot and wireshark during our research, these tools alone could not perform automated end-to-end diagnosis.

In comparison with trace visualisation, inference methods identify and deduce connection information from packet traces. Inference of packet traces is used in several network applications, as shown in Table 2.2. These tools use techniques such as simple packet sequence analysis [197, 200], heuristic analysis [206], and machine-learning techniques [203, 205, 75] to infer the connection behaviour.

A number of inference tools have been developed to determine connection characteristics (network tomography) such as RTT [195, 196], congestion window [196], bandwidth estimation [197, 198, 199], and packet loss [200, 147, 207]. However, these studies focus only on different parts of a complete network diagnostic problem and are primarily used to collect the information needed to guide the diagnostic decisions. Consequently, application of any one technique is prohibitive in a comprehensive diagnostic tool focused on capturing wide range of network attributes and behaviours.

The following two sections discuss how TCP inference tools and techniques relates to network diagnostics and drawbacks of exiting methods towards scalable, automated UD diagnosis system has been highlighted in Table 2.3

TCP Trace Analysis	
Visualisation methods	Inference methods
• Trace statistics [193] • Trace graphs [192] • Trace packet headers and sequences [194]	• Network tomography – Determining round trip time (RTT) [195, 196] – Bandwidth estimation [197, 198, 199] – Packet loss estimation [200, 147] – TCP windows and congestion estimation [196] • Protocol diagnosis [201, 202, 21] • Intrusion detection and security [203, 204] • System fingerprinting [75, 74] • Internet traffic identification and classification [205, 62]

Table 2.2: TCP trace analysis tools, techniques and applications.

2.9.1 Network diagnosis using TCP trace analysis

Diagnosis of network performance problems requires a methodical approach. First, the faulty segment of the network has to be isolated and second, the exact root cause of the problem should be identified. Analysis of packet traces, especially from the TCP, is a sophisticated inference-based technique used to diagnose complicated network problems in specialised cases [208].

Although the existing research literature lacks a comprehensive, automated tool to infer network faults from TCP traces, there have been efforts to explore the possibility of such a system. Table 2.3 provides a comparison of available TCP inference-based network diagnostic tools and the drawbacks of these tools.

The tool tcpanaly by Paxton [201] was intended to detect subtle variations of different TCP types and identify otherwise hidden TCP implementation issues. Tcpanaly was an early attempt to automate the inference of packet traces for fault diagnosis, although it was only capable of detecting errors in the TCP itself. The TCP Behaviour Inference Tool (TBIT) [209] was another tool designed to characterise TCP behaviour to detect non-compliance in TCP implementations in public web servers. Work by Jaiswal and co-researchers [196, 212] on inferring connection characteristics through passive analysis of packet traces also attempted to automate the diagnosis using a heuristic process, which was later extended to include a more extensive set of rules by Mellia and co-researchers [206, 213, 202].

In recent years, revived attention has been given in the research community to the creation of automated diagnostic systems using TCP packet traces. The project web100 [12] focuses on collecting per-connection TCP statistics through kernel instrumentation (KIS) and has received much attention in the research community. The Web100 tool-set has been used extensively for diagnosing high-speed connectivity issues in projects such as the CERN-Large Hadron Collider and the Visible Human project [208]. The capability of web100 to capture major protocol events, using parameters otherwise hidden from users, has been in-

Tool	Characteristics	Drawbacks
tcpanaly [201]	• Designed to detect TCP protocol errors • Earliest attempt to automate the TCP trace analysis process • Analyses packet sequences and uses fixed code segments for detecting abnormalities.	• Only designed to detect TCP protocol errors • Fails to create a general technique to detect errors. Instead, the authors have coded the algorithms for "idiosyncrasies of the different TCP implementations [201]", an approach that limits scalability.
TBIT [209]	• Developed to detect bugs and non-compliance in TCP implementations deployed in public web servers. • Creates artificial packet sequences to test the server TCP behaviour.	• Artificial packet sequences are often blocked by firewalls. • Diagnosing capability is limited to a pre-programmed set of tests and TCP implementations.
Mellia et al. [206]	• Uses cascading heuristic technique to match different predefined conditions using trace information. • Considers RFC-2581 [172] as the base TCP standard.	• Only capable of detecting a few phenomena and lacks the ability to provide root cause diagnosis for the trace behaviour. • Heuristics become extremely complicated when detecting complex faults. • A general heuristics cannot be developed since behaviour of many TCP variants differs from RFC-2581.
NPAD / Pathdiag [210]	• Uses web100 [12] as the basic information source for monitoring TCP behaviour. • Relies on active measurement and passive analysis of path data. • Uses sender TCP parameters to detect client and path issues. • Performs effectively when hops between client and server are limited.	• Deployment is complex and time-consuming because of server kernel recompilation or dedicated diagnostic servers needed by web100. • Relies on the TCP flag negotiations to determine the status of connection options. However, negotiated functions are often not properly performed by the TCP implementation [51, 211] and diagnosis can be misled by header flags. • The connection details are captured only at the server. No mechanism to directly gather client's TCP parameter data. • Since most diagnostic parameters are calculated using available algorithms, the accuracy of each of these algorithms against different types of TCP in different platforms cannot be guaranteed. • New algorithms must be added to the analysis engine in order to detect any additional faults.
NDT [21]	• Uses web100 to gather per-connection TCP statistics. • Relies on active measurement and passive analysis of path data. • Uses bi-directional data streams and server TCP KIS captures.	• Recompilation of server kernels hinders the flexible deployment of the tool in any node. • For detection of every fault AND logic of multiple conditions is checked. However, manifestation of each condition in the packet stream can vary with the implementation of TCP. • Only offers limited diagnostic capability

Table 2.3: Diagnostic tools utilising TCP inference: comparison of characteristics and drawbacks

strumental in some of the latest automated diagnostic tools. Web10Gig, a direct follow-on to the work introduced in Web100 is currently working to provide an instrumented implementation of TCP as part of standard Linux to diagnose hidden network issues in gigabit networks [214].

NPAD diagnostic servers with Pathdiag [210] and Network Diagnostic Tool (NDT) [21] use the information extracted using web100 instrumentation to diagnose connectivity problems of the client systems. Similar to Web100, statistical information for TCP research (SIFTR) is a kernel module available in FreeBSD since kernel 6.3 that logs a range of statistics on active TCP connections to a log file [215]. It provides the ability to make highly granular measurements of TCP connection state, aimed at system administrators, developers and researchers. Using SIFTR measurements, diagnostic systems can be developed to detect specific characteristics of network faults.

Table 2.3 shows the drawbacks in the existing solutions to infer UD soft-failures using TCP. Some of the most common issues are (i) limited scalability due to the algorithm design targeting a specific fault, (ii) being based on heuristics or expert rules, (iii) requiring privileged access to the end user device to capture data, (iv) requiring a kernel or core system modification at the user to set up the tool. These limitations have prevented such solutions from being used in commercial networks. As evidenced by recent patent filings by networking companies [216, 217, 218], there is a renewed interest in creating automated, scalable TCP inference mechanisms for fault diagnosis. However, the solutions created must have the flexibility to scale easily to diagnose faults in a wide range of user devices and network conditions. The research proposed in this thesis focuses on (i) using simple data collection mechanisms which do not require client side modifications, (ii) comprehensive signatures of faults instead of expert rules and (iii) modular, scalable machine learning mechanisms for root cause diagnosis.

2.9.2 Machine learning for TCP-based behaviour inference

When trained with packet traces representing a specific behaviour, supervised ML algorithms can identify a similar behaviour in test traces.

ML-based inference of TCP traces has been used in several applications. In recent studies, Dondo et al. [203] used ANNs, Shon et al. [219] used SVMs, and Kuang et al. [220] used kNN algorithms to infer network intrusion events using TCP packet traces. With the emergence of a diverse range of internet applications, Internet traffic classification has gained substantial momentum. Hong et al. [50] used Bayesian classifiers for inferring traffic categories from packet traces, and similarly, SVMs have been used by Yuan et al. [205]. Machine learning algorithms have also been used for network tomography applications such as TCP throughput prediction using SVMs [198] and packet loss estimation [147] using Bayesian networks. TCP inference using Bayesian classifiers has been used for remote system fingerprinting by Beverly [75], and in a similar study, Burroni et al. [121] introduced a remote OS identification tool using ANN.

A number of studies have used machine learning for root cause diagnosis of enterprise networks [29, 30, 31, 32, 33], access links [34, 35], home networks [36], and computer systems [37, 31]. These diagnostic tools lack the functionality and generalisation required for a broader diagnostic solution. For example, the decision-tree-based "NEVERMIND" [34] is a tool developed purely for the diagnosis of ADSL link problems, while the decision-tree-based "Netprints" [36] is only used for diagnosing Wi-Fi home network issues. Furthermore, these methods require information such as user requests, event logs, system calls or private network traffic, which demand privileged access. These limitations can be avoided by using an inference-based method with an end-to-end TCP connection, independent of the link layer. Our literature survey did not find any comprehensive, scalable, intelligent inference techniques using TCP packet traces for an automatic diagnosis of network performance problems.

CHAPTER THREE

SIGNATURES OF NETWORK SOFT FAILURE IN END-USER DEVICES

3.1 Introduction

Studies have shown that ML-based detection techniques [36] can provide the automation desired in soft failure diagnostic systems. ML-based techniques offer better scalability [221, 222] compared to systems based on expert rules such as Pathdiag [210] and NDT [21]. ML algorithms capture the runtime behaviours of networked devices and characterise them to create unique "signatures" of network or UD failures. A signature is defined as a collection of features (or attributes), each representing a single aspect of runtime behaviour, and provides the key to differentiate normal behaviours from abnormal or faulty behaviours.

A detailed discussion of common uses of network signatures in on-line traffic classification, IDSs, and failure detection systems was presented in Section 2.4. These signatures often contain only a limited set of features, because they were generated for specific applications. In addition, recent studies overlook the effects of network or link variations on signatures by capturing training and evaluation data from the same network location. Insufficient numbers of features, dependence on stable link properties, source protocol variants [24], and large sample sizes make existing signatures published in the literature unsuitable for UD soft

failure diagnosis. Attempts to use such signatures for soft failure detection at the device level have resulted in unacceptably high false detection rates [82].

Therefore, in this chapter, we propose a new way of characterising network soft failures at the UD, with the broad objective of creating an automated diagnostic system. The main contributions of the chapter are summarised as follows:

(i) The novel concept of aggregated TCP statistics-based normalised signatures is proposed as a scalable and comprehensive way of uniquely characterising a UD soft failure. These signatures provide the foundation for ML-based diagnostic systems that can save time and costs.

(ii) Link adaptive signature estimation (LASE) is proposed as a technique to dynamically generate NSSs for arbitrary link conditions from a limited set of collected NSSs to improve the scalability.

(iii) We collect data from various networks and apply the signature concept to create signatures of soft failures. We analyse how different network properties affect such signatures and identify the challenges, dependencies, and limitations.

(iv) We also conduct extensive analysis of the LASE technique to offer an insight into various types of features, to show that these signatures are scalable and can be estimated with a high degree of accuracy.

3.2 Normalised Statistical Signature (NSS)

As remote access to an UD is usually not possible for a network service provider, a TCP packet stream provides an ideal observation point to identify the root causes of network performance-related issues. Because of TCP's position in the middle of the protocol stack and its reliable transport functionality, performance issues in other layers are embedded as anomalies in TCP packet streams. Furthermore,

TCP has the added advantage of being supported by most networks and devices today. Because of these unique benefits offered by TCP, we rely on the features extracted from captured TCP packet streams to characterise UD anomalies.

Our focus in this thesis is fault diagnosis performed from the edge of the network administrator domain, using the edge router/access server as the observation point. This narrows down the path to an access link and eliminates complexities that can affect the uniformity of captured features. The diagnostic server is deployed as an application package with simple installation, and modules containing trained classifiers can be attached to the main system.

The capability of a diagnostic system is primarily dependent on the effectiveness of extracted signatures to characterise faults. Key aspects in constructing a signature include the information source, observation point, capture mechanisms, data extraction, and post-processing. Furthermore, the features used to construct a signature should be applicable to a wide variety of network conditions and allow the diagnostic system to be used to capture new abnormalities.

3.2.1 Operational overview

Figure 3.1 shows an operational overview of the signature generation process. The user can initiate the diagnostic process, which runs as a browser-based application. This application loads the "user modules" and starts the data connections for active device monitoring. The "capture modules" then acquire packet traces of controlled TCP connections between the diagnostic server and the UD. However, it is not necessary for the diagnostic server and the capture module to be co-located. The capture module can be deployed into the access router of the last-mile link or to the edge of the network, while the diagnostic server resides in a dedicated location. With this method, the user's privacy is protected, as they do not have to relinquish privileged administrative-level access to network operator personnel, as in most conventional remote diagnostic services. Mean-

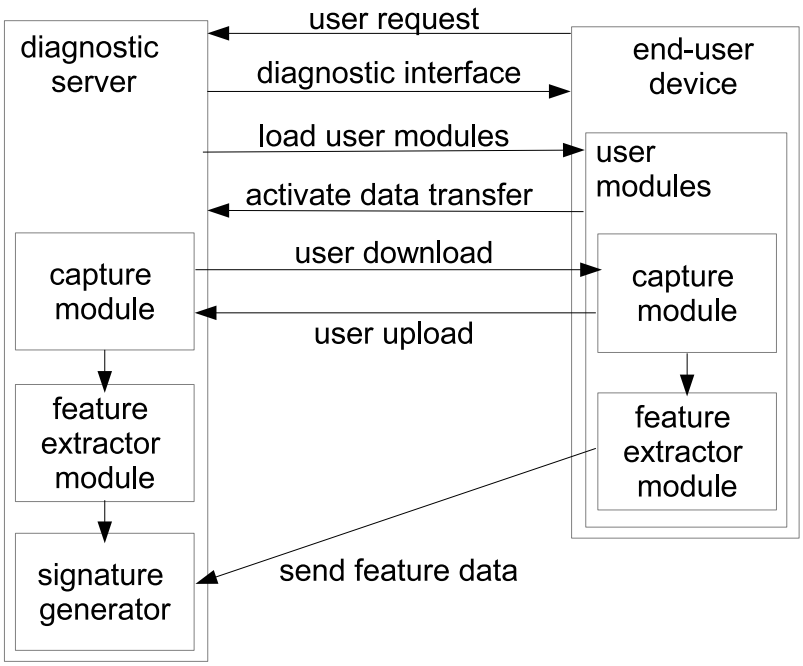


Figure 3.1: Operational overview of the UD soft failure signature generation process. The capture module can be deployed into the access router of the last-mile link or to the edge of the network.

while, the operator can remotely collect the necessary packet data for diagnosis through this automated process. Collected packet traces are then analysed at the server and the UD to extract aggregated statistical attributes of the trace. Only the extracted feature data is transmitted from the UD to the diagnostic server as a key-value pair and only requires under 10 Kilobytes of newly create overhead data to be transmitted between user and the ISP server. If required, raw traces from the UD can also be transferred to the diagnostics server for more advanced investigations.

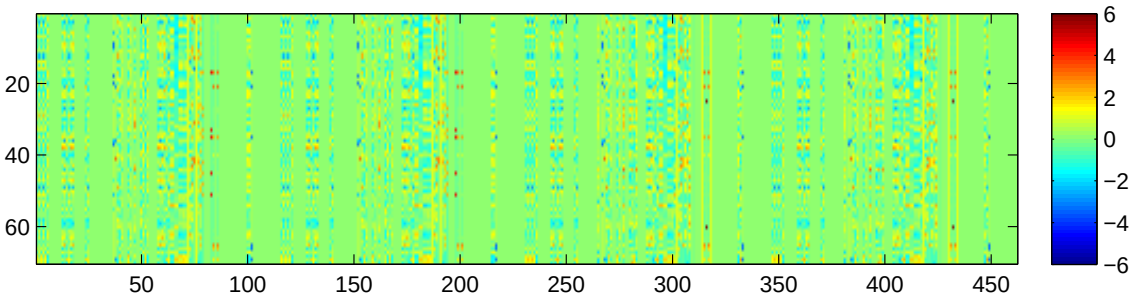
When a user initiates the diagnostic process, two packet traces are captured, one from an upload and another from a download. We extract 230 distinct attributes called "raw features" from each trace and then serially combine them (amalgamate) to create a feature vector of 460 features. This feature vector with 460 features is called a "statistical signature". The statistical signature is then normalised against a previously collected healthy baseline to create a feature vector

called a “normalised statistical signature“ (NSS).

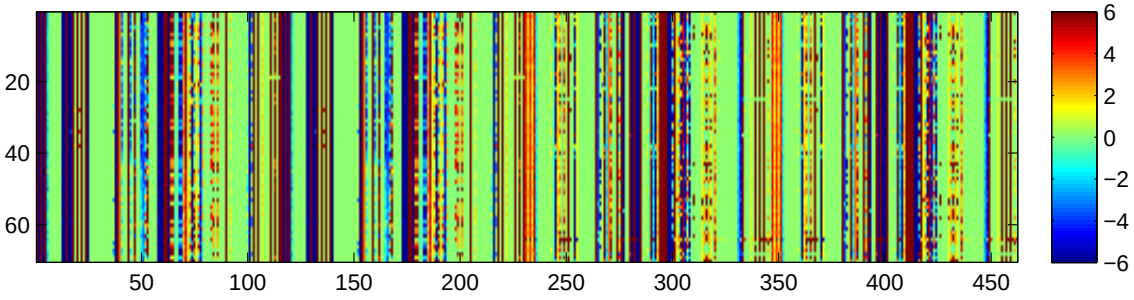
Figure 3.2 shows NSSs for nine different UD soft failures. In each sub-figure, we have grouped 70 NSSs that belongs to the same fault to show that every fault has a unique NSS ”pattern” i.e. while the NSSs may differ slightly, for the same fault, they have a pattern that is similar. We have visualised 70 NSSs per group as opposed to only a few to show the unique patterns created for each fault and consistency of the pattern within the group. However, the selection of 70 NSS per group is arbitrary. The sub-figures clearly show, for a particular fault, the proposed process generates a consistent signature with a unique set of characteristics, and each group of NSSs are uniquely identifiable. However, figures also show that within each group, the NSSs show minor variations due to the stochastic nature of real-world networks.

To diagnose the root causes of UD soft failures, it is necessary to characterise and build a signature of normal network behaviour. This signature serves as the baseline of a “healthy“ UD. In Chapter 1, we defined a healthy UD as a device capable of delivering typical network performance in the prevailing local network circumstances to an end-user. Healthy signatures are captured by placing controlled, healthy UDs with optimal configuration settings as active probes in the access network as part of the system initialisation process. Depending on the dynamic nature of the network, these healthy UDs actively probe NSS generation process and the captured signatures are used to update the healthy baseline. Since, trace characterisation technique proposed here is agnostic to the specific TCP types, limited number of healthy UDs can create the baseline signature without impacting the NSS accuracy for UDs with a variety of TCP stacks.

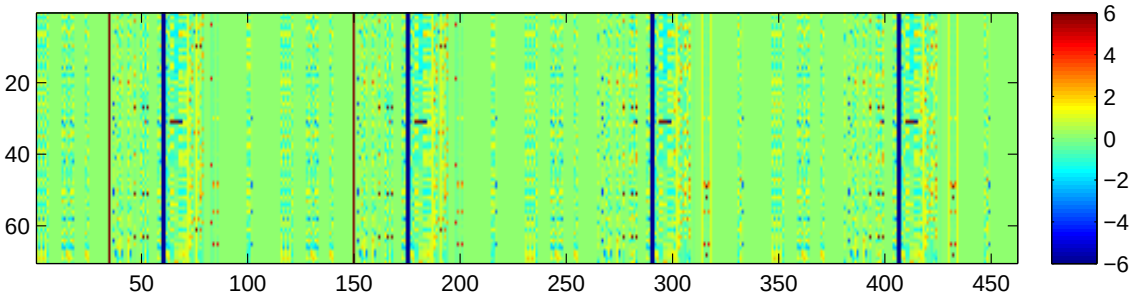
When a device suffers from a failure, the “faulty“ signature is analysed against the baseline healthy signature to identify the unique pattern of the deviations from the expected behaviour. These deviations can be used to pinpoint the likely cause. Figure 3.2 also highlights the challenge in diagnosing the root causes using



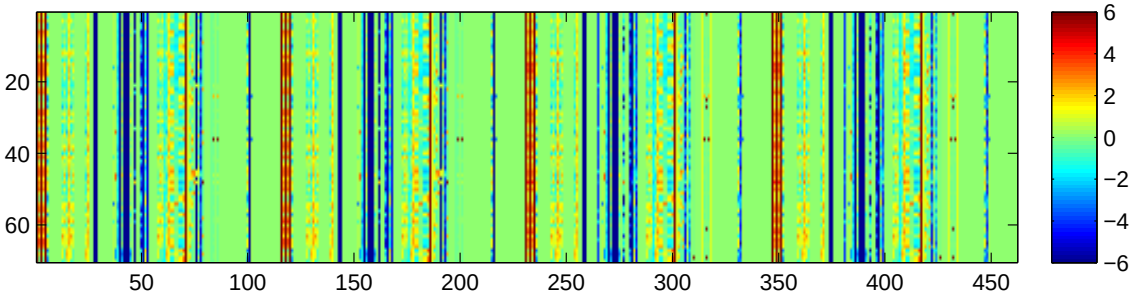
(a) Healthy UD



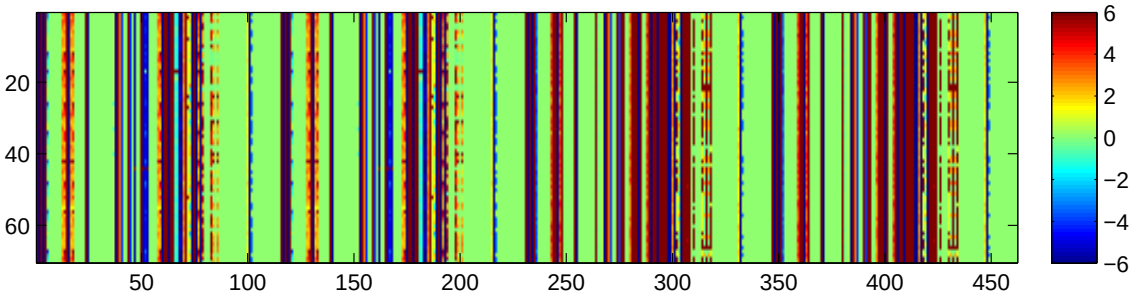
(b) Disabled SACK error



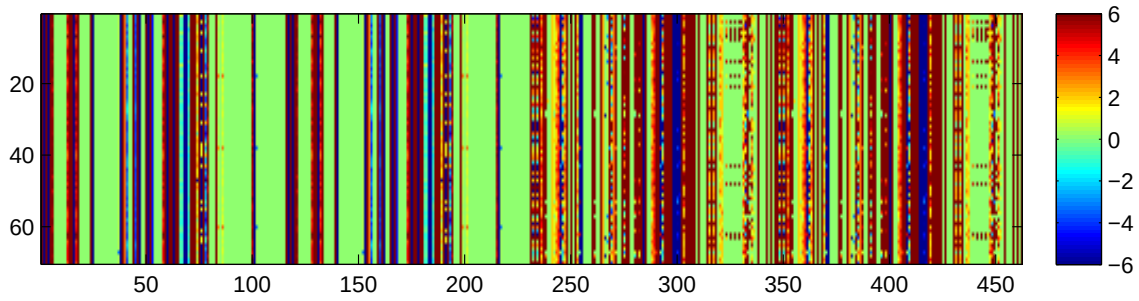
(c) TCP timestamps error



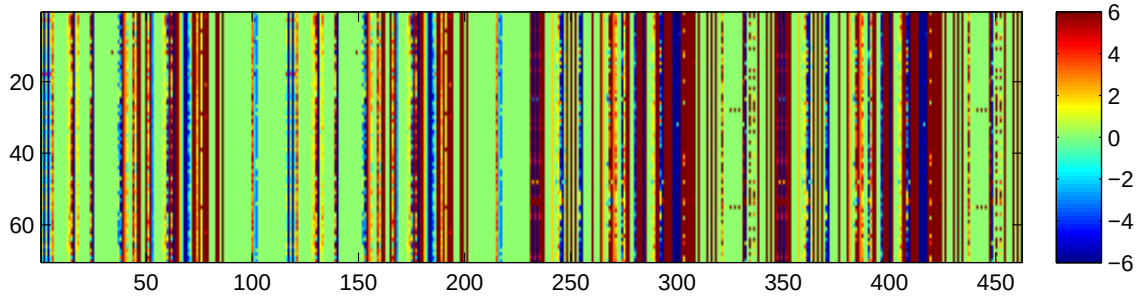
(d) Window scaling error



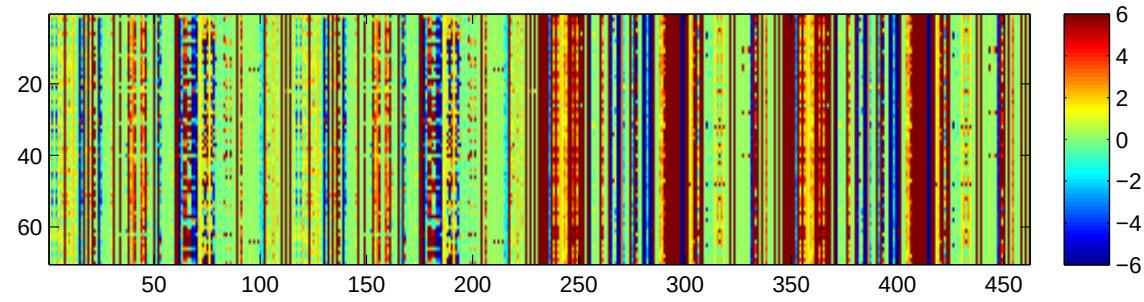
(e) Link-UD speed mismatch



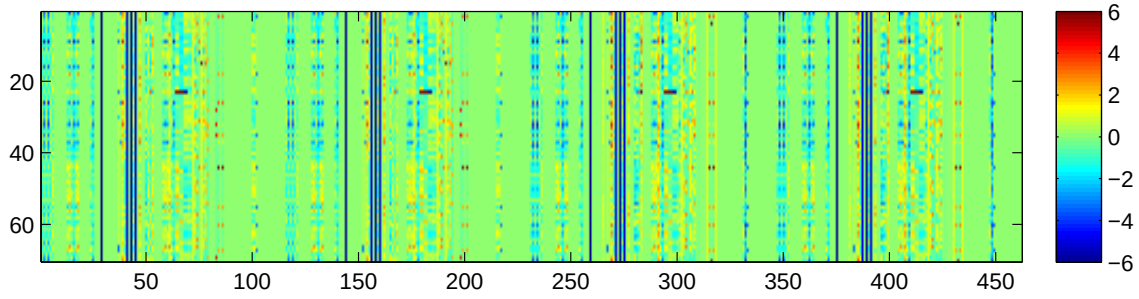
(f) Duplex mismatch



(g) UD firewall causing delay



(h) UD firewall causing packet loss



(i) Simultaneous insufficient read-write buffer

Figure 3.2: Comparison of NSSs for common UD soft failures. Here, the x-axis (columns) of each figure represent 460 features of the 70 NSSs shown on the y-axis (rows). Each of the features has been normalised and scaled $[-6, 6]$ and each of the features is represented by a coloured vertical line projecting the scaled feature value to RGB space. Although, the NSSs of the same fault show minor variations due to stochastic nature of the networks, groupings of 70 NSSs of the same fault clearly shows that the NSSs are consistent in uniquely characterising a fault and can be used as the "fingerprint" for a diagnosis.

signatures collected over real-world networks. Since every signature is slightly different, the diagnostic technique has to be able to create a "model" of the fault pattern that's generalised for the same fault. However, at the same time, it should also preserve the differences (create different models) for different faults. It is a challenge to create such complex models and often, machine learning is uniquely suited to provide the best solutions.

3.2.2 Capture module

Most signatures in the literature are generated using the passive collection of arbitrary traffic flows captured from live networks. Due to the on-demand nature of our system, we opted to create signatures using a user-initiated connection with artificially-generated data between the UD and the diagnostic server.

As previously mentioned, TCP has many advantages as an information source. It is supported by almost all networking devices, and its reliable transport function means that network failures impact meaningfully on its behaviour. The protocol has lightweight and portable packet-capturing mechanisms [223], and the popularity of the protocol has resulted in a large number of techniques being developed over the years for information extraction [193, 224] from TCP packet traces.

When a user initiates the test, two TCP-based data transfers of a fixed 20 MB file (an upload and a download) serially run between the UD and server. The time taken to gather data depends on the file size, link quality, and the UD fault. A larger size file transfer generates more packets and consequently, a more statistically robust signature. However, a larger file increases the data collection time, especially when the UD or link is highly degraded. For the purpose of this research, we used a file size of 20 MB as that size enabled us to collect sufficient samples for the research without compromising the data accuracy. However, this file size can be increased as long as training and testing samples are collected us-

ing a same size file transfer. Bi-directional packet streams of these connections are captured using the packet capture utilities deployed at both the diagnostic server and UD. We use a self-contained, portable packet capture mechanism that does not require kernel manipulations or installations.

Constant transfer size, protocol, capturing parameters, and static server configuration provide a baseline performance characteristic, unavailable when using passive capture of arbitrary live traffic flows. By controlling the content of captured traffic and limiting the amount of traffic captured, we avoid many privacy concerns [225] that arise when using live traffic.

3.2.3 Feature extraction module

The two captured TCP packet traces are then sent to the feature extraction modules where they are analysed and statistical attributes are extracted. We use an analysis tool based on tcptrace [193] to identify, analyse, and aggregate the trace information, the result of which we call "raw" features.

Packet captures are collected from both the upload and the download at the server and client. We extract 115 features from server side capture and the other 115 features from the client side capture for each transfer. The 230 features from the download is then concatenated with the same 230 features from the upload, each an aggregated statistical attribute of the trace. In summary, process extracts 460 unique features. Table 3.1 shows broadly (but not exhaustively) the categorised types of features with a limited set of examples. An exhaustive list of features can be found in Appendix 1.

3.2.4 Signature generator

The raw features extracted at the user module are sent to the signature generator at the diagnostic server. The signature generator combines the feature set col-

Feature type	Features
Cumulative totals of various packet types	Total packets SACK packets DSACK packets Retransmitted packets Triple dupACKs ⋮
Cumulative payload characteristics	Unique bytes Packets with data Data retransmitted Data out-of-order Data missing ⋮
Max, min, mean, cumulative observation frequencies of events	Out-of-order events Cumulative acknowledged segments Data retransmissions Max segment retransmissions Zero window advertisements Window probes ⋮
Max, min, mean information on variable parameters	Window advertisements Segment size ⋮
Initial state and final state parameters	Window scaling advertisements Initial window ⋮
Round trip time, arrival time progression analysis	Retransmission times-max-min-ave Idle times-max-min-ave RTT-max-min-std ⋮
Boolean parameters of TCP settings	3-way handshake flags (SYN/FIN) TCP options Data pushed ⋮

Table 3.1: Types and limited examples of features extracted from the TCP trace. An exhaustive list of features can be found in Appendix 1. The same unique set of 115 features are extracted from both upload and download packet streams at the server side and client side. Separate cumulative features are created for each direction ($client \leftrightarrow server$) considering the bi-directional nature of the connections.

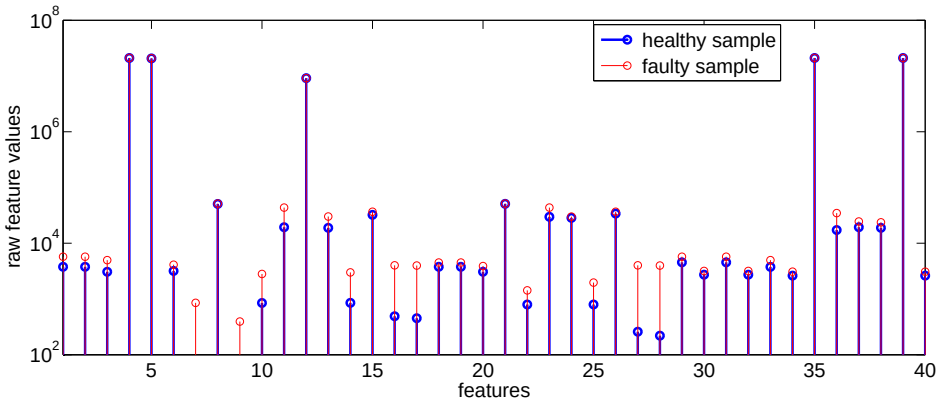


Figure 3.3: Comparison of the first 40 raw features extracted from healthy and faulty UDs.

lected at the diagnostic server with those collected at the UD to create the final raw feature vector with 460 features which is called the raw signature.

For training and testing purposes, we store all our feature vectors in a database $\Theta_{\text{cfd}} = \{(\mathbf{x}_1, y_1, \mathbf{z}_1), \ldots, (\mathbf{x}_n, y_n, \mathbf{z}_n)\}$. Here $\mathbf{x}_i \in \Re^m$ is the m-dimensional feature vector, $y_i = \{\text{cf}_0\}$ is healthy or $y_i \in \{\text{cf}_1, \ldots, \text{cf}_p\}$ is the fault label (when known) associated with the feature vector $\mathbf{x}_i$, p is the number of fault cases, $i = 1, \ldots, n$, and n is the number of signatures. $\mathbf{z}_i$ is the vector of expected baseline link properties (e.g. link delay, bandwidth etc.) of the sample. The feature vector $\mathbf{x}_i$ combined with the class label y_i and link properties $\mathbf{z}_i$ is called the "raw signature" of the i^{th} instance.

3.2.5 Creating NSSs

Figure 3.3 shows the raw values of the sub-set of the first 40 features for two UDs, one healthy and one suffering from a fault. The figure shows that significant portions of the features have values that differ greatly between the healthy and faulty cases. The patterns of these differences vary for every fault and provide the key distinctions needed for a unique identification. Also, with a $log - y$ axis, the figure shows that the features in their raw form have significantly different scales. The scale variations are expected because of the myriad of types of statistical at-

tributes considered.

For a given access link, traces captured from a controlled healthy UD with optimal system settings provide the baseline performance signature. Capturing the healthy signature if part of the system initiation process where network operator place multiple healthy UDs in the network as active probes to generate sufficient healthy traces to provide a consistent baseline. The healthy signature is updated periodically depending on the dynamic network behaviour to account for network changes over time by automatically triggering the signature generation process. We normalise the raw features in the suspected faulty signatures against the healthy baseline features for the particular access link. Given that $\mathbf{x}$ is a m-dimensional feature vector, $(\mathbf{x}_{\text{cf1}}, \text{cf1}, \mathbf{z})$ is the sub-set of raw faulty signatures, and $(\mathbf{x}_{\text{cf0}}, \text{cf0}, \mathbf{z})$ is the sub-set of raw healthy signatures,

$$\text{normalised } \mathbf{x}^k_{\text{cf1}} = \frac{\mathbf{x}^k_{\text{cf1}} - \mu}{\sigma} \tag{3.1}$$

where, μ and σ is the mean and standard deviation of the population $\mathbf{x}^k_{\text{cf0}}$ for feature $k = 1, ..., m$. The vector of these normalised features together with the class labels are called the NSS. The normalisation of the signatures (i) re-scales the features to a more manageable range, (ii) provides a baseline to all the signatures for a clearer and easier comparison, (iii) helps reduce computational complexity and improve the detection accuracy later when used in machine-learning algorithms, and (iv) improves stability of the features in a dynamic network and the portability of the signatures between networks.

Figure 3.4 shows the first 50 features of a normalised signature or an NSS. The feature values of two different faults have been standardised against the same baseline, and the figure shows a clear difference between the signatures. These differences are the fingerprint of a fault that can be detected using various pattern identification mechanisms later for a automated diagnosis.

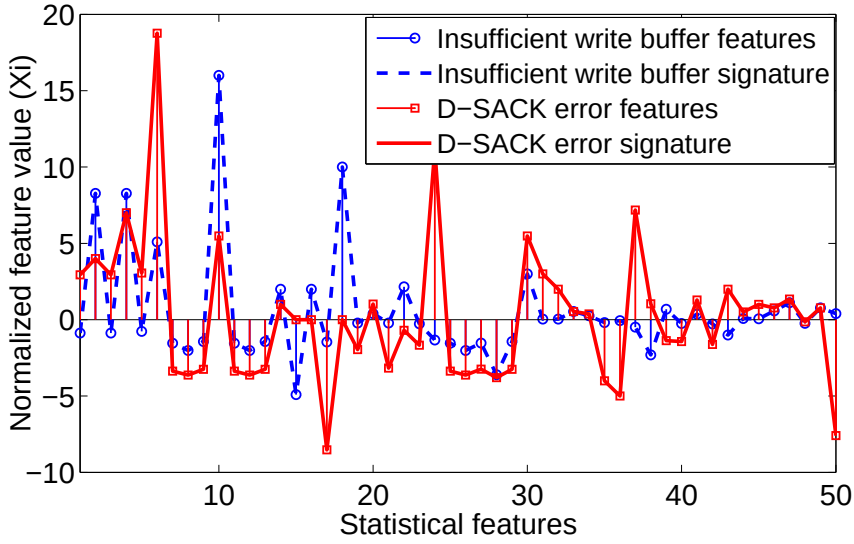


Figure 3.4: Comparison of the first 50 features in NSSs for UDs with insufficient buffer and D-SACK errors.

3.2.6 Fault severity

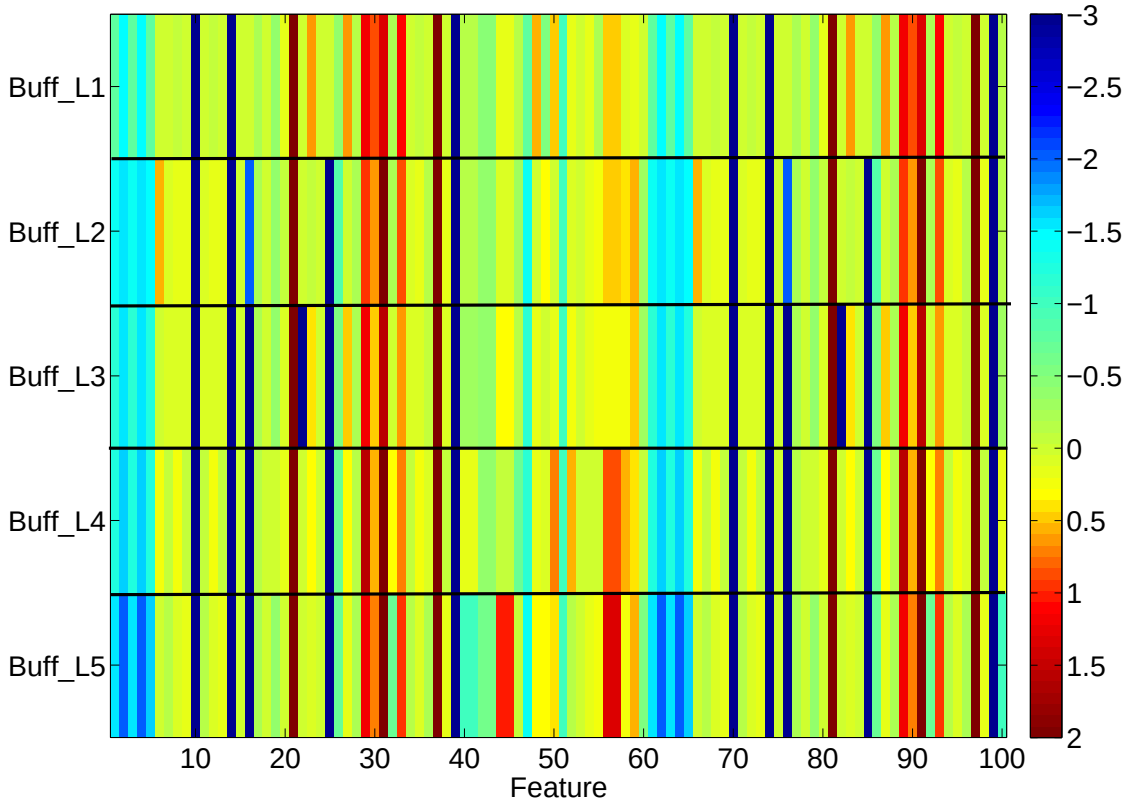


Figure 3.5: Comparison of the first 100 features in NSSs for varying levels of read buffer limitation at the UD.

Faults suffered by the UDs can be boolean in nature (e.g. disabled or enabled options) or can take a range of values and have a proportional effect on the performance. For example, the receive buffer level of the device, depending on the size, can have a negligible or significant impact on the throughput. The boolean types of faults are easily identifiable from NSSs, as evident in Figure 3.2b and Figure 3.2c. The existence of a "continuous" type of fault is also effectively characterised by the NSSs, as shown in Figure 3.2g and Figure 3.2i.

The ability to characterise the severity of the fault, at least to a couple of discrete levels, can have significant benefits in a diagnostic application. To identify the NSSs' capability, we varied the severity in 6 levels for various buffer and queue limitations and collected data over the campus network. Figure 3.5 shows a comparison of partial NSSs (100 features) for the case of insufficient read buffer. The features are scaled to a [-3,2] range and colour mapped for easy comparison. The figure clearly shows that some features show distinct intensity variations following the fault level. A cascading level diagnostic system can first use the overall NSSs to identify the type of the failure and further continue to narrow down the level of the failure using a specific sub-set of features.

3.2.7 Advantages of NSSs

For the root cause diagnosis of UD performance problems, the proposed signatures and the generation process offer many advantages, including the following:

- A large feature set means even the minute variation of UD behaviour can be captured. The most suitable feature sub-set can be selected on a fault basis instead of depending on a smaller number of common features to characterise all the faults.

- The aggregated statistical features capture the overall connection behaviour instead of specific packet sequences. The behaviours of individual TCP im-

plementations (between platforms) are avoided.

- The signatures are solely based on TCP packet traces. No other information is needed.
- Healthy baseline is periodically updated to account for normal network variations.
- Data for the NSSs can be sourced without remotely accessing or physically logging on to the systems. This is a capability unavailable in many network diagnostic tools.
- No kernel modifications, software installation, or specialised knowledge is needed at the UD. The portability of the modules simplifies the implementation.
- The same signatures can be used for identifying soft failures in intermediate nodes in a network by deploying a trace collection module in a neighbouring node.

3.3 Data collection

Healthy and faulty signatures can be collected from live networks such as campus LANs, enterprise networks or ISP networks. However, collecting sufficient samples from live networks to conduct research with statistically robust analysis poses a number of challenges. Obtaining labelled data which is accurate and representative of all types of behaviours is often prohibitively expensive, due to the amount of time and effort required. Learning-based methods require data not only for testing and comparison but also for training, resulting in even higher data requirements. The data used for training needs to be representative of the network behaviour to which the learning-based method will be applied, possibly

requiring generation of new data for each deployment. Labelling is often done manually by a human expert and hence substantial effort is required to obtain the labelled training data set. Typically, obtaining a labelled set of anomalous data instances covering all possible types of anomalous behaviour is more difficult than obtaining labels for normal behaviour. Moreover, anomalous behaviour is often dynamic in nature. For example, new types of anomalies might arise for which there is no labelled training data and this can introduce inconsistencies into labelled data sets.

The collection of sufficient high quality data is not a problem unique to the present research. Most ML and ML-based application research faces similar issues and a number of solutions have been proposed in the research literature. Some studies, mostly dealing with smaller data sets, opt for manual labelling [226, 37]. Considering all the possible scenarios and the number of samples we required, manual labelling was not a feasible solution in our particular study. Another option is to develop a specific set of algorithms, either heuristic-based or more advanced, such as active learning methods [227] to create the labelled data set. However, this is a deviation from the main aims of this research, and such techniques were deemed beyond the current research scope. Another common approach in the research community is to use established data sets with already sampled data [228]. Unfortunately, there are no data sets currently available for network soft failures in UDs. A final compromise some studies have resorted to is to inject a set of anomalies deemed representative of the kind the anomaly detection system should detect [229]. A number of techniques have been proposed that inject artificial anomalies in a normal dataset to obtain a labelled training dataset [230, 231, 232]. Whilst artificial anomaly injection works in some situations, the complex nature of the network signatures meant that an accurate representation of the real-world networking environment and a specific scenario cannot be achieved by artificially manipulating the data.

After evaluating all available options in the literature for data collection, we opted to use a combination of network test-beds, fault injectors and network emulators to re-create a select set of UD failures in a number of controlled and uncontrolled (live) networking environments. We used a purpose-built fault injection module, and instead of injecting artificial anomalies into the collected data, our fault injector module injected faults into the UD itself. This approach provided a number of advantages over all the other options previously mentioned:

(i) Fault injection creates more realistic data. Once a fault is injected into a UD, the behaviour is identical to an actual device suffering from the same fault.

(ii) Data can be collected in a controlled environment with known network parameters and stable end-points

(iii) Data collection can be automated

(iv) Data collection can be parallelised

(v) A range of scenarios and network conditions can be re-created with minimal effort

(vi) The actual faulty scenario is fully known, and no unknown hidden issues pollute the dataset

(vii) The data can be accurately labelled without the additional overhead of manual processing

(viii) It is not necessary to have buy-in from users to engage with the system when experiencing problems

(ix) It eliminates privacy concerns of real users and complexities around compliance procedures.

The dataset created during this research was collected over diverse networking environments, with a variety of link conditions, and a range of UD soft failures and TCP types. We created 16 types of UD conditions, 5 of which had varying levels of severity, and used 8 TCP variants over 3 distinct network test-beds with changing link parameters. For each of the unique scenarios, a number of samples were collected at different times to enable a more robust and realistic analysis. The final data set consisted of 1.2 million samples of TCP traces and the extracted NSSs, making five terabytes of data in total. All the samples are accurately labelled and both raw data captures and extracted signatures have been preserved with the highest level of granularity possible. To our knowledge, no such data set of UD failures exists and this is a key contribution to the networking and ML research community.

3.3.1 Fault injection into UD

We created a fault injection module to re-create some common problems in UDs. We limited the UDs to personal computers as they are the most common UDs. We used a Linux (Kernel 2.6.32 with Ubuntu distribution) operating system in the UD as it offers the most flexibility in terms of manipulating the system configuration to re-create some common UD network performance issues. The Linux-based UD also offers the flexibility to switch between TCP types, including those that are used in other operating systems such as Microsoft Windows. Given that our data samples are purely TCP traces, this flexibility offered us the capability to emulate a variety of UD types, especially the TCP layer of the UD using a single device.

The faults that were injected were only indicative sets of common issues, representing a very small sub-set of all the possible scenarios. The objective was to re-create a select few UD faults so that proposed frameworks and systems could be evaluated with realistic data. The following list describes the various faults

injected into the device in addition to the healthy UD scenario.

(i) **Disabled SACK error:** Selective acknowledgement (SACK) capability of the TCP layer is critical to the perceived bandwidth, particularly when the network has inherent packet loss. Some OSs and devices come with the TCP SACK option disabled by default and sometimes system administrators disable the SACK option across the whole organisation due to concerns over support with legacy middle-ware. Disabling SACK may have a negative impact on the user, particularly when performing high-speed transfers over lossy networks.

(ii) **Disabled D-SACK error:** Duplicate Selective Acknowledgement (DSACK) is another feature to improve the bandwidth of a connection over a lossy network by avoiding unnecessary re-transmissions [211, 51, 233, 234]. This capability is also disabled by default in some devices. Some applications installed on the device can override DSACK configurations and create unintended performance bottlenecks when the device moves from a non-lossy network to a lossy network.

(iii) **Insufficient write buffer:** The write buffer of the device is crucial when sending data through the network, especially a single large file. Insufficient write buffer memory allocations can create a bottleneck in the end-to-end connection [208, 28]. Whilst most middle-ware can auto-tune the write buffers, UDs usually come with limited auto-tuning capability and conservative buffer levels. When artificially limiting the buffer levels to produce this common network bottleneck, we used five different levels of severities as distinct cases to provide more granular data.

(iv) **Insufficient read buffer:** The read buffer of the device is crucial when receiving data through the network, especially large files. Insufficient read

buffer memory allocations can create a bottleneck in the end-to-end connection. UDs usually come with limited auto-tuning capability and conservative buffer levels. We used five different levels of severities by limiting the buffer capacity at distinct levels to create more granular data.

(v) **Simultaneously insufficient read & write buffer:** This particular scenario includes both a write buffer limitation and a read buffer limitation in the same UD. Having two different types of faults in a single signature increases the complexity of the diagnosis, and the NSSs in this special scenario were used to evaluate the system's robustness and capability in such a scenario.

(vi) **TCP timestamps are not working/in error:** Timestamps are a useful feature in the network stack, especially to keep track of round trip times and sequencing. Timestamps are used by a number of different algorithms to optimise the connection characteristics. Disabled timestamps have a negative impact on the connection speeds and the stability of the connection.

(vii) **Window scaling error:** The window scaling capability of the networking stack is vital to utilise the bandwidth offered by high-speed network links. Disabling this capability severely hinders the users from using the full bandwidth on offer. It can cause a user's Internet connection to malfunction intermittently for a few minutes, then appear to start working again for no reason. Although most users are unaware of it, this is one of the most common issues that create a performance bottleneck in UDs (e.g., when the "public" Internet connection type is chosen in Windows OS, this overrides the normal window scaling behaviour and restricts the window to a lower level, creating a bottleneck.)

(viii) **Limited reordering threshold:** Re-ordering is a common occurrence in lossy networks. Depending on the network characteristics, the default reordering threshold can be insufficient to properly manage the packets that

are out of order and trigger re-transmission requests. This will reduce the perceived bandwidth of the user.

(ix) **Link-UD speed mismatch:** Most middle-ware (e.g. routers) and UDs come with auto-negotiation capability to synchronise link speeds between the router and the device. However, in some instances auto-negotiation fails or is not supported. In such situations, the link speed offered by the router can be different to the link speed configured at the UD. This issue is difficult to diagnose and manifests only as a performance problem. We created a number of levels of mismatch, for example, 100 Mb/s with 10 Mb/s and 1000 Mb/s with 10 Mb/s.

(x) **Duplex mismatch:** Similar to the previous case, it is crucial for the link (router) duplex mode (full-duplex, half-duplex) to match the device configuration. When the duplex modes are mismatched, the user will notice severe performance problems and stability issues in the connection. This is a common issue in corporate environments, particularly when there are legacy middle-ware devices in use.

(xi) **Link-UD speed mismatch & duplex mismatch:** This scenario created both faults in the same UD simultaneously. If the issue of failed synchronisation is caused by the faulty middle-ware, it is likely that both attributes will be mismatched between the link and the UD.

(xii) **UD firewall causing packet loss:** Most devices today have a firewall either pre-installed or installed as third-party software. The firewall adds an extra layer of security to the network communications, which means that data has to be scanned at the application layer before reaching the user. When poorly configured, firewalls can cause packet losses, which will be perceived by the user as a loss of bandwidth. For a non-technical user, it is often difficult to find out if the firewall is causing the issue.

(xiii) **UD firewall causing packet delay:** Similarly to the previous case, firewalls can delay the packets from reaching the application. This can be compounded by deeper levels of security restriction and additional features such as deep packet inspection of some firewalls. We specially injected firewall packet delays to understand if the NSSs from such delays are different to delays that are caused by the network or data-link layer.

(xiv) **Overloaded UD CPU:** One of the most common network bottlenecks is caused by overloaded CPUs. This can be due to the number of simultaneous applications users run, the small capacity of the CPU - especially in mobile devices, or due to a virus or malware in the device. Overloaded CPUs can impact not only on large file transfers, but also on bursty traffic like web browsing.

(xv) **Overloaded UD memory:** Another common issue causing slow network performance is the lack of a device's random-access memory or RAM. This can be due to the RAM utilisation of other applications or insufficient RAM provided in the device. Even if the read and write buffers have been correctly allocated, users will experience slow or interrupted connections if the device RAM is not sufficient.

(xvi) **UD HDD I/O overloaded - faulty:** Most data that is transmitted into or out of the UD interacts with the device's hard disk (HDD). Hard disks have input and output (I/O) operation limits which determine the number of operations the disk can handle in a second. Despite disk I/O always being much higher than network I/O, it can still be overwhelmed by HDD-heavy applications, faulty applications, rogue malware, a faulty HDD hardware, a faulty HDD driver or a faulty I/O bus. In these scenarios, the perceived network data transfer rates can suffer.

3.3.2 TCP variants

Section 2.8.1 above provided details of various implementations of TCP. We used the TCP variants in Table 3.2 in the UD by dynamically changing the system configurations to activate and de-activate the TCP variant being used. Since each TCP variant behaves slightly differently, the data collected across these various TCP implementations is important to demonstrate NSSs ability to generalise the overall behaviour using aggregate statistics. Data from different TCP variants also provide test samples to evaluate the ability of the proposed diagnostic methodologies to cope with minor changes between the TCP variants.

TCP variant
TCP Reno
TCP New Reno
TCP BIC
TCP CUBIC
High-speed TCP (HSTCP)
TCP Vegas
TCP Westwood
TCP Hybla

Table 3.2: TCP variants used in the UD

3.3.3 Network test-beds

Throughout this research, we used the following test-beds to collect the sample data.

Test-bed 1: Laboratory controlled network with emulated links

The first test-bed re-created the scenario shown in Figures 3.6 and 3.7. This set-up offered the best control over all the elements in the end-to-end connection, including the UD and access link. The test-bed was established to create the most generic usage scenario of the diagnostic system: a UD connecting to the access

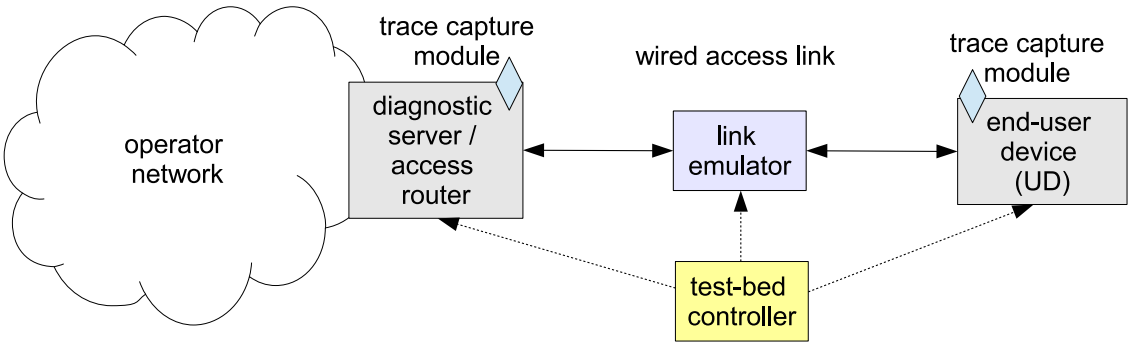


Figure 3.6: Test-bed 1: An access router, directly connected to a UD over a wired access link emulated using dummynet-based link emulator.

router directly through the access link. The access router sits at the edge of the service provider network and in this particular scenario, is co-located with the diagnostic server.

The UD and the diagnostic server ran on Linux 2.6.32 with Ubuntu distribution, capable of running many TCP variants. The access link was emulated using a network emulator, dummynet [235] on FreeBSD 7.3. The test-bed controller was used to orchestrate the events in a pre-configured order so that the data collection process could be automated.

The connection link between the UD and the access router is of particular inter-

Figure 3.7: Monash University laboratory setup of the test-bed 1

est to us in this research. The link properties, such as bandwidth, latency, packet loss, bandwidth-delay product (BDP), and middle-ware queues, have an impact on some of the NSS features. To evaluate and model the impact, we required data across a range of links, which would have been prohibitive in a live network. However, by using a link emulator, we can dynamically change the link characteristics such as link latency and packet loss, effectively emulating the effects these parameters will have on the NSSs.

We used FreeBSD-based dummynet, a live network emulation tool, originally designed for testing networking protocols, and since then used for a variety of applications including bandwidth management. Dummynet has been commonly used for running experiments in user-configurable network environments [236]. It simulates/enforces queue and bandwidth limitations, delays, packet losses, and multi-path effects using a mechanism called *pipe*. More details on the operating mechanism and the configuration of dummynet can be found in [235, 236].

We particularly focused on emulating the effects of bandwidth, latency and packet-loss level link parameters. The matrix of emulated link parameters can be found in Table 3.3. When emulating packet losses, dummynet uses a uniform random loss model that can be configured to a specified loss probability. However, it is commonly understood that in real world networks, packet loss often occurs in bursts. Consequently, dummynet's emulation of packet losses is most likely not representative of all Internet packet losses. With test-bed 1, our objective is to collect aggregated statistics of the overall packet traces to establish a baseline healthy signature and faulty signatures for evaluating the proposed systems within a controlled laboratory condition. Since, we are consistently using dummynet to emulate packet losses in our controlled environment, all signatures generated on test-bed 1 are based on the same uniform loss model. We subsequently extend our datasets with real-world networks in test-beds 2 and 3 to capture data with more realistic bursty packet loss behaviour.

Bandwidth (Mb/s)	Latency(ms)	Packet loss rate (%)
120, 104, 88, 72, 56, 48, 40, 32, 24, 16, 9.6, 5.6, 4, 1.6, 0.8	0, 5, 10, 15, 20, 25, 30, 40, 50, 60, 70, 80, 90, 100	0 , 1, 5, 10, 15, 20

Table 3.3: Emulated link parameter range

Test-bed 2: University local area network (LAN)

The second test-bed shown in Figure 3.8, instead of emulated links, used the university LAN as the network between the UD and the access router/diagnostic server. This set-up was created to collect real-world data that would be identical to any enterprise LAN network and private cloud network. Instead of a direct link between the two end-points, this set-up introduced a multi-hop path with at least three routers and a number of switches between them. The UD was connected to LAN through a router that assigned a subnet IP address to the UD in one scenario and used network address translation (NAT) to assign an IP in the second scenario. This was done to capture the effects of NAT on NSS, as NAT is a common feature in network set-ups.

We used the live network with subnets that were already being used by day-to-day university activities, including a large number of students conducting their studies. The data collection was done throughout the day and throughout the semester so that there are variations in the network load. The data collected in

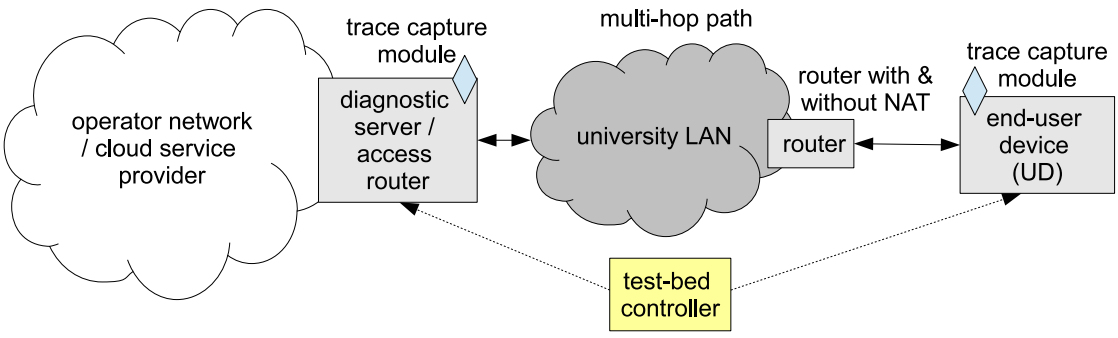


Figure 3.8: Test-bed 2: Collects data over multi-hop paths in real-world enterprise networks with live cross-traffic.

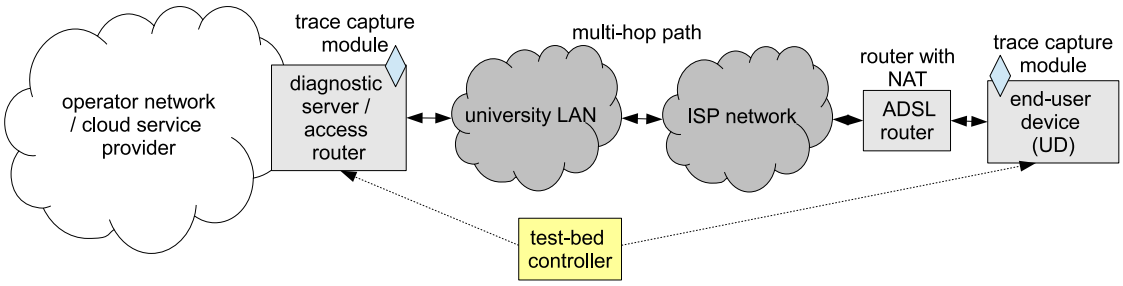


Figure 3.9: Test-bed 3: Connects through ISP core network and ADSL access links.

this test-bed was used in evaluating the accuracy of the developed techniques against real-world data.

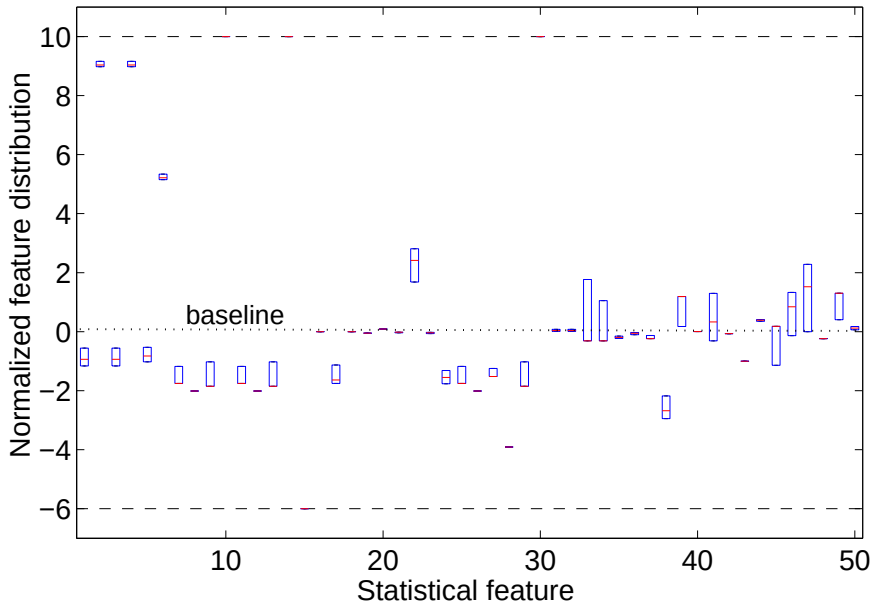
Test-bed 3: Monash University LAN+ISP ADSL

The third test-bed shown in Figure 3.9 was a complex set-up that included the ISP network and ADSL connection. This test-bed collected data samples that are closest to the scenario where a public user is accessing the diagnostic service. The last-mile access link of the ISP is a complex connection with a number of factors affecting the link performance. We used home routing equipment at the UD and a commercially available ADSL connection to re-create a common networking scenario. Due to the commercial implications around data usage, we limited the scope of the data collection to TCP New Reno and conducted only one round of sample collection (approximately 1000 samples). The sample collection on this network is still ongoing and will continue during future work to be used to evaluate complex scenarios.

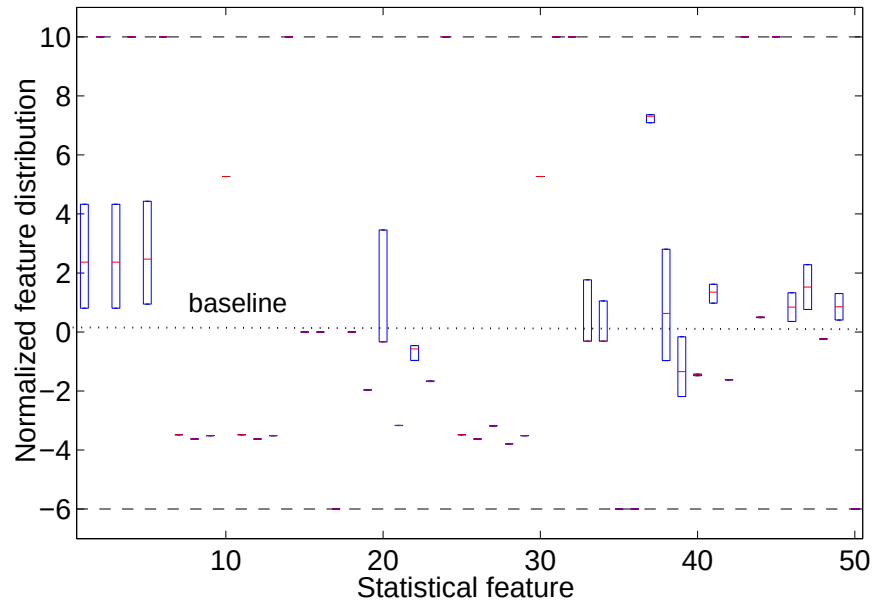
3.4 Types of features

Even under exactly the same conditions, network connections always incorporate a noise component. While the NSS extraction process yields unique signatures for each fault, it is necessary to investigate the influence the statistical noise and variations inherent in live end-to-end connection have on the stability of the features.

The stability of a feature can be defined as consistent feature magnitude across multiple samples collected over the same link under the same end-point condi-

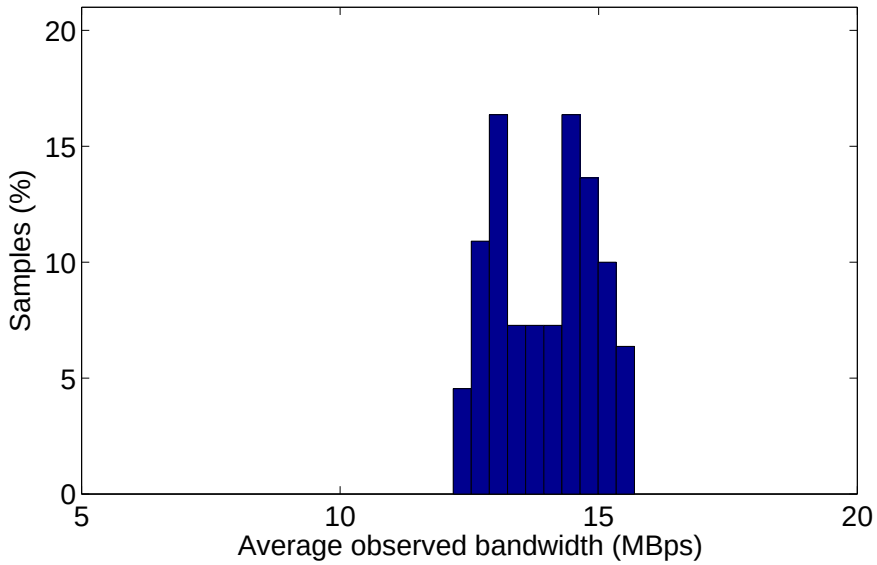


(a) Write buffer error

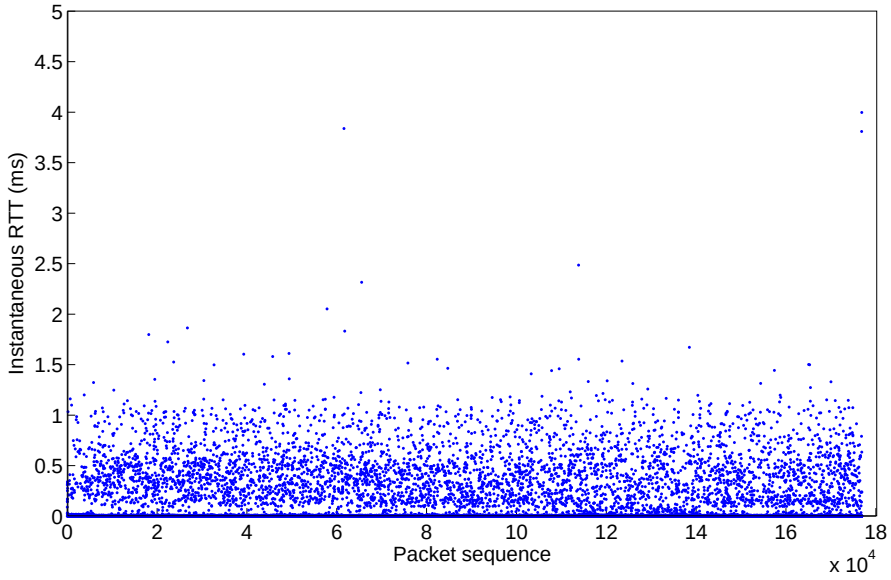


(b) Read buffer error

Figure 3.10: Variance of normalised values of the first 50 features in NSSs for 30 samples when collected over a live network link with variations shown in Figure 3.11a and Figure 3.11b. In Figure 3.10a and Figure 3.10b, the red line of each feature marks the median while the bottom and top edges of the box mark the 25th and 75th percentiles. First 50 features of the 460 NSSs features have been selected as representative to clearly visualise the distributions.



(a) Histogram of the bandwidth variations observed



(b) Instantaneous latencies of the links

Figure 3.11: Link properties for the samples used in Figure 3.10

tions.

As mentioned in Section 3.2, NSSs have 460 features. Figures 3.10a and 3.10b show box plots of first 50 normalised features taken from 30 samples of NSSs with write buffer and read buffer errors. To improve the clarity of the visualisations, we have chosen only to visualise first 50 features as representative instead of all 460, although all of the features are being used for the analysis. These samples

were collected over the live campus network with other active users. Figure 3.11a shows the average bandwidths in MB/s, and Figure 3.11b shows the instantaneous latencies observed between the faulty UDs and the server during the data collection.

In Figures 3.10a and 3.10b, the red line in each feature marks the median, while the bottom and top edges of the box mark the 25th and 75th percentiles. These two figures show the statistical variations of the features when collected over end-to-end connections that had bandwidths between 12 MB/s to 16 MB/s and RTT between 0.2 ms to 4 ms. From Figures 3.10a and 3.10b, we can clearly identify three types of features.

Stable-significant features: These are features that have only a small variance around the median with a clear deviation from the baseline. The features remain stable within a range of bandwidths, as shown in Figure 3.11a and a range of latencies, as shown in Figure 3.11b. These are the primary features that contribute to a unique identification.

Stable-Null features: These are features that have small variance around the median, yet show no deviation from the baseline. These features do not contribute as much information as the significant features for a unique identification, but in combination with significant features, they are useful to create unique feature patterns.

Unstable features: These features show a significant variation around the median. Although machine-learning algorithms can compensate for a certain amount of statistical noise, most of these features will have a negative impact on diagnosis.

The figures show that for every fault, only a sub-set of features provides consistent and valuable information for an identification. These sub-sets of features also vary with the type of fault, as observable in Figures 3.10a and 3.10b. The

large feature set, therefore, enables the NSS to characterise a wide range of faults through a single representation. A subsequent "feature selection" algorithm or complexity reduction algorithm can be used to filter out null/unstable features for individual fault types for use in an ML-based fault diagnostic system.

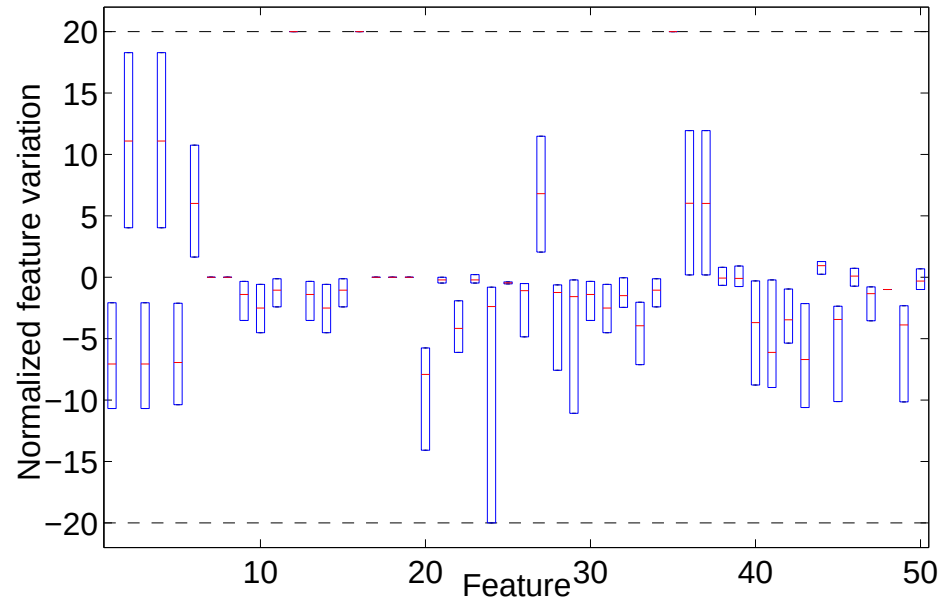
3.5 Link dependency of the features

The properties of the access link have a significant impact on the NSS. An analysis of NSSs collected over different link conditions was performed to identify how link variations can impact the signature.

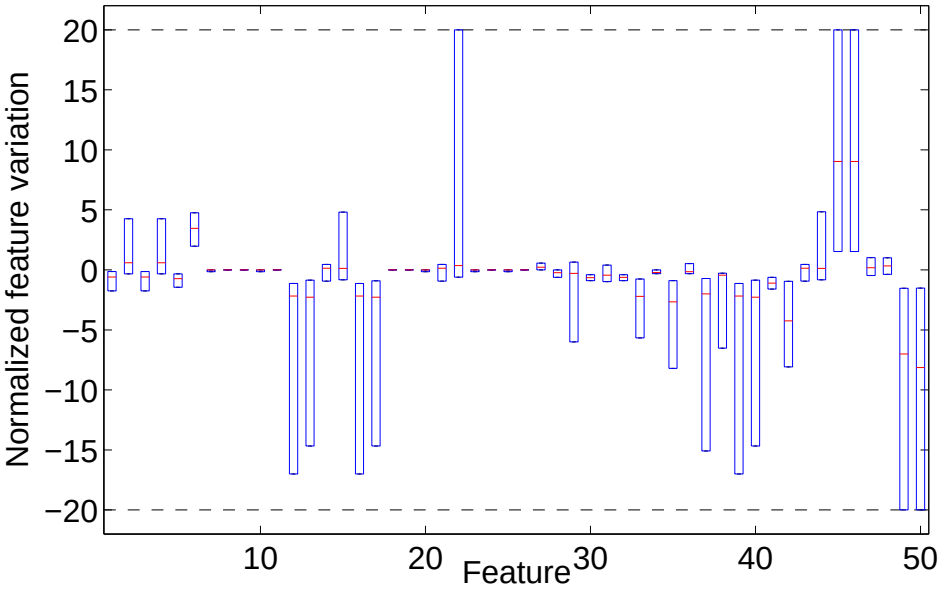
Figure 3.12 shows the variance of first 50 normalised features in terms of box plots taken from NSSs of the same fault over various links. The data was collected using the laboratory test-bed where links with different maximum bandwidth and minimum latency combinations were emulated. Figures 3.12a and 3.12b show the feature variance for 200 sample NSSs collected over links with variable minimum latencies of 5 ms-100 ms and constant maximum bandwidths of 5 MB/s and 15 MB/s, respectively. Figures 3.12c and 3.12d show the feature variance for 200 sample NSSs collected over links with variable maximum bandwidths of 100 kB/s-15 MB/s and constant minimum latencies of 20 ms and 100 ms, respectively. First 50 features of the 460 NSSs features have been selected as representative to clearly visualise the distributions. These experimental results show that whilst a small sub-set of features remain link-independent, the majority of the features are affected by link properties such as latency and bandwidth.

The capabilities of supervised ML systems are limited by the variations included in the training samples. A diagnostic system can be trained with only a small set of samples when the link properties remain relatively stable, as in the case shown in Figure 3.11. In such a system, the features (filtered) are capable of accounting for smaller variations in the network, thus allowing a successful diag-

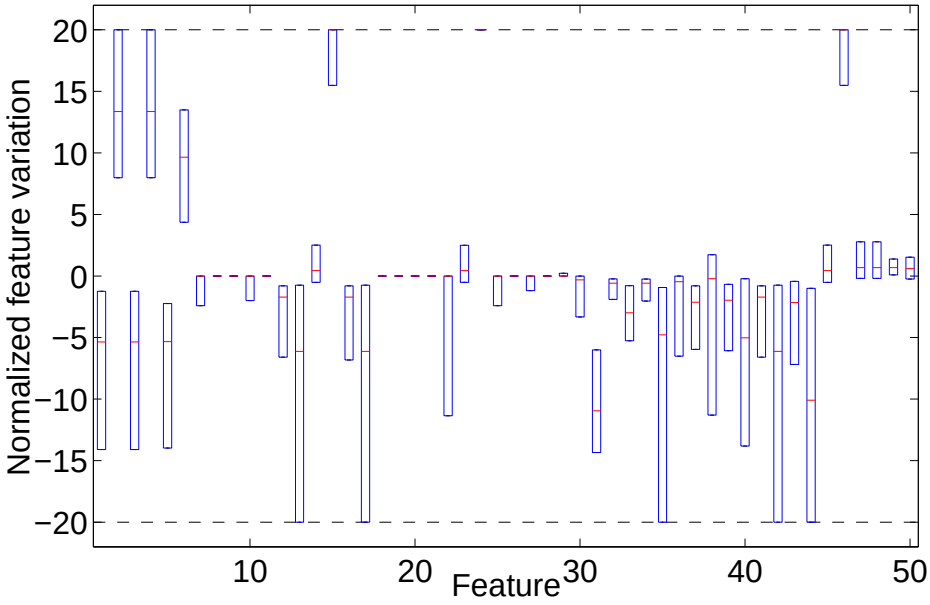
nosis. However, as Figure 3.12 shows, most features show a large variation when considering significant changes in the link profile.



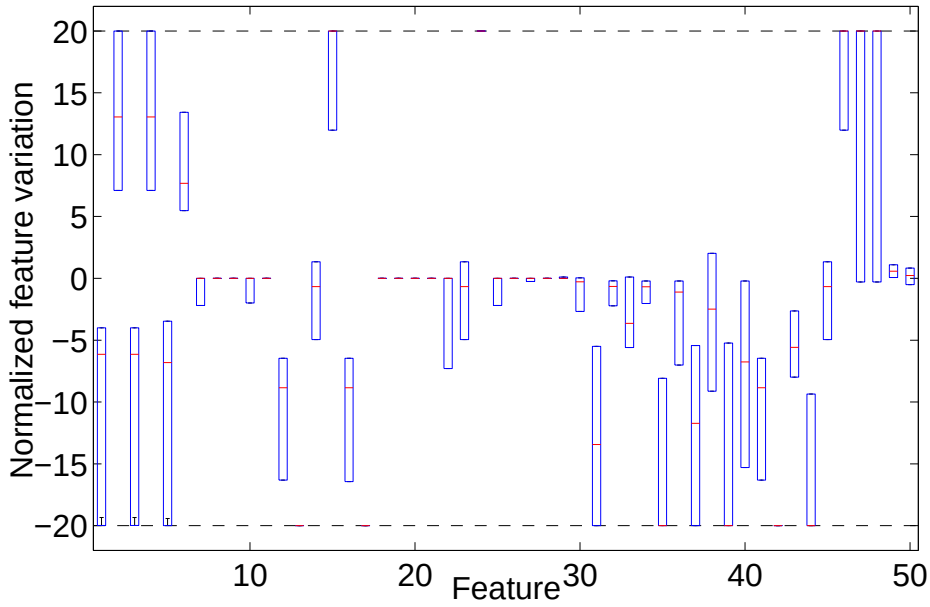
(a) Bandwidth=5 MB/s, Latency=0 ms - 100 ms



(b) Bandwidth=15 MB/s, Latency=0 ms - 100 ms



(c) Bandwidth=100 kB/s-15 MB/s, Latency=20 ms



(d) Bandwidth=100 kB/s-15 MB/s, Latency=100 ms

Figure 3.12: Variance of normalised values of the first 50 features in NSSs when samples are collected over links with different latency and maximum bandwidth properties. First 50 features of the 460 NSSs features have been selected as representative to clearly visualise the distributions.

3.6 Link-adaptive signature estimation (LASE)

As discussed in previous sections, NSS features become inconsistent when samples are collected over links that have significantly different properties. Link char-

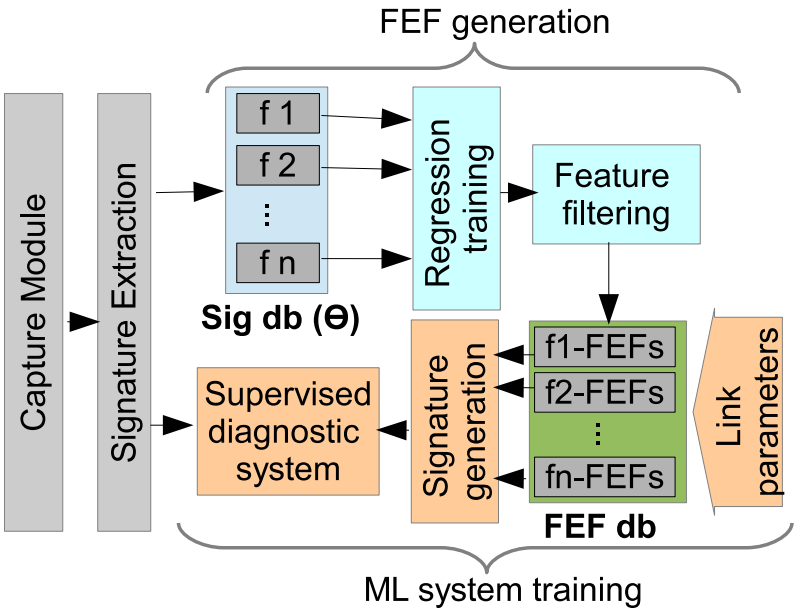


Figure 3.13: Operational stages of LASE and diagnostic system training

acteristics, such as maximum bandwidth, minimum latency, packet loss, congestion and packet reordering, change from network to network, which makes the portability of a diagnostic system between networks challenging. Training a single diagnostic system for exhaustive combinations of link parameters requires an impractically large dataset collected over a wide range of link conditions. Consequently, there is a trade-off between generalising the ML diagnostic system and minimising the number of NSS samples required for training. Analytical modelling of the performance relationships between link properties and connection characteristics has been attempted in the literature [56] as a solution but has thus far proven unsuccessful.z

To reduce the amount of training data needed and to avoid the need for a complex analytical model, we propose the concept of link-adaptive signature estimation (LASE).

3.6.1 Operational overview of LASE

LASE is a regression-based estimation technique, designed for modelling the behaviour of a feature with changes in link conditions. This is achieved by collecting a small set of samples for each UD fault, while varying link parameters within a specific range. These samples are then used to model a feature estimator function () based on multivariate regression techniques. When NSSs are needed to train a new diagnostic system for a particular network, these FEFs can be used to estimate the features for the specific links (within the link parameter training range) and create the NSSs for a specific fault. These link-specific signatures can then be used to train the ML system. As discussed before, for every fault, features can be identified as either stable or unstable. Prior to LASE training, the unstable features are filtered from the NSSs, as these features can affect the consistency of signatures.

Figure 3.13 shows the two main operational stages of LASE: (i) FEF generation, and (ii) training of the supervised ML system. The "capture" and "signature extraction" modules collect controlled healthy and faulty NSSs over different links with delay, bandwidth, loss, etc. parameter combinations. These NSSs, together with the link parameters, are then sent for regression training.

Regression is the process in which models (FEFs) are fitted to observational data to identify the complex relationship between dependent variables (feature values) and independent variables (link parameters). Assume that the number of faults considered is p, each NSS vector is comprised of m features, and n sets of NSSs are available for each of the p faults from k number of link parameter combinations, each denoted by the vector $\mathbf{z}_k$. Then regression training samples for each fault will be a $n \times m$ matrix of feature values and parameter labels as follows:

$$\begin{bmatrix} (f_1^1, \mathbf{z}_1) & (f_2^1, \mathbf{z}_1) & \cdots & (f_m^1, \mathbf{z}_1) \\ (f_1^2, \mathbf{z}_2) & (f_2^2, \mathbf{z}_2) & \cdots & (f_m^2, \mathbf{z}_2) \\ \vdots & \vdots & \ddots & \vdots \\ (f_1^n, \mathbf{z}_k) & (f_2^n, \mathbf{z}_k) & \cdots & (f_m^n, \mathbf{z}_k) \end{bmatrix}$$

Regression algorithms use each column; n samples of the feature f_m collected over k different links as the input and model a FEF to estimate the behaviour of f_m within the parametric range of $\mathbf{z}_k$. Consequently, once the FEFs are modelled for every fault, a $p \times m$ matrix of FEFs as follows will be created. In this matrix, each row is a vector of FEFs associated with a single fault. These FEF vectors are then used to estimate the link-specific feature vector of a NSS.

$$\begin{bmatrix} \text{FEF}_1^1 & \text{FEF}_2^1 & \cdots & \text{FEF}_m^1 \\ \text{FEF}_1^2 & \text{FEF}_2^2 & \cdots & \text{FEF}_m^2 \\ \vdots & \vdots & \ddots & \vdots \\ \text{FEF}_1^p & \text{FEF}_1^p & \cdots & \text{FEF}_m^p \end{bmatrix}$$

We define a feature's "predictability" as the FEF's ability to closely match the estimated feature values with real sample data. To evaluate the predictability, we use a common goodness of fit measure "coefficient of determination" (R^2), which is defined by $R^2 = 1 - \text{SS}_{err}/\text{SS}_{tot}$, where SS_{err} is the sum of squares of residuals and SS_{tot} is the sum of the squared differences from the mean of the feature. R^2 can be between 0 and 1, with a value closer to 1 indicating a more predictable feature. As not all features are perfectly predictable, the FEFs are tested for estimation accuracy and only the FEFs of predictable features for each

UD fault are stored in a database.

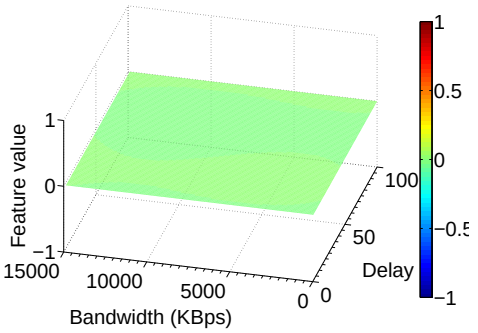
The regression technique largely dictates the accuracy of the model and is an important design criterion. To build the FEFs, we used three different techniques: (i) polynomial regression with least-square optimisation (Poly) [237], (ii) general regression neural networks with Levenberg-Marquardt training (GRNN) [238, 239], and (iii) support vector regression with radial-basis kernel (SVR) [140]. The details of these methods can be found in the references.

When training a diagnosis system for a specific link, FEF-generated NSSs can be used either entirely, or in combination with those collected over the live network. Using LASE, diagnostic systems can be trained to suit the exact networking environment, however dynamic, with a minimal set of sample data. LASE also provides the adaptive capability to an ML-based diagnostic system where systems can automatically alter the learning data sets, depending on the prevailing network conditions.

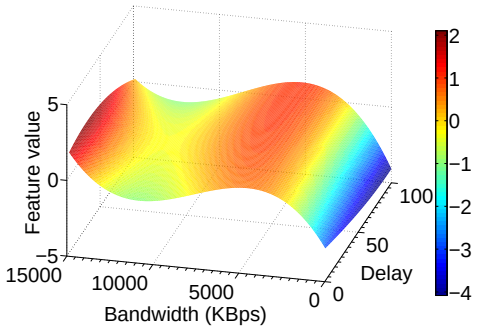
3.6.2 Experimental evaluation of LASE

The performance of the LASE system was evaluated experimentally using network data collected in the laboratory network test-bed shown in Figure 3.6 (TB-1 data set). The LASE was evaluated for common bottlenecks and details of four cases are presented here. We collected NSS samples whilst varying bandwidths between 100 kB/s-15 MB/s, and link delays between 0 ms-100 ms, which covers typical residential access network performance to the high speed Intranet. Although many link parameters affect the signature, at this stage, we have investigated only bandwidth and delay variations, as they are the most common in access links. We assume that links or intermediate devices do not suffer from any degradations.

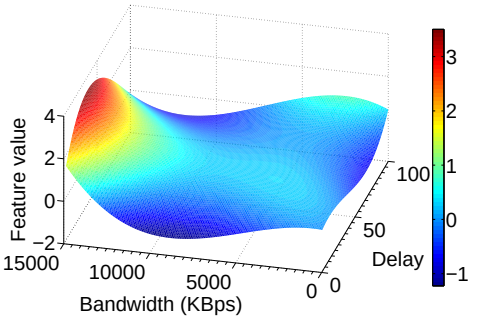
Faults were emulated at the UDs, and the TCP traces were collected using the capture module. Two datasets with varying link bandwidth and delay com-



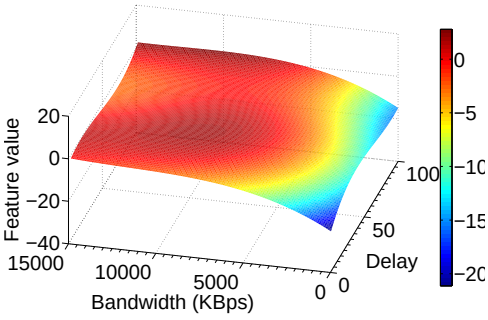
(a) Healthy



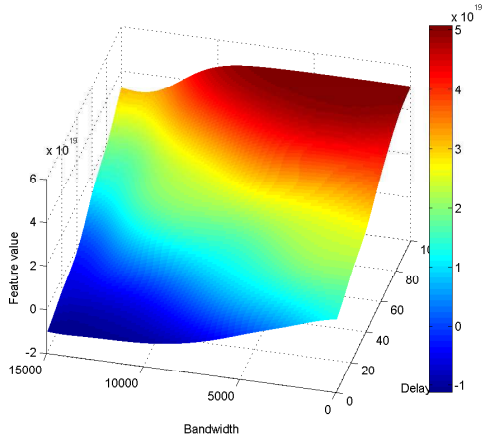
(b) Disabled SACK error



(c) Disabled DSACK error

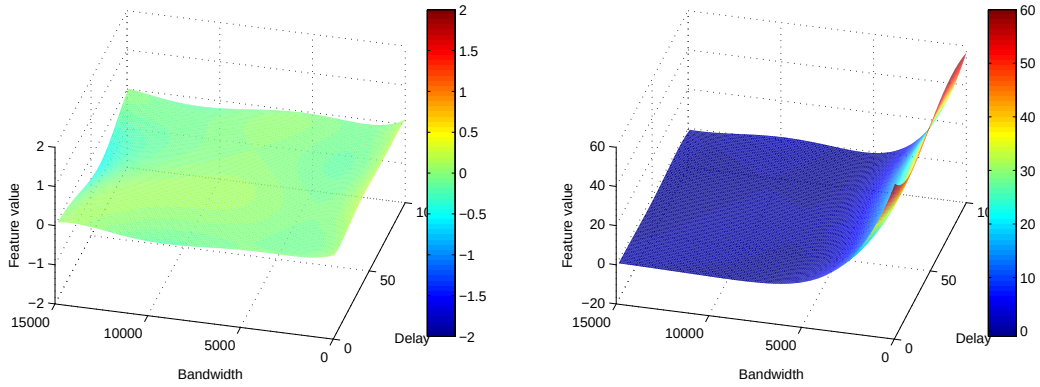


(d) Insufficient write buffer



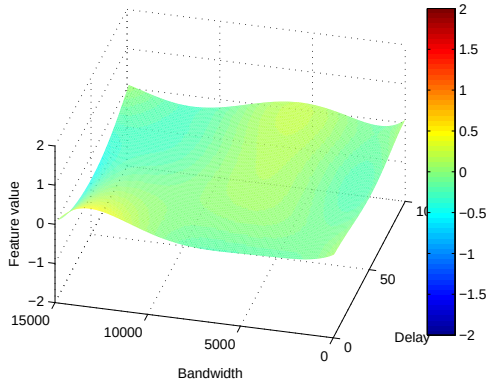
(e) Insufficient read buffer

Figure 3.14: Behaviour of "Total-cumulative-acknowledgements" feature over different links for three UD problems.

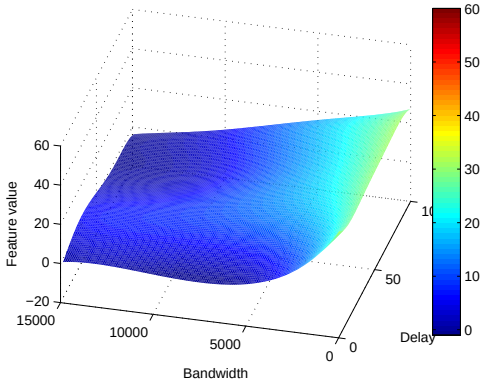


(a) Healthy

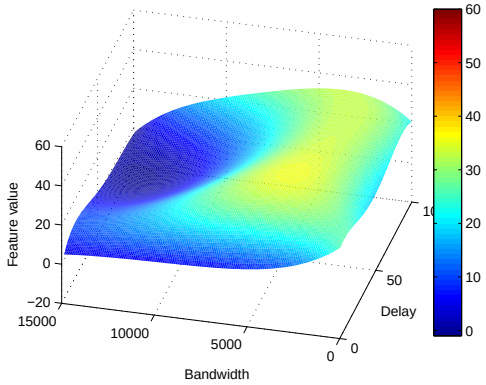
(b) Disabled SACK error



(c) Disabled DSACK error

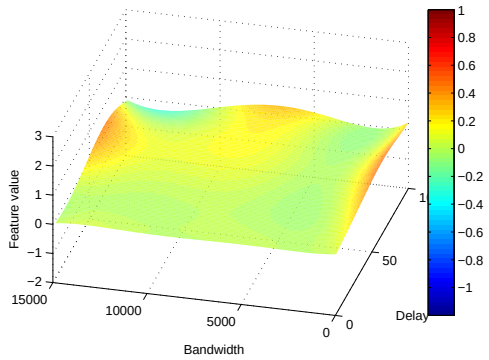


(d) Insufficient write buffer

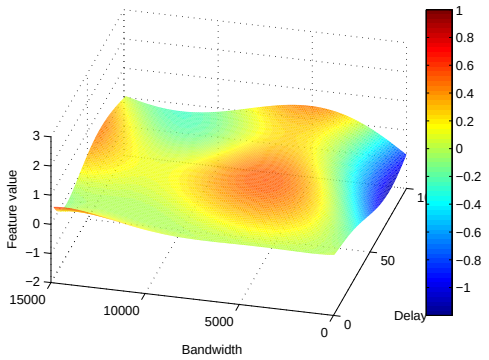


(e) Insufficient read buffer

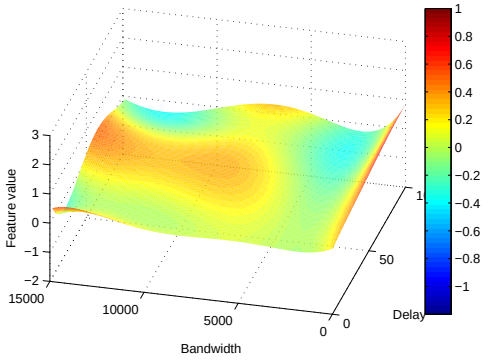
Figure 3.15: Feature behaviour estimation using LASE for "Total number of packets sent"



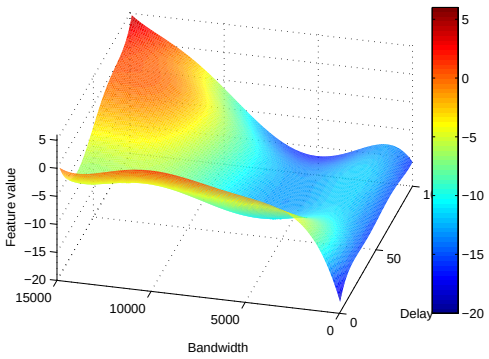
(a) Healthy



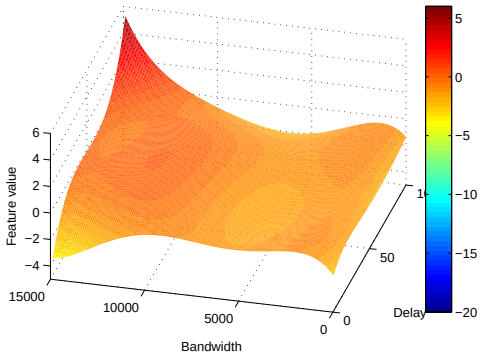
(b) Disabled SACK error



(c) Disabled DSACK error



(d) Insufficient write buffer



(e) Insufficient read buffer

Figure 3.16: Feature behaviour estimation using LASE for "Average segment size" of the connection

binations were collected for the healthy and each of the faulty cases. The first dataset was used for training the FEFs, and the second set was used to evaluate the performance of the FEFs. The training samples were limited to 5 samples per combination of 5 bandwidths, and 5 delay levels ($k_1 = 25$, $n = 125$). Evaluation samples were collected over links with parameter combinations that were

UD fault		Feature (FEF)	
CF0	Healthy	FEF1	$tot.pkts$
CF1	Write buff limit	FEF2	$ack.pkts$
CF2	Read buff limit	FEF3	$RTT.3wh$
CF3	SACK error	FEF4	$cum.ack$
CF4	DSACK error	FEF5	$tri.dupacks$
CF5	Read&write buff limit	FEF6	$avg.segm$
CF6	Disabled timestamps	FEF7	$avg.win.adv$
CF7	Winscale error	FEF8	$stream$
		FEF9	$xmit.time$
		FEF10	$pure.acks$
		FEF11	$rexmt.data.bytes$
		FEF12	$min.retr.time$

Table 3.4: Key

not used in training ($k_2 = 100$, $k_2 \neq k_1$), making a total of 2200 samples per fault for a statistically robust evaluation of FEF performance.

Figure 3.14 shows the estimated FEFs of the "Total-cumulative- acknowledgement" feature for four UD cases. The healthy UD (Figure 3.14a) shows a flat surface, as features are always normalised against the healthy baseline. Figures 3.14b, 3.14c, and 3.14d show the behaviour of the features over the training parameter range with respect to the healthy baseline. Figures 3.15, and 3.16 also show similar FEFs for two other features, "Total number of packets sent" and "Average segment size" respectively. The figures show that the FEFs are different in every scenario, and thus highlight the need to have the LASE mechanism in place to estimate the NSSs to generalise a diagnostic system across dynamic networks.

Table 3.4 provides the notation that will be used in the following discussion. We base the discussion of LASE performance on 12 features (FEF1, $\cdots$, FEF12) of 4 types of faults (CF1, $\cdots$, CF4), although the analysis was not limited to these cases. Tables 3.5, 3.6, and 3.7 summarise the performance of LASE for the three types of regression algorithms used.

From the results, we can clearly identify three main categories of features:

1. As shown in Table 3.5, features such as $tot.pkts$, $ack.pkts$, $RTT.3wh$, $cum.acks$,

		Example feature (R^2)				
Fault	Reg.	FEF1	FEF2	FEF3	FEF4	FEF5
CF1	Poly.	0.9865	0.9865	0.9793	0.9913	0.9877
	SVR	0.9559	0.9559	0.9902	0.8908	0.9783
	NN	0.9890	0.9890	0.9885	0.9959	0.9998
CF2	Poly.	0.9825	0.9825	0.9793	0.9740	0.9877
	SVR	0.9694	0.9694	0.8669	0.8804	0.9779
	NN	0.9980	0.9980	0.9911	0.9949	0.9997
CF3	Poly.	0.9939	0.9939	0.9793	0.9852	0.9846
	SVR	0.9203	0.9203	0.9995	0.9221	0.9631
	NN	0.9877	0.9877	1.0000	0.9821	0.9861
CF4	Poly.	0.9939	0.9939	0.9793	0.9852	0.9846
	SVR	0.9203	0.9203	0.9995	0.9221	0.9631
	NN	0.9877	0.9877	1.0000	0.9821	0.9861

Table 3.5: Estimation performance of LASE measured with R^2 for predictable features

		Example feature (R^2)			
Fault	Reg.	FEF6	FEF7	FEF8	FEF9
CF1	Poly.	0.9948	0.2600	0.9935	0.9835
	SVR	0.9994	0.0041	0.9083	0.8143
	NN	0.9991	0.3947	0.9999	0.9970
CF2	Poly.	0.9948	0.9944	0.9928	0.9829
	SVR	0.9994	0.9695	0.9021	0.8301
	NN	0.9999	0.9870	0.9999	0.9996
CF3	Poly.	0.8934	0.3947	0.5742	0.9875
	SVR	0.8936	0.0019	0.1111	0.8705
	NN	0.8925	0.0296	0.5035	0.9921
CF4	Poly.	0.8921	0.6254	0.8246	0.3653
	SVR	0.8918	0.4366	0.1111	0.4705
	NN	0.8978	0.7506	0.5035	0.5921

Table 3.6: Estimation performance of LASE measured with R^2 for fault-specific predictable features

and $tri.dupacks$ can be predicted with high accuracy using all regression models. ML systems such as SVMs are sufficiently generalised against the $< 2\%$ error of prediction, and the smaller error in most these features does not affect the final outcome.

2. Table 3.6 shows that feature FEFs $xmit.time$, $avg.segm$, $stream$, and $avg.win.adv$

		Example feature (R^2)		
Fault	Reg.	FEF10	FEF11	FEF12
CF1	Poly.	0.3608	0.0963	0.2362
	SVR	0.3172	0.0038	0.0156
	NN	0.2292	0.2111	0.0582
CF2	Poly.	0.3465	0.1355	0.0141
	SVR	0.2536	0.0001	0.0013
	NN	0.1177	0.0436	0.1697
CF3	Poly.	0.1220	0.3485	0.30018
	SVR	0.1302	0.0314	0.0013
	NN	0.0356	0.2903	0.0717
CF4	Poly.	0.1520	0.0567	0.1870
	SVR	0.4502	0.0038	0.0323
	NN	0.2456	0.2351	0.1160

Table 3.7: Estimation performance of LASE measured with R^2 for unpredictable features

	Feature				
Fault	FEF1	FEF2	FEF3	FEF4	FEF5
CF1	93.245	92.456	96.234	90.543	95.625
CF2	89.567	93.256	97.234	90.899	94.563
CF3	90.164	93.975	96.343	91.237	94.089
CF4	92.456	91.238	98.674	92.053	97.541

Table 3.8: Estimation accuracy (in %) of Poly-based FEFs trained with test-bed data for predicting NSSs collected over the live network

are only successful in predicting the features with certain faults (marked in grey). These particular FEFs are used only to train the systems to detect that particular UD fault. This is because some faults can adversely affect particular features to the point that these features become sporadic. From extensive testing we have conducted using large datasets collected from multiple networks, we have identified that these unpredictable features are only dependent on the type of fault. However, this does not guarantee that, in some network environments such as mobile networks, other factors can affect the feature predictability, and some features might join the unpredictable group.

3. Table 3.7 shows that none of the models succeeded in predicting features such

		FEF1		FEF4	
Fault	Reg.	TT	$AFET$	TT	$AFET$
CF1	Poly.	41.4152	0.0031	34.0945	0.0030
	SVR	1452.44	0.1933	1544.88	0.2019
	NN	24.7756	0.3056	29.8138	0.3037
CF2	Poly.	38.7863	0.0032	36.4268	0.0031
	SVR	1408.31	0.2063	1476.53	0.2163
	NN	27.2962	0.2930	25.9615	0.3292
CF3	Poly.	34.4573	0.0029	35.5837	0.0033
	SVR	1659.25	0.2474	1733.46	0.2445
	NN	24.3569	0.2887	36.1969	0.3142
CF4	Poly.	29.9123	0.0029	34.2792	0.0032
	SVR	1579.69	0.2474	1659.87	0.2556
	NN	24.4077	0.2887	25.7183	0.3098

Table 3.9: Train time (TT) of FEF and average feature estimation time ($AFET$) in LASE (in milliseconds)

as $pure.acks$, $rexmt.data.bytes$, and $min.retr.time$. These features only have a loose correlation with link parameters and are considered universally unpredictable.These FEFs are discarded as estimated values from these FEFs can negatively impact a diagnostic system's accuracy.

Tables 3.5and 3.6 suggest that the models created using Poly performed slightly better than NN and SVR. However, NN and SVR are more suitable than the simple Poly technique to create complex models with higher dimensionality when more link parameters are considered. In addition, SVR is more robust when training samples are contaminated with outliers, especially when collected over live networks.

Trained only using test-bed data, FEFs are also capable of accurately estimating features in NSSs collected in the live network. Table 3.8 shows the accuracy of five predicted features in four fault cases when compared to the real network NSSs. These features were chosen as they were determined to be predictable across all four faults, as shown in the Table 3.5. The prediction accuracies in Table 3.8 demonstrate the successful estimation performance of the FEFs. This

capability offers the unique advantage of training the FEFs in a controlled test environment before being deployed in real-world networks.

The time taken for training and estimation is important, considering the large number of regression models involved. We used two parameters; first, train time (TT) to define the time taken to model a single FEF, and second, average feature estimation time (AFET), to define the time it takes to estimate a single feature value. Both these measurements are taken in milliseconds, and the ensuing results are shown in Table 3.9. Since the number of total FEFs is $m \times p$, the time taken to train the complete system is $\sum_{i=1}^{p} \sum_{j=1}^{m} TT(i,j)$, which approximates to 45 s for Poly, 2000 s for SVR, and 34 s for NN given $m = 320$ and $p = 4$. Although FEF training is not a frequent event, the results show that both the Poly and NN models are significantly faster to train than SVR. In contrast, AFETs are crucial as they determine the time needed to produce new NSS data. The table shows that Poly has the fastest and NN has the slowest estimation performance for a single feature. The worst-case average delay with NN is only 390 ms for estimating all NSSs in a $m = 320$ and $p = 4$) scenario.

With only 125 samples per UD fault, LASE extended the NSSs (with filtered features) and consequently, the diagnostic capability over links with a bandwidth range of 100 kB/s-15 MB/s and delay range of 0 ms-100 ms. Even in the worst case, the estimation delays for the whole system remain small. These results demonstrate that with LASE, NSS-based diagnostic systems can be adapted to new access links and links with time-varying properties without sacrificing significant performance.

3.7 Conclusion

We have proposed a technique to uniquely characterise network soft failures in UDs towards the creation of an automated fault diagnostic system. To create the

NSSs, we combined an exhaustive set of aggregated TCP statistical features that are diverse and robust. The TCP streams were collected on-demand and with fixed size transfers to minimise the variables and improve the comparability. The NSSs created for various faults showed the capability to uniquely characterise the faults for an effective identification. Furthermore, the NSSs created over varying link conditions showed the level of impact the link has on feature stability and created sub-sets of significant, null, and unstable features.

Next, we discussed the methodology used for collecting data throughout this research. Raw packet traces have been captured by injecting faults into actual UDs and we presented number of test-beds that we have used to collect data. The data set included 17 different UD scenarios including healthy UDs and faults, 8 different TCP variations at UD and 3 different test-beds. Link emulation was used to precisely control link conditions in the laboratory test-bed, and live networks with increasing complexity were used in the other two test-beds to collect more realistic data with cross-traffic, congestion and variations.

We then introduced LASE to estimate NSSs for a continuous range of link properties by creating FEFs using a small dataset and multivariate regression techniques. The trained FEF models were subsequently used to generate NSSs anywhere within the training range. This approach minimises the number of NSSs needed to train diagnostic systems for link variations. Our evaluations with both test-bed and live network data demonstrate the successful prediction capabilities as well as the acceptable computational delays of LASE. The results also show that not all features can be predicted successfully, and if LASE is performed, those unpredictable features should be discarded from the NSS.

CHAPTER FOUR

FAULT INFERENCE AND CLASSIFIERS

In the context of network diagnosis, the process of analysing a trace (or an NSS created from the trace), identifying the trace behaviour, and interpreting the unique characteristics to determine the root causes is called "inference". In traditional diagnostic practice, the inference process is usually carried out by network professionals to identify the root causes. Instead of a specific set of rules, they are primarily dependent on their experience and the expected baselines to locate artefacts by exhaustively analysing trace data [34, 37]. The human cognitive ability to derive the relationship between previously seen events to a slightly different yet related event is a challenge to replicate. In addition, the subtle dissimilarities in an artefact produced by TCP variants and different UDs also have to be compensated. Section 2.2.1 discussed the traditional practices of diagnosis, especially using TCP traces.

In this chapter, we introduce a new inference method using NSSs and ML-based classifiers to uniquely identify the root causes of network performance problems. The motivation is to create a methodology that is automated, scalable and modular so that it can be utilised in creating complex diagnostic systems.

Embedded artefacts in NSSs

As discussed in Section 2.8, the TCP layer directly observes most network events from the middle of the protocol stack. Network problems have different effects on the packet stream, and the NSSs generated from the TCP packet traces contain anomalies that are unique to the fault. These anomalies are called the "artefacts" and are the basis to separate a healthy UD from a faulty UD. In Chapter 3, we discussed in detail how the NSSs are generated. Figure 3.2 shows some of the NSSs with the artefacts unique to each fault. These unique artefact patterns provide the key to the diagnosis of the specific root causes.

4.1 Pattern classifiers for inferring from NSSs

The proposed inference method uses supervised ML-based pattern classification to identify artefacts unique to different types of faults. More details on supervised machine learning can be found in Section 2.7.1. The labelled NSSs, collected on live networks or generated by emulating network faults in controlled environments, are used to initially train pattern classifiers to create generalised models of each fault. Subsequently, these trained models can be combined to create an end-to-end comprehensive diagnostic system.

The design criteria of the key elements to overcome the challenges presented by an inference problem are as follows:

- Classifier boundaries are well balanced between generalisation and accuracy. Generalisation is essential, as NSSs with the same artefacts can be slightly different, depending on the network conditions, TCP variant, and the UD type.

- Robust against inaccurate training instances and irrelevant attributes because of the random nature of the networks.

- Robust against interdependent attributes because of the interdependency of TCP trace parameters.
- Automated feature selection and parameter selection.
- Easily trainable with a small number of samples.

The learning algorithm in a classifier is a critical choice for the performance of the overall diagnostic system. For this purpose, we first identified candidate algorithms - Decision Trees, Artificial Neural Networks (ANNs), Naïve Bayes (NB), k-Nearest Neighbour (kNN), and Support Vector Machines (SVMs), conducted a comparative analysis, and tested their performance with respect to generalisation, classification accuracy, number of training samples required, and training speed. Based on the observations from these comparisons and the literature review, we chose the SVM approach because of its superiority in yielding the most accurate results. It also exhibits higher tolerance and requires fewer training samples. The comparative performance of the methods is shown in Figure 2.5.

4.2 SVM-based fault classifier module (FCM)

Using SVM as the pattern classification algorithm, we introduce a "fault classifier module" as shown in Figure 4.1. Each fault classifier module is designed to detect artefacts from a single type of fault using L2 soft-margin, non-linear SVM classifiers.

The module operates in two phases: first, the training phase creates an appropriate classifier model using trace data samples collected from known faults. The training phase starts with feature extraction, raw signature generation, and data pre-processing to create the NSS. The process of creating a NSS from a raw packet trace was discussed in detail in Chapter 3. The most relevant features for the particular fault being detected are then selected from the NSS and these fea-

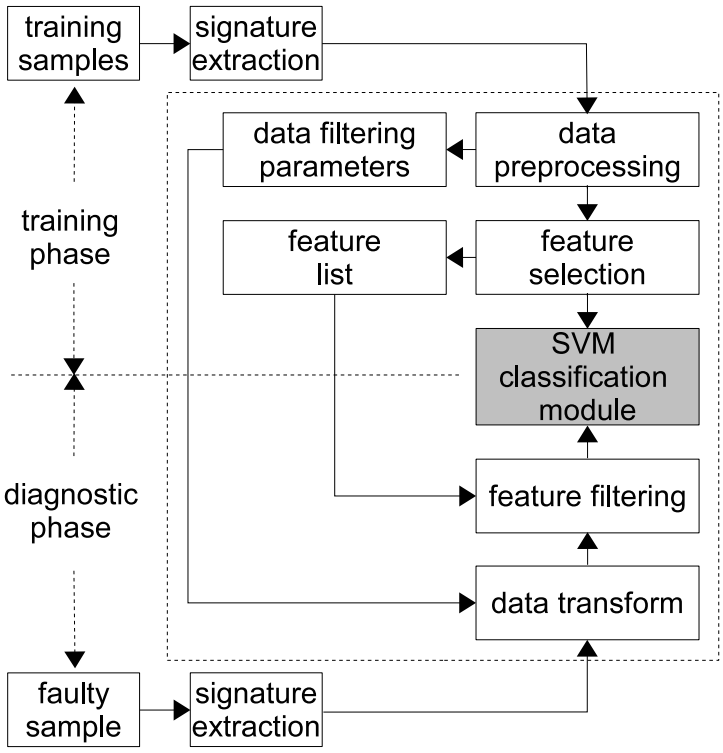


Figure 4.1: SVM-based fault classifier module for artefact identification. Each classifier detects only a single fault with parameters, and a feature list is optimised for the specific problem. This module implements the basic inference method, which is extendible using a combination of modules trained using various data sets to scale the diagnostic system.

tures are used to train the SVM. All these stages are equally vital for building a robust classification model with the best accuracy. During the diagnostic phase, a raw traces sample is sent through NSS generation first, and the trained classifier model is then used to determine the artefacts hidden within the NSS.

4.2.1 Feature selection

Although the NSS format is identical for every sample, only a particular subset of features contributes to the artefact. Unnecessary features increase the computational complexity [240], create over-fitting of classifier boundaries [241], and reduce classification accuracy [242]. Therefore, insignificant features should be removed from the training data. In this study, we use an automated feature selection method to select the most suitable feature subset for a particular classifier. In

addition, if the LASE technique has been utilised in creating the training data, the unpredictable and unstable features must be removed from the NSS as they are not reliable.

There are two main categories of feature selection algorithms: (i) filters which use statistical characteristics of features, and (ii) wrappers which cross-validate all variations of feature subsets to select the best set. Wrappers are considered to perform better than filters, but are computationally expensive. Our proposed method, as shown in Figure 4.2, similar to work discussed by Xing et al. [243] and Das [244], follows a hybrid approach (between filter and wrapper) for isolating the best feature subsets. We first use a filter technique, Student's t-test (two-sample sample t-test) (implemented similar to [245]), to assess the significance of every feature for separating the two classes. Next, the features sorted in order of significance are cross-validated by incrementing the number of features selected for each class (wrapper technique) against test data to identify the best number of features required for each classifier. Student's t-test is a common statistical data analysis procedure for hypothesis testing, and determines whether two independent populations have different mean values for a specified extent. The feature selection process reduces the q-dimensional feature vector in Eq. (4.1) to m-dimensions ($m < q$), where the combination of m features creates the artefact.

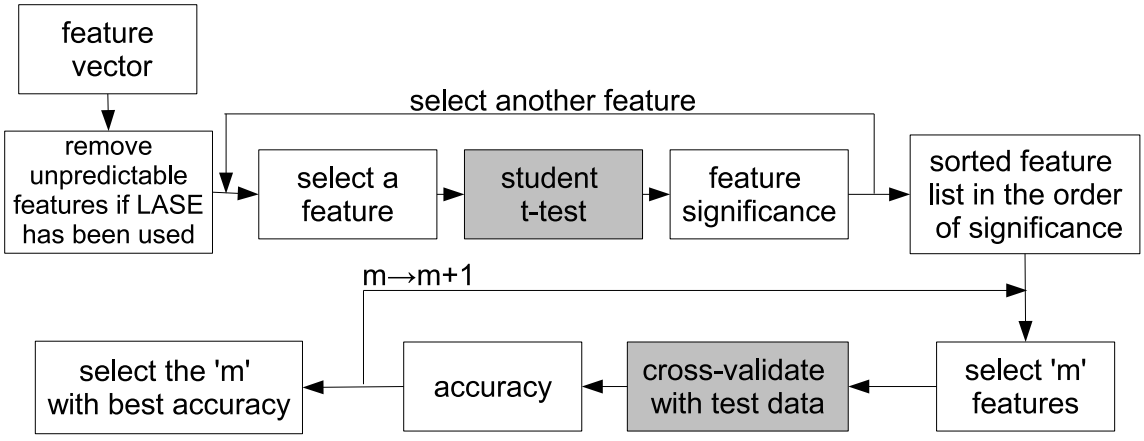


Figure 4.2: Hybrid feature selection technique for isolating the best feature subset of the artefact.

This process creates a new signature database of classifier-ready NSSs (Θ).

4.2.2 SVM classification

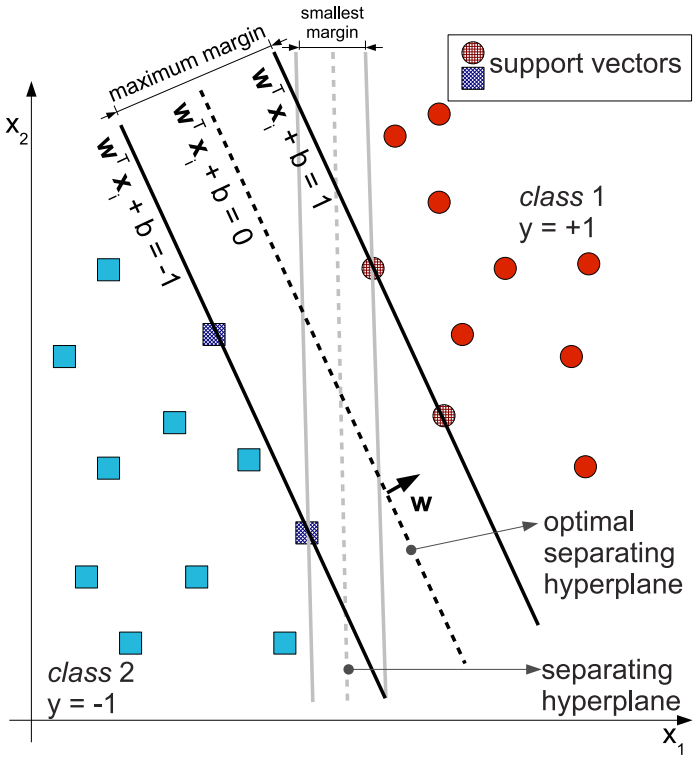
An SVM classifier module is used to model the best separating boundary between the "faulty" class and the "other" class. Here, the other class includes healthy NSSs and other known fault NSSs except the specific NSSs, from the fault being detected. n trace samples in Θ are used for training the SVM with the class label *+1* for the faulty class and *-1* for the other class. For an m-dimensional input feature vector, the resultant class boundary is a m-dimensional hyper-surface that separates the two classes with the maximum margin.

The artefacts of the same class show subtle variations due to the practical data collection process. Different TCP variants and device platforms also deviate from some of the artefact parameters with respect to the training samples. We use L2 soft-margin SVMs (Eqs. (4.3) and (4.4)) instead of hard-margin SVMs (see Chapter 2.7.2 for details) in each classifier module to create generalised class boundaries that offer better classification accuracy against deviated artefacts (see Figure 4.3b). The separating class boundaries can either be linear or non-linear, depending on the artefact type. We use kernel mapping (Eq. (4.6)) with kernel functions (Eq. (4.8)) chosen based on the classification problem of each classifier module. The SVMs are trained by solving the QP problem (Eq. (4.5)) and the resulting class boundaries and the models are stored in each classifier module.

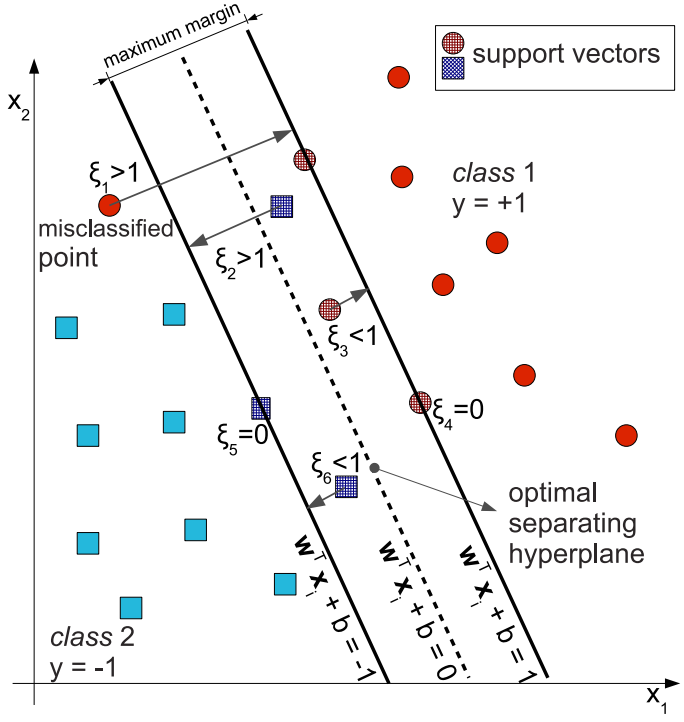
4.2.3 Training the fault classifier

The binary-SVM is trained to classify data into two classes: faulty and other. Training requires two sets of trace samples, each from the two classes.

After NSS generation and feature selection, the training data set for the partic-



(a) Hard-margin SVMs



(b) Soft-margin SVMs

Figure 4.3: Comparison of hard-margin SVMs for classification of linearly separable classes and soft-margin SVMs for classifying overlapping classes.

ular FCM Θ of *n* instances is in the form of

$$(\Theta) = \{(\mathbf{x}_i, y_i)|\mathbf{x}_i \epsilon \Re^m, y_i \epsilon \{+1, -1\}\}_{i=1}^n \tag{4.1}$$

with $\mathbf{x}_i$ being an m-dimensional feature vector and class label y_i, where $y_i = +1$ for the faulty class and $y_i = -1$ for the other class, to which each $\mathbf{x}_i$ belongs. For example, a sample trace ($i = 1$) from a faulty class, with four features ($m = 4$) is denoted by $\{0.5, 0.03, 0, 0.99, +1\}_{i=1}$ (e.g. 1). For L2 soft-margin and kernel-mapped SVMs, an m-dimensional input feature vector results in an m-dimensional hyper-surface class boundary. The hyper-surface separates the two classes with the maximum margin.

For the data set given in Eq. (4.1), a linear decision function

$$D(\mathrm{X}) = \mathbf{w}^{\mathrm{T}}\mathrm{X}_i + b \text{ for } i = 1, ..., n, \tag{4.2}$$

where, **w** is the *m*-dimensional weight vector exist and the optimum hyperplane is found by minimising

$$Q(\mathbf{w}, b, \xi) = \frac{1}{2}\mathbf{w}^{\mathrm{T}}\mathbf{w} + \frac{C}{2}\sum_{i=1}^{n}\xi_i^p \tag{4.3}$$

with respect to **w**,*b* and ξ, subject to the inequality constraints:

$$y_i(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i + b) \geq 1 - \xi_i \text{ for } i = 1, ..., n \text{ and } \xi_i \geq 0 \tag{4.4}$$

where, non-negative ξ is called the slack variable and allows a degree of inseparability between the two classes and $p = 2$ for L2 soft-margin SVM. This forms a convex quadratic programming problem (QP) [246] and is solved after converting the constrained problem given by Eqs. (4.3) and (4.4) into an unconstrained

problem:

$$L(\mathbf{w}, b, \alpha, \xi) = \frac{1}{2}\mathbf{w}^{\mathrm{T}}\mathbf{w} + \frac{C}{2}\sum_{i=1}^{n}\xi^2 - \sum_{i=1}^{n}\alpha_i(y_i(\mathbf{w}^{\mathrm{T}}\mathrm{x}_i + b) - 1) + \xi_i), \tag{4.5}$$

where $\alpha_i(\geq 0)$ are the Lagrange multipliers introduced to enforce the positivity of the ξ_i. The optimal saddle point ($\mathbf{w}_0, b_0, \alpha_0, \xi_0$) is found where L is minimised with respect to $\mathbf{w}, b$ and ξ_i and maximised with respect $\alpha_i(\geq 0)$ following Karush-Kuhn-Tucker (KKT) [247] conditions. This is called the training of SVM.

However, to enhance the separability of the linearly inseparable data, using the non-linear vector function $g(\mathbf{x}) = (g_1(\mathbf{x}, ..., g_l(\mathbf{x})^{\mathrm{T}}$, the original *m*-dimensional input vector **x** is mapped into the *l*-dimensional feature space. The linear decision function for the obtained *l*-dimensional feature space is given by Eq. (4.6).

$$D(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}g(\mathbf{x}) + b \tag{4.6}$$

where, **w** is the *l*-dimensional weight vector, and b is the bias term. According to the Hilbert-Schmidt theory, the mapping function $g(\mathbf{x})$ that maps **x** into the dot-product feature space satisfies

$$K(\mathbf{x}, \mathbf{x}_i) = g(\mathbf{x})^{\mathrm{T}}g(\mathbf{x}_i) \tag{4.7}$$

where $K(\mathbf{x}, \mathbf{x}_i)$ is called the kernel function. The kernel function avoids the actual mapping $g(\mathbf{x})$ and directly calculates the scalar products $g(\mathbf{x})^{\mathrm{T}}g(\mathbf{x})$ in the input space. We cross-validated and analysed the performance of the classifier for multiple kernel functions $K(\mathbf{x}, \mathbf{x}_i)$ in Eq. (4.8) to select the most suitable kernel for the classification problem.

$$K(\mathbf{x}, \mathbf{x}_i) = (\mathbf{x}^{\mathrm{T}} \cdot \mathbf{x}_i) \qquad \text{Linear kernel} \tag{4.8a}$$

$$K(\mathbf{x}, \mathbf{x}_i) = (\mathbf{x}^{\mathrm{T}} \cdot \mathbf{x}_i + 1)^2 \qquad \text{Quadratic kernel} \tag{4.8b}$$

$$K(\mathbf{x}, \mathbf{x}_i) = (\mathbf{x}^{\mathrm{T}} \cdot \mathbf{x}_i + 1)^3 \qquad 3^{\mathrm{rd}}\,\text{degree polynomial kernel} \tag{4.8c}$$

$$K(\mathbf{x}, \mathbf{x}_i) = \exp(-\|\mathbf{x} - \mathbf{x}_i\|^2)/2\sigma^2 \qquad \text{Gaussian RBF kernel} \tag{4.8d}$$

4.2.4 Diagnosing faults

At the end of the training phase, a classifier module contains (i) filtering parameters (scale factors) derived during NSS normalisation, (ii) a feature list chosen during feature selection, and (iii) a classifier model created using SVMs. During the diagnostic phase, a trained module is used to detect if the particular fault exists in a trace taken from an undiagnosed source. Packet traces are captured using the same trace collection technique as before, and then undergo signature extraction, data transformation to create the NSS, and feature filtering using the stored values. The SVM classifier then classifies the NSS to the corresponding class and outputs a numerical $+1$ if the sample contains the specific fault the FCM was trained for or -1 if it does not. The class labels are then assigned to the trace accordingly. Each module is responsible for detecting a single fault and will only indicate the presence or absence of that particular fault.

4.3 Advantages of the proposed method

The proposed inference method with fault classifier modules offers many advantages over other methods proposed in the literature, as follows:

(i) SVM binary classifiers offer extremely accurate classification compared to other single multi-class or heuristic-based classifiers.

(ii) Classifiers can be quickly trained with only a few data samples.

(iii) Diagnostic capability evolves with the diversity of the fault signature databases instead of the inference method. This is in contrast to most other proposed diagnostic solutions, where the inference methods can only identify a specific type or group of issues.

(iv) Users can collaborate to create common signature repositories and new diagnostic modules, encompassing a wide range of faults, networks and device platforms.

(v) The method relies solely on packet traces collected at the endpoints and is implemented as an client-server application. This provides flexibility for the operator to easily deploy diagnostic systems at any desired network location without significant modification to servers or access routers.

(vi) Using this method, UDs and systems can be diagnosed without remote access or physically logging on to the systems. This is a capability unavailable in many network diagnostic tools [33, 36].

(vii) The proposed technique avoids the idiosyncrasies of individual TCP implementation and relies only on connection statistics to detect the artefacts. This enables us to create a diagnostic technique, which is in most cases TCP agnostic.

(viii) The inference technique considers overall trace behaviour rather than TCP negotiation headers to avoid misleading header and flag information [51, 211].

(ix) Modules trained for different types of faults can be combined to create complex diagnostic systems targeted to specific networks and applications.

4.4 Performance evaluation of FCM

The classification performance of the individual FCMs was evaluated using the test-beds introduced in Section 3.3. We created two datasets, one from test-bed 1, which was a laboratory network with emulated links (TB-1 data set) and another from test-bed 2, which used the live campus network (TB-2 data set). The TB-1 dataset was collected over an emulated, full duplex, wired access link with 80 Mb/s bandwidth, 10 ms delay with no packet loss and no packet reordering. The TB-2 dataset was collected over a number of days around the clock, to include network peak traffic periods and low traffic periods. Table 4.1 shows the list of 15 scenarios we created at the UD. We collected data from 14 faulty conditions (CF1-14) and a healthy UD (CF0). For both TB-1 and TB-2 datasets, we collected data using the 8 TCP variants listed in Table 3.2 as they are used in many common devices. We collected 100 trace samples for each unique fault-TCP combination, and the final dataset included $12,000$ traces from TB-1 and $12,000$ traces from TB-2.

First, the individual FCMs were trained to detect a single CFx condition us-

Fault	Description
CF0	Healthy
CF1	Disabled SACK error
CF2	Insufficient write buffer
CF3	Insufficient read buffer
CF4	Disabled D-SACK error
CF5	TCP timestamps are not working/in error
CF6	Window scaling error
CF7	Limited reordering threshold
CF8	Link-UD speed mismatch
CF9	Link-UD speed mismatch & duplex mismatch
CF10	UD firewall causing packet loss
CF11	UD firewall causing packet delay
CF12	Overloaded UD CPU
CF13	Overloaded UD memory
CF14	UD HDD i/o overloaded - faulty

Table 4.1: Key: List of faults

ing the TB-1 data set. We used data from all the TCP variants, i.e. 800 samples per fault, during both training and testing. As there were sufficient data, we conducted 4-fold, 5-fold and 8-fold cross-validation to obtain a statistically robust result. In k-fold cross-validation, the original dataset was randomly partitioned into k equal size sub-sets. Of the k sub-sets, a k-1 sub-set was retained as the validation data for testing the model, and the remaining set was used as training data. The training dataset included 200, 160, and 100 samples respectively. The cross-validation process was then repeated k times (the folds), with each of the k sub-sets used exactly once as the training data. The k results from the folds were then averaged (or otherwise combined) to produce a single estimation. The advantage of this method over repeated random sub-sampling is that all observations were used for both training and validation, and each observation was used for training exactly once. We used the four types of SVM kernels given in Eq. (4.8) to evaluate how each kernel performs against the NSSs.

Figures 4.4 and 4.5 show the variation of detection accuracies against the number of selected feature sets for two faults, insufficient read buffer (CF3) at the UD, and link-UD speed mismatch & duplex mismatch (CF9) respectively. Both figures show a similar pattern, with the detection accuracy peaking around 34 features for CF9 and 22 features for CF3. The selection of the kernel has a significant impact on the overall system performance when the FCMs are used in a full diagnostic system. It is clear from the graphs that the Gaussian RBF kernel consistently outperformed the other kernels, quadratic kernel being the second best, then the linear kernel and the 3rd degree polynomial kernel showed the worst performance. CF9 FCM achieved an accuracy of 98.21% with a 34 feature sub-set, and CF3 FCM achieved an accuracy of 93.37% with a 22 feature sub-set. A key advantage of the FCM concept is the ability to select the optimal set of parameters for the specific detection task. FCMs independently select the optimum feature sub-set and the best kernel for the specific detection task to achieve the

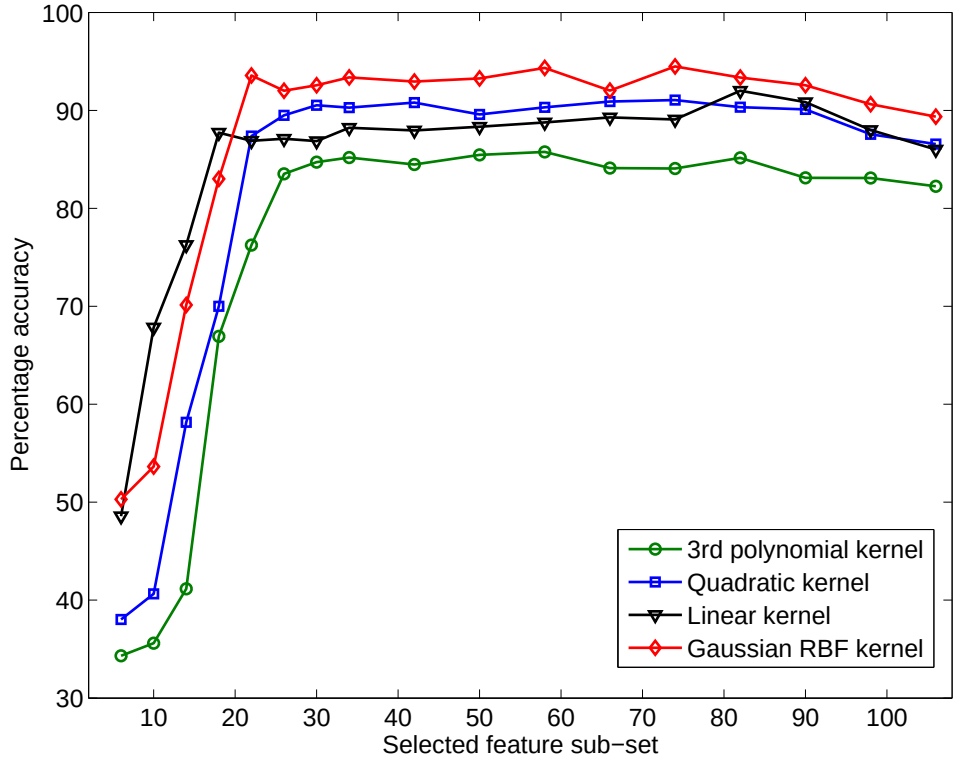


Figure 4.4: Classification accuracy variation of CF3 FCM with feature sets for different SVM kernels.

best possible detection accuracy. If the inference method used a universal set of parameters similar to most systems reported in the literature, we would only be able to achieve a 91.86% accuracy for the CF9 FCM (assuming we set the feature sub-set to 22 features throughout). In summary, with the selection of optimum FCM parameters, the modules were capable of detecting the fault with a high degree of accuracy. Both Figures 4.4 and 4.5 show the FCM detection accuracy only marginally increases after the initial peak and then starts to decrease as the feature sub-set increases. Given that additional features add complexity to the model, this marginal increase in performance can be offset by the loss of classifier generalisation capability, especially when deployed in dynamic network environments. This is visible in the results from the decreasing accuracy due to the over-fitting of the classifier boundary to tightly fit the training data.

The Table 4.2 shows the peak detection accuracies achieved for all 15 FCMs and their respective parameters. The results show that the FCMs are capable

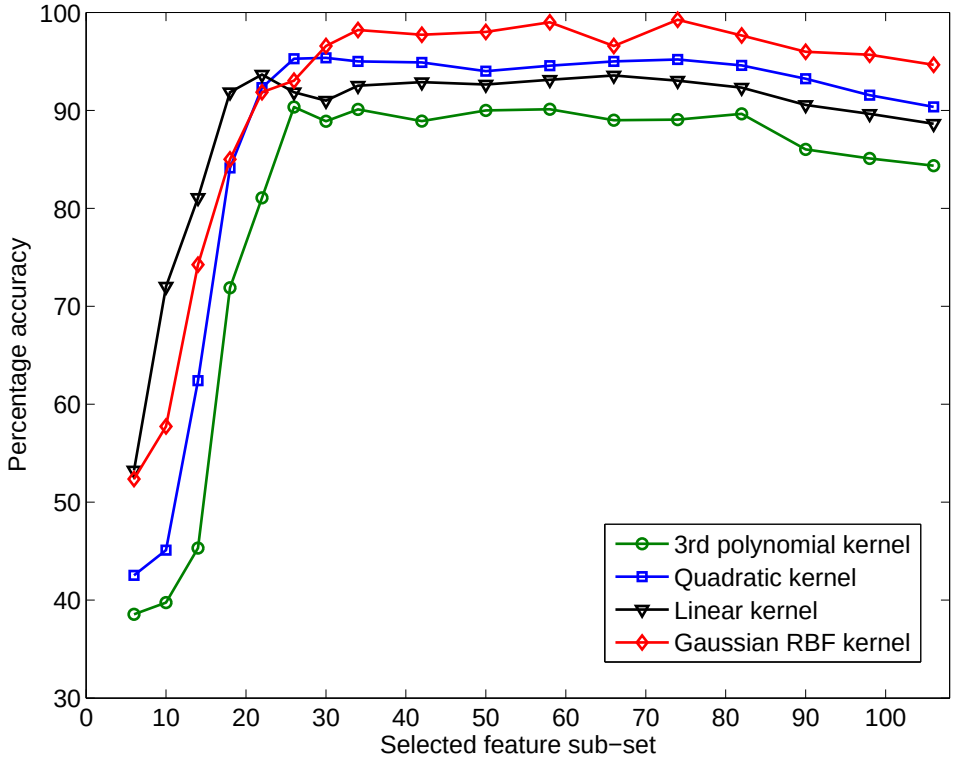


Figure 4.5: Classification accuracy variation of CF9 FCM with feature sets for different SVM kernels.

of detecting the faults embedded in the NSSs across all the 14 faults with high accuracy and also successfully detected the healthy UDs with 97.87% accuracy. The lowest detection accuracy was observed with CF4, D-SACK error, which was a result of the NSS being very close to SACK error NSS. However, even for the most complex case, we achieved an accuracy of 90%. The required feature subset was different between the FCMs as some complex NSSs required additional features to discern the fault from the rest of the NSSs. In most cases, the Gaussian RBF kernel offered the best performance, except in CF1 and CF2, where the linear kernel offered the best performance.

Following the same training criteria, we trained another set of FCMs with data collected from TB-2, the live network. The live network NSS features showed a larger variation due to the dynamic nature of the traffic, especially due to the highly congested links during peak hours. This dataset was used to demonstrate the robustness of FCMs in live networks with cross traffic and congestion. The

Fault	TB-1 lab network	Optimum features	TB-2 live network	Optimum features
CF0	97.87	12	95.74	10
CF1	98.25	34	95.46	34
CF2	98.46	26	96.01	22
CF3	93.56	22	91.77	22
CF4	90.32	66	88.24	50
CF5	96.53	42	94.2	34
CF6	94.3	50	92.33	58
CF7	99.02	18	97.11	22
CF8	95.73	22	93.48	30
CF9	96.58	18	93.7	42
CF10	98.21	34	95.72	34
CF11	97.32	30	95.79	50
CF12	93.87	74	91.93	90
CF13	97.58	58	95	74
CF14	95.56	42	92.89	50

Table 4.2: FCM classification accuracy and optimum feature sets for mixed TCP data set collected from test-bed 1 and test-bed 2.

statistical and holistic nature of the NSS and the SVM classifier generalisation successfully offset the effects of the dynamic nature of the network. Table 4.2 shows that all the UD conditions were successfully identified by each FCM with a high degree of accuracy, ranging from 88.24% – 97.11%. The detection accuracies were slightly lower than those for the laboratory network, due to the stable, more controlled conditions in the laboratory network.

The evaluation to this point used a dataset that had samples from all TCP types. However, in the worst case, training data might not be available from a number of TCP types, specially if the training data is collected on a more restricted network. To evaluate the worst-case performance of the FCMs, we trained two sets of FCMs with TB-1 and TB-2 datasets, but used only a single type of TCP for the training dataset. As TCP New Reno is the most commonly used TCP in the world, we used NSSs with New Reno at the UD for training and used all the other NSSs with 7 different TCP types for testing the performance. Table 4.3 shows the overall accuracies achieved for detecting each case for both datasets. The results

Fault	TB-1 lab network, mixed TCP training set	TB-1 lab network, single TCP training set	TB-2 live network, mixed TCP training set	TB-2 live network, single TCP training set
CF0	97.87	95.63	95.74	94.5
CF1	98.25	96.11	95.46	94.32
CF2	98.46	96.29	96.01	94.84
CF3	93.56	90.6	91.77	89.81
CF4	90.32	87.49	88.24	86.41
CF5	96.53	93.91	94.2	92.58
CF6	94.3	92.13	92.33	91.16
CF7	99.02	96.11	97.11	95.2
CF8	95.73	93.31	93.48	92.06
CF9	96.58	94.03	93.7	92.15
CF10	98.21	95.85	95.72	94.36
CF11	97.32	94.43	95.79	93.9
CF12	93.87	90.88	91.93	89.94
CF13	97.58	95.08	95	93.5
CF14	95.56	93.53	92.89	91.86

Table 4.3: Comparison of FCM classification accuracy when trained with mixed TCP data set vs TCP New Reno, single TCP data set and tested with mixed TCP data set.

show that, whilst the accuracies are slightly lower than those for the mixed TCP training datasets, FCMs are still capable of detecting the faults with a high degree of certainty. The figures from the laboratory network show an accuracy variation between 87% – 96% and those from the live network show an accuracy variation between 86% – 95%. Given that this was a worst-case scenario, the FCMs are still capable of achieving acceptable accuracies for a diagnostic system.

4.5 Conclusion

In this chapter, we proposed an inference method using machine learning-based pattern classification to identify artefacts unique to different types of faults. Using L2-soft margin SVM, we created fault classifier modules, which were used to identify a single fault. The FCM was trained using NSSs to create a class bound-

ary that is unique to the particular fault and capable of automatically detecting similar NSSs when sent through the classifier at the diagnostic stage. Since NSSs contain many features that do not provide any information to separate the two specific classes, we introduced a hybrid feature selection technique using filters and wrappers to reduce the signature feature set.

During the performance evaluation, we used data from 14 different faults and healthy UDs collected from two test-beds to demonstrate the diagnostic accuracy of the individual FCM modules. We demonstrated the effects of feature selection and the impact various TCP types can have on the FCMs. The results show that FCMs can be used to reliably identify individual faults through artefacts embedded in the NSSs. These FCMs provide the basic building blocks necessary to build more complex and comprehensive diagnostic solutions.

CHAPTER FIVE

AUTOMATED INFERENCE SYSTEM FOR DIAGNOSING SOFT-FAILURES IN USER DEVICES

In the previous chapter, we introduced the design of a fault classifier module (FCM) that performed automated fault inference using NSSs and supervised ML. A single module has been designed to detect a single fault and can be combined to diagnose multiple failures. In this chapter, we introduce IAND-*k*, an Intelligent Automated Network Diagnostic system for known problem diagnosis with the capability to significantly reduce the time between problem reporting and resolution for a network service provider or administrator. The system we propose (i) uses FCMs for root cause diagnosis, (ii) detects faulty links, and (iii) specifically focuses on identifying root causes of the UD faults.

5.0.1 IAND-*k* system

An overview of the system is shown in Figure 5.1. Whilst the comprehensive IAND-*k* system includes both link problem diagnosis and UD fault diagnosis, the scope of the present research is limited to UD fault diagnosis. The system which is the focus of this discussion is marked in grey in Figure 5.1 and for clarity, will be called IAND-*k*UD.

Most network diagnostic systems today follow a bottom-up design approach,

where a particular algorithm is used to look for a specific anomaly in the packet trace. In contrast, the IAND-*k* system follows a top-down design approach, where a common inference method using FCMs has been utilised across the entire system, but each FCM is specifically trained to diagnose an individual fault. This design offers a number of unique advantages as follows:

(i) The system is more manageable, since a common core inference method is utilised across the whole system.

(ii) Classifier parameters are chosen independently for each module, depending on the complexity of the artefact and the classification problem. This approach offers better accuracy compared to a single set of classifier parameters catering to all types of faults. As the classifier has been parametrised, each module can be fine-tuned to optimise its sensitivity to a particular fault.

(iii) The inference method can be upgraded across the whole system and the system can be easily re-trained due to its modular design.

(iv) The system can be extended by adding new modules instead of changing the core inference algorithms in the entire system.

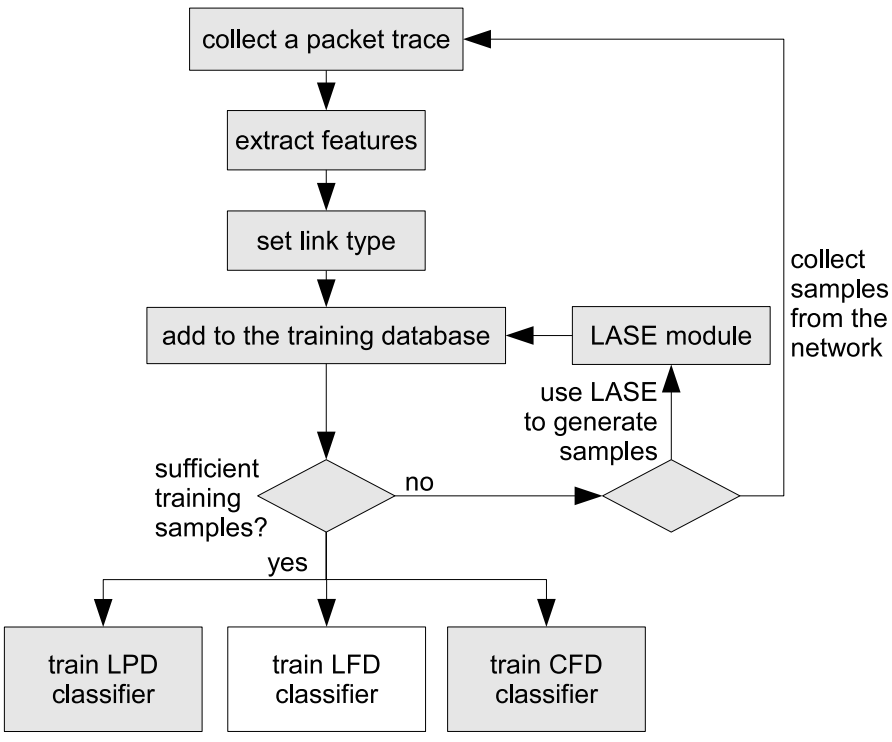
(v) The system's scalability can be automated by simply replicating an inference module and training with new data.

(vi) Each module can be more easily re-trained compared to a complete system in case of a training error.

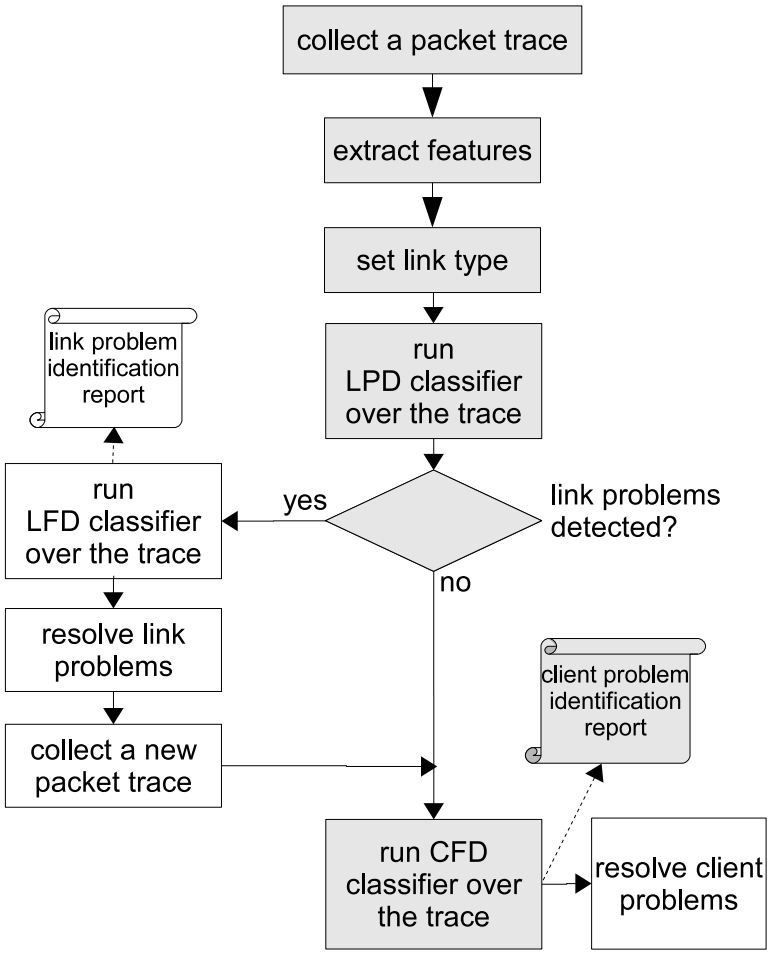
The IAND-*k* system can be adopted to work with most types of access networks and enterprise networks to detect a wide range of UD faults by changing the training dataset.

5.0.2 Operational overview

The operational overview is shown in Figure 5.1. The IAND-*k* system contains three major cascading classifiers. Given that the access router is optimised for the best possible connection, performance problems experienced by the end user can either be a result of an access link/path problem or a UD problem. First, the system has to determine whether the link connecting the UD to the access router is faulty and is the cause of the performance problem experienced by the user. The *link problem detection (LPD) classifier* simply reports whether a link is operating as expected. If the link is found to be faulty, the problem needs to be identified, and the root cause needs to be fixed. Then the *link fault detection (LFD) classifier* is responsible for identifying the exact root cause of the link problem. The LFD classifier design is beyond the scope of the present research, and we have left the automatic diagnosis of the access link problem to a future study. When the link faults have been diagnosed and resolved, the *client fault diagnostic (CFD)* classifier is run to identify the root causes of performance problems in the UD.



(a) IAND-*k* training phase



(b) IAND-*k* diagnosis phase

Figure 5.1: Operational overview of the IAND-*k* system: Link problem detection (LPD) classifier identifies packet traces collected over faulty links. The link fault diagnostic (LFD) classifier and client fault diagnostic (CFD) classifier diagnose the exact root cause of the connection problems. The grey components are the main focus of the proof-of-concept user device diagnostic system (IAND-*k*UD), as link/path problem diagnosis is beyond the scope of the present research.

The IAND-*k* system has two stages of operation: (i) the training phase (see Figure 5.1a), and (ii) the diagnosis phase (see Figure 5.1b). During the training phase, labelled data samples are used to train the system classifiers to detect various known faults in the particular network. Depending on the link type and the data availability, the LASE module can be engaged to generate additional training data. Then all the classifiers are trained by training individual FCMs and setting the appropriate parameters such as the selected feature set and classifier bounds.

During the diagnosis phase, users experiencing performance issues engage

the system through a web interface. The tool first loads the capturing modules into the user device and collects a TCP packet trace of a stream of data between the user device and the access router. Details of the trace collection mechanism have been discussed in Section 3.2. The captured packet trace is then converted to an NSS. This NSS is subsequently sent through the classifiers, undergoing feature filtering and classification. The output at each classification stage is a diagnostic report which outlines the possible issues. This system focuses only on the diagnosis, while the actual fix for the issue and the resolution process are outside the scope of this system.

Deployment

The IAND-*k* system is ideally suited to diagnose a UD directly connected to the access router through the last-mile access link for best diagnostic accuracy. A simple connection between the access router and UD results in clearer NSSs and more consistent diagnosis. NSSs have a higher noise component as the number of hops and bandwidth-delay products increase. The system uses the packet traces of a TCP stream that runs on an on-demand basis between the user device and the service provider router, as shown in Figure 5.2. The system deploys trace collection/capture modules at the access router and once engaged, on the user's device. The diagnostic system can be co-located with the access router, but can be located in a dedicated centralised location if servicing a number of access routers. The collected traces are then sent to the IAND-*k* server to perform the diagnosis.

5.0.3 Link problem detection (LPD) classifier

The LPD classifier determines if the connection problems experienced by the user are caused by a faulty access link. It detects the artefact patterns which exist only when a link performance degrades from the expected baseline. The link performance problems may be caused by many phenomena (including unexpected

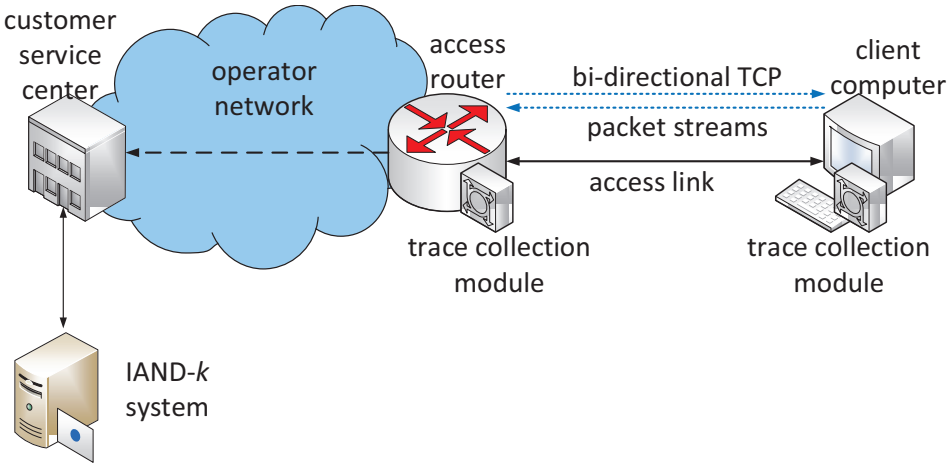


Figure 5.2: Deployment of the IAND-*k* system in a network operator environment.

packet losses and excessive delays). We define an access link performing at the expected baseline as a *healthy access link*, whereas a link with degraded performance is a *faulty access link*. The performance expectation of a healthy link is network-specific. The operator has the freedom to train several LPD classifier modules following the same design criteria, each designed to detect a specific type of faulty access link with a predetermined baseline performance (e.g. 24 Mb/s or 12 Mb/s DSL link, 14 Mb/s HSDPA link, 54 Mb/s 802.11g link with 1% packet loss or 5% packet loss). This gives the operator the flexibility to easily select the most suitable LPD classifier for the user's access network during the diagnostic phase.

The problem presented to the LPD classifier conforms to a binary classification problem with two outcomes, either a faulty or a healthy link. The LPD classifier design uses a single fault classifier module with an LPD-SVM classifier (see Figure 5.3) for classifying the faulty link artefacts. During training, a trace signature database (Θ_{lpd}) in the form of Eq. (4.1) with faulty link traces in the FAULTY class and healthy link traces in the HEALTHY class is used. The LPD classifier goes through the training process described in Section 4.2. The LPD-SVM finally creates a classifier model with a hyper-surface separating the link artefacts.

The LPD classifier is capable of detecting faulty links, even if both link and UD

faults simultaneously cause connection problems. However, this task is challenging, because artefacts created by UD faults either (i) mask those artefacts from faulty links, or (ii) create false positives as link problems. To create a robust LPD classifier model, the training data should contain traces collected with faulty as well as healthy UDs for both the faulty and healthy link classes. The feature selection algorithm then identifies the unique features that indicate a faulty link. With cross-validation-based selection of parameters, SVM creates a complex class boundary separating the classes. The soft margins of the classifier boundary compensate for any deviations of the artefacts caused by UD behaviour, TCP type and slight network variations.

The use of a mechanism to detect faulty links before diagnosing a UD ensures that traces sent through the CFD classifier do not contain any faulty link artefacts. This simplifies the design of the CFD classifier and improves classification accuracy.

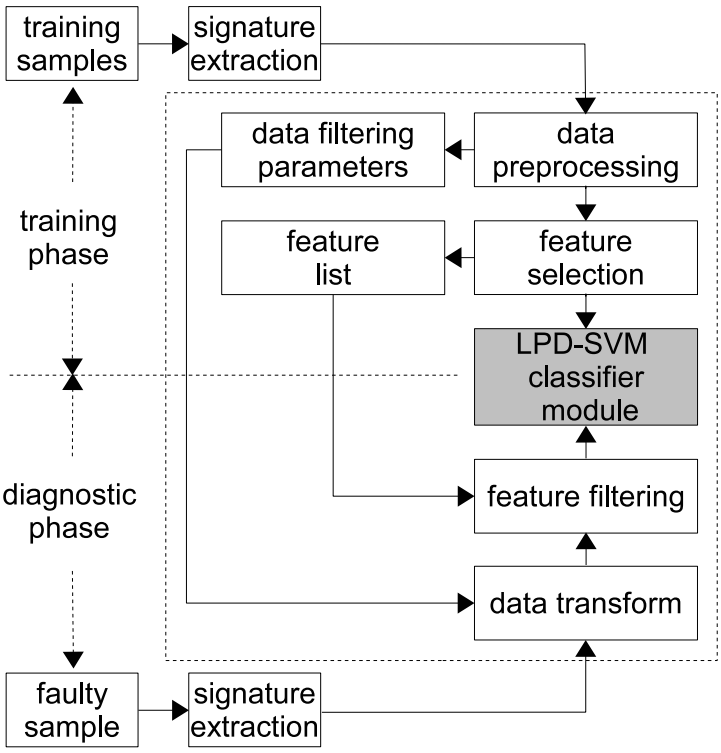


Figure 5.3: LPD classifier design using a single FCM.

5.0.4 Client fault diagnostic (CFD) classifier

The first stage of the IAND-*k* system ensures that the access link is not causing the connection problem. The second stage, the client fault diagnostic (CFD) classifier, identifies the specific types of UD faults, if any, causing the performance problem.

The major emphasis in designing the IAND-*k* system lies in the CFD classifier stage, which automates the UD failure diagnosis. The design of a classifier to separate and identify the artefacts from different UD problems is a challenge for a number of reasons: (i) the diverse range of possible problems, (ii) the subtlety in variations between their artefacts, (iii) variations of TCP types between devices, (iv) multiple simultaneous problems. We propose two CFD classifier designs: CFD-P, which uses a parallel set of FCM modules, and CFD-M, which uses a matrix of FCM modules.

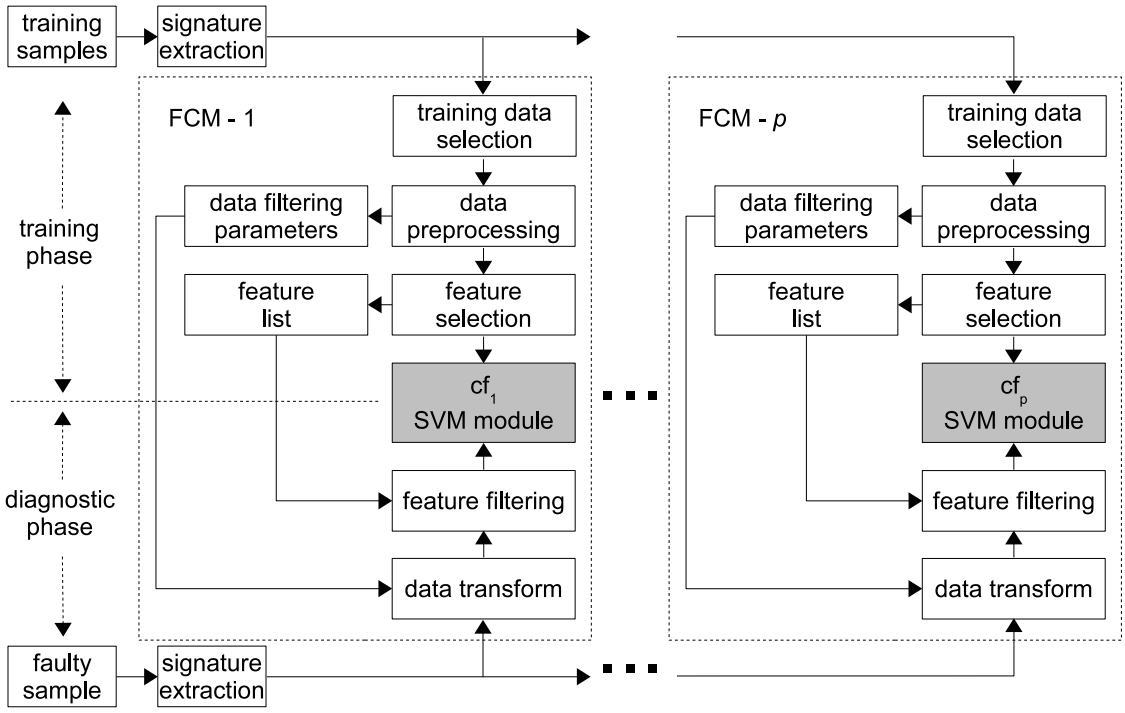


Figure 5.4: CFD-P classifier design for the IAND-*k* system.

CFD-P classifier design

The CFD-P classifier design uses a parallel network of fault classifier modules (FCMs), each trained to diagnose a single fault, to collectively perform a multi-class classification, as shown in Figure 5.4. A device experiencing connection problems is defined as a *faulty UD* and a device that is not is defined as a *healthy UD*. In CFD-P design, each FCM is trained with two classes, one class being NSSs from a specific faulty UD (faulty class) and the other being all the other NSSs, including healthy and other known faults (other class). The modular design offers the flexibility to continually add new diagnostic capability to the IAND-*k* system, without having to reprogram or retrain the complete system.

The training samples are stored in an NSS database, Θ_{cfd} after the signature extraction process:

$$\Theta_{cfd} = \{(\mathbf{x}_i, y_i) \,|\, \mathbf{x}_i \epsilon \Re^m, y_i \epsilon \{cf_0, cf_1, cf_2, ..., cf_p\}\}_{i=1}^n \tag{5.1}$$

where $\mathbf{x}_i$ is the m-dimensional feature vector and y_i is the class label. The class label $y_i = cf_0$ for a healthy UD and $y_i = cf_1, cf_2, ..., cf_p$ for *p* types of different UD faults.

Each module then selects the training data sub-set (Θ_{cf}^j) with traces labelled as cf_j for the faulty class and all the other traces cf_k where $k \neq j$ as the other class for training the j^{th} binary FCM, as in:

$$\Theta_{cf}^j = \{(\mathbf{x}_i, y_i) \,|\, \mathbf{x}_i \epsilon \Re^m, y_i \epsilon \{cf_k, cf_j\}\}_{i=1}^n, \tag{5.2a}$$

$$y_i \equiv cf_k \equiv -1, \tag{5.2b}$$

$$y_i \equiv cf_j \equiv +1 \text{ for } j = 1, ..., p. \tag{5.2c}$$

In CFD-P, *p* types of different UD faults will result in $p+1$ number of FCMs in the system, including a FCM for the healthy UD.

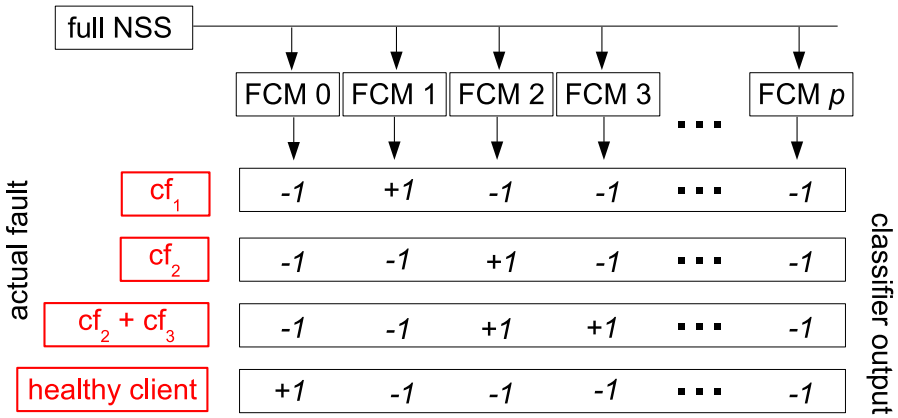


Figure 5.5: Diagnosis output of the CFD classifier.

Then, each FCM module independently processes data, improving the class coherency, and selects the unique feature sub-set (artefacts) that separates the two classes for the specific fault. This feature sub-set is then sent to the pattern classifier module to model the classifier boundaries. Each FCM in the CFD-P classifier uses L2 soft-margin SVMs for pattern classification similar to the LPD classifier design.

During the diagnosis phase, an undiagnosed trace is sent through all FCMs simultaneously, as shown in Figure 5.5. Each of the modules first filters the relevant feature set from the NSS. Then, each of the classifiers independently determines if the trace contains artefacts from the particular type of fault. By using the $+1$ FCM outcomes, we can determine the fault artefacts embedded in the NSS and consequently, the root cause. A $+1$ outcome in cf_0 FCM indicates that the user's device does not suffer from any faults the system is designed to detect.

CFD-M classifier design

The similarities in fault artefacts can sometimes mislead the FCMs in the CFD-P classifier, resulting in misclassification and false positives. In addition, the large dataset involved in training CFD-P classifiers can lead to a longer training delay. The CFD-M has been designed with a matrix of FCMs, each classifying a spe-

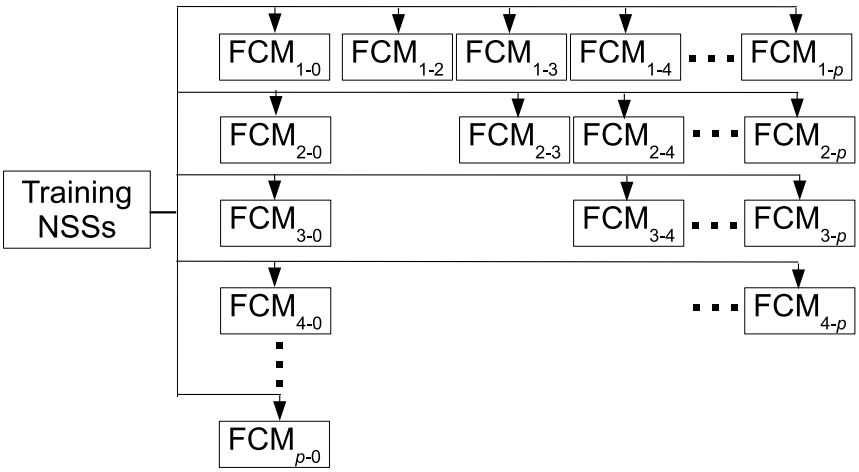


Figure 5.6: High-level CFD-M classifier design for the IAND-*k* system.

cific fault class against another class, either healthy or faulty. Figure 5.6 shows the high-level design of the CFD-M classifier, and each FCM comprises the same components as in Figure 5.4. Comparatively, CFD-M is a complex design and the number of FCMs exponentially increases when scaling the system.

The classifier still uses the same training NSS database, Θ_{cfd} in Eq. (5.1). However, when training individual FCMs, each module selects the training data subset ($\Theta_{cf}^{(j,k)}$) with traces labelled cf_j for class 1 and cf_k for class 2 for training the $(j,k)^{th}$ binary FCM as in:

$$\Theta_{cf}^{(j,k)} = \{(\mathbf{x}_i, y_i) \mid \mathbf{x}_i \epsilon \Re^m, y_i \epsilon \{cf_j, cf_k\}\}_{i=1}^{n}, \tag{5.3a}$$

$$y_i \equiv cf_j \equiv +1, \tag{5.3b}$$

$$y_i \equiv cf_k \equiv -1 \text{ for } j = 1, ..., p \text{ and } k = 0, 1, ..., p \text{ and } j \neq k. \tag{5.3c}$$

In CFD-M, *p* types of different UD faults will result in $p(p+1)/2$ number of FCMs in the system.

During the diagnosis phase, an undiagnosed trace is sent through all FCMs simultaneously. Each of the modules first filters the relevant feature set from the NSS. Then, each of the classifiers independently determines the class of the NSS out of the two classes the specific FCM has been trained to detect, based on the

artefacts and labels the output with $cf_0, cf_1, cf_2, ..., cf_p$. Recognition of the correct class is achieved by maximum voting, where each FCM votes for one class. If two classes receive the same number of votes, the NSS is determined to have multiple faults. If the majority class is cf_0, the UD is determined to be healthy.

5.1 Performance evaluation

This section discusses the performance evaluation conducted for the IAND-*k*UD system. The evaluation included an analysis of the performance of the IAND-*k*UD system with two LPD classifiers (for two types of access links), and the CFD classifier in both CFD-P and CFD-M configurations. The diagnostic system was set up on test-bed 1 (TB-1) introduced in Section 3.3. The experimental set-up emulated a full duplex wired access link with a 80 Mb/s bandwidth, 10 ms delay with no packet losses and no packet reordering as one healthy link (L-1), and 16 Mb/s bandwidth, 20 ms delay with no packet losses and no packet reordering as another healthy link (L-2). Faulty links were emulated by inducing packet

Fault	Description
CF0	Healthy
CF1	Disabled SACK error
CF2	Insufficient write buffer
CF3	Insufficient read buffer
CF4	Disabled D-SACK error
CF5	TCP timestamps are not working/in error
CF6	Window scaling error
CF7	Limited reordering threshold
CF8	Link-UD speed mismatch
CF9	Link-UD speed mismatch & duplex mismatch
CF10	UD firewall causing packet loss
CF11	UD firewall causing packet delay
CF12	Overloaded UD CPU
CF13	Overloaded UD memory
CF14	UD HDD i/o overloaded - faulty

Table 5.1: Key: List of faults

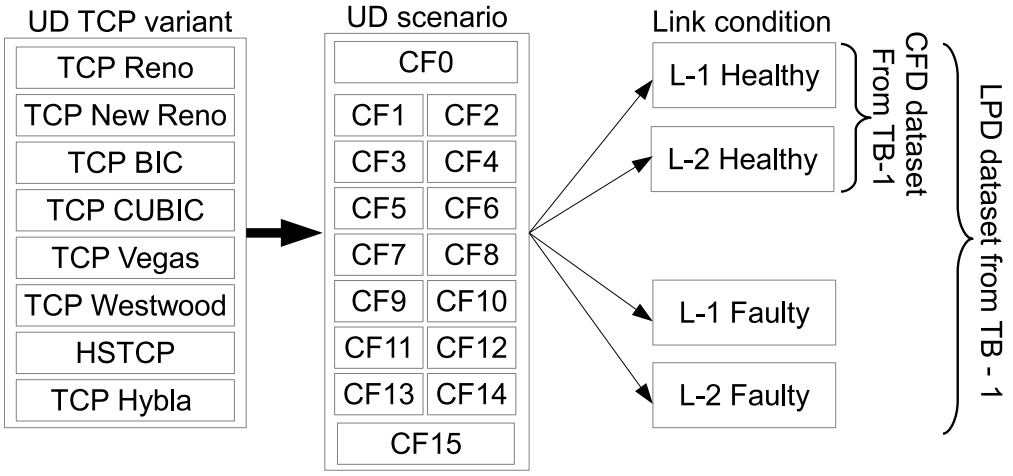


Figure 5.7: Datasets created for training and testing the LPD and CFD classifiers. This dataset was collected from TB-1, the laboratory emulated network.

losses (from 1% up to 10%) and increased delays (from 15ms up to 100ms), making a total of 10 faulty scenarios. Both the access router and the healthy UD (Linux 2.6.32) had a protocol stack optimised for the healthy link. We emulated 14 faults in the UD as listed in Table 5.1, and the CFDs were configured to diagnose all 14 faults and the healthy UD. We also collected data for UDs simultaneously suffering from insufficient read & write buffer (CF15) to evaluate the system's capability in diagnosing multiple simultaneous faults. We collected data for the eight TCP variants listed in Table 3.2. Figure 5.7 shows the full dataset from TB-1 used for LPD and CFD classifier training and testing. With the 10 faulty link scenarios, 2 healthy links, 8 TCP types and 15 UD conditions, the TB-1 dataset had a total of 144,000 NSS samples.

5.1.1 Diagnostic performance of LPD classifier

The training database for the LPD classifier Θ_{lpd} contained two classes, faulty and healthy links, and a sample visual representation of the NSSs is shown in Figure 5.8.

The feature selection technique proposed in Section 4.2.1 requires cross-validation

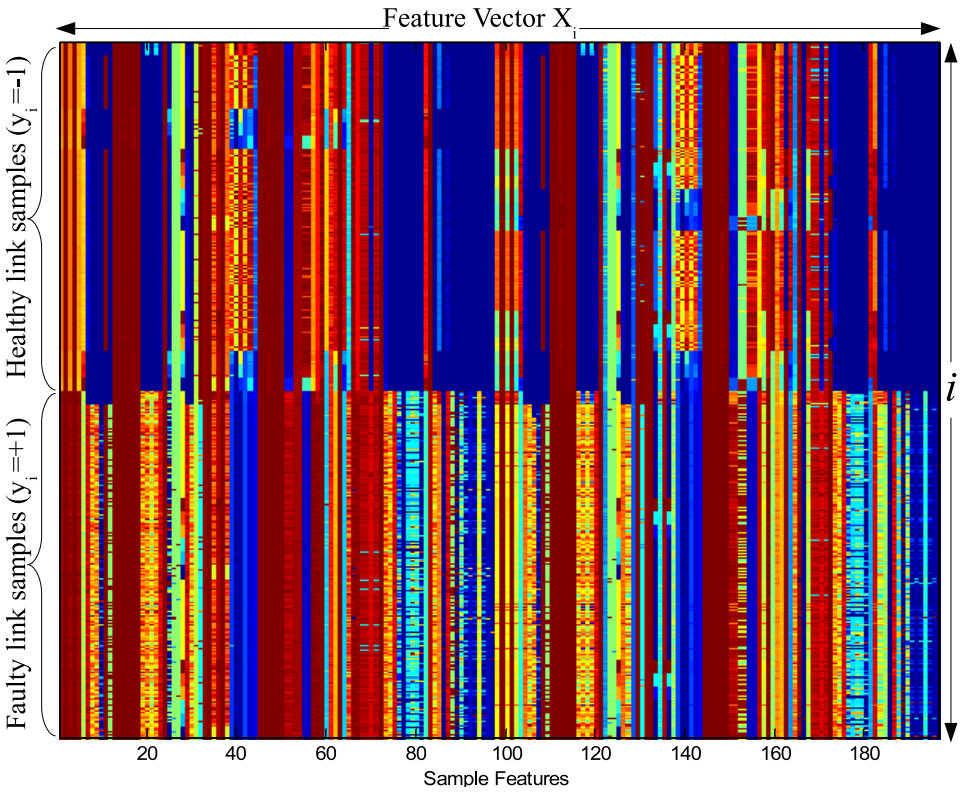
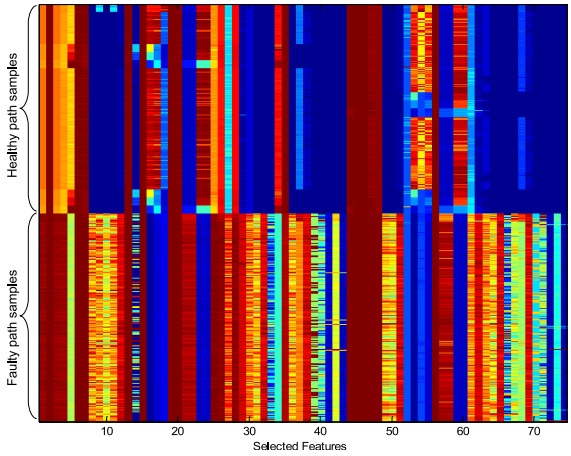


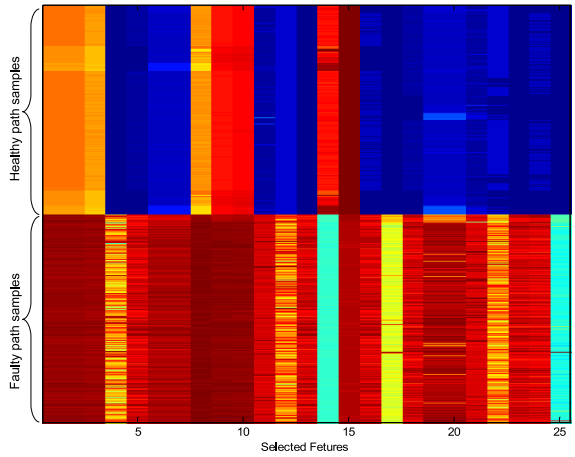
Figure 5.8: LPD classifier NSS database, Θ_{lpd} for comparison of faulty and healthy link NSS characteristics. Here, only partial NSSs have been shown for visual clarity.

before selecting the best feature sub-set. We cross-validated a number of feature sub-sets as discussed in detail during FCM performance evaluation reported in Section 4.4. For example, Figure 5.9 shows the two Θ_{lpd} databases, where Figure 5.9a has 75 chosen features, and Figure 5.9b has 25. When compared to Figure 5.8, both of these feature-limited NSS datasets show a clearer separation between the two classes. The feature selection technique has reduced the dimensionality of the problem by 73% and 91% with 75 and 25 feature sub-sets, respectively.

To evaluate the detection accuracies, first we trained the two LPD classifiers with NSSs from a mix of TCP types. One class was the faulty link, and the other class was the healthy link. Although we had a large number of samples, we limited the training sets to a smaller set of samples to evaluate the LPD classifier



(a) 75 sorted features selected after Student's t-test feature selection



(b) 25 sorted features selected after Student's t-test feature selection

Figure 5.9: Comparison of NSS databases after feature selection.

performance in a more data-deprived scenario. All the remaining data from the TB-1 dataset was used for testing the classifier. This is an important consideration, since collecting training samples is time-consuming, expensive and imposes additional deployment overhead on the IAND-*k* system. Accuracies were evaluated for the four types of SVM kernels given in Eq. (4.8).

Figures 5.10 and 5.11 show the LPD classifier accuracies over the numbers of training data used. As the figures show, for all kernel types, classifier accuracy

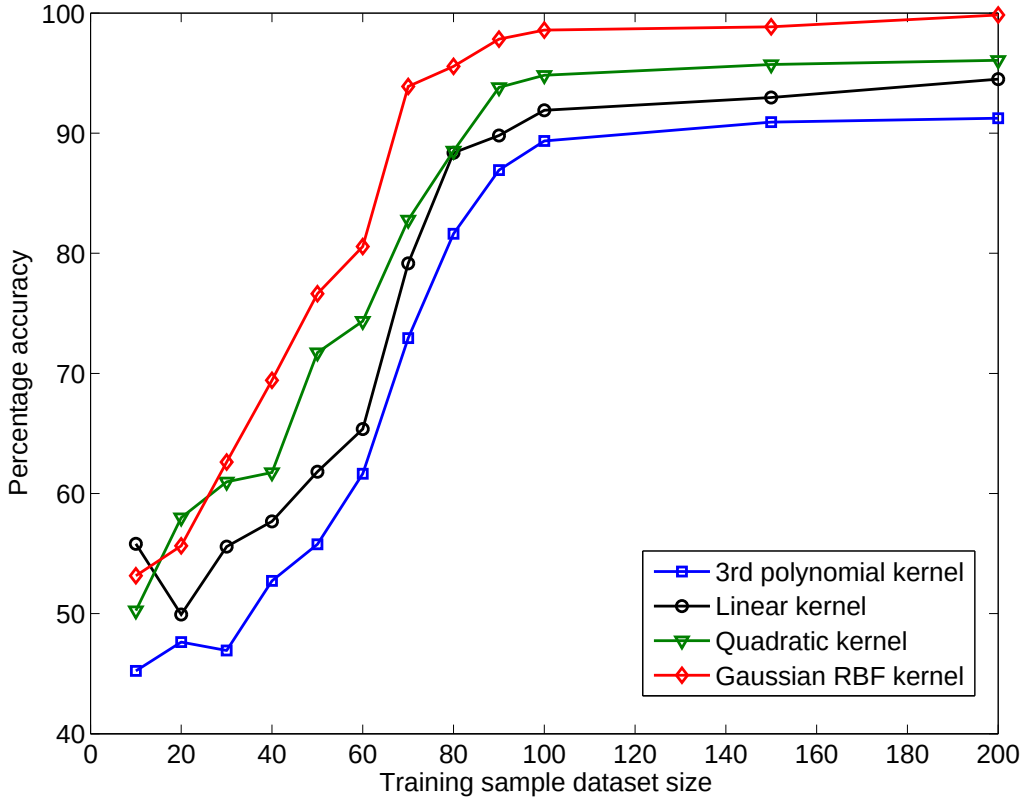


Figure 5.10: Percentage accuracy of L-1 link LPD classifier with training sample data set size for SVM kernels.

improved with increasing numbers of training samples being used. However, the rate of improvement diminished with more samples. The results show that when deploying the LPD classifier with the IAND-*k* system, there is a trade-off between the number of samples being used and the overall accuracy of the classifier. Over time, as the system collects more and more data, the accuracy will continue to increase, although a reasonable starting point can be achieved with around 100 samples per class. Table 5.2 shows the overall accuracies achieved by the LPD classifier with 100 training samples per class and the optimum feature sub-set.

L-1 TB-1 detection accuracy	Optimum features	L-2 TB-1 detection accuracy	Optimum features
98.57%	34	97.46%	58

Table 5.2: LPD classification accuracy and optimum feature sets for mixed TCP datasets L-1 TB-1 and L-2 TB-2

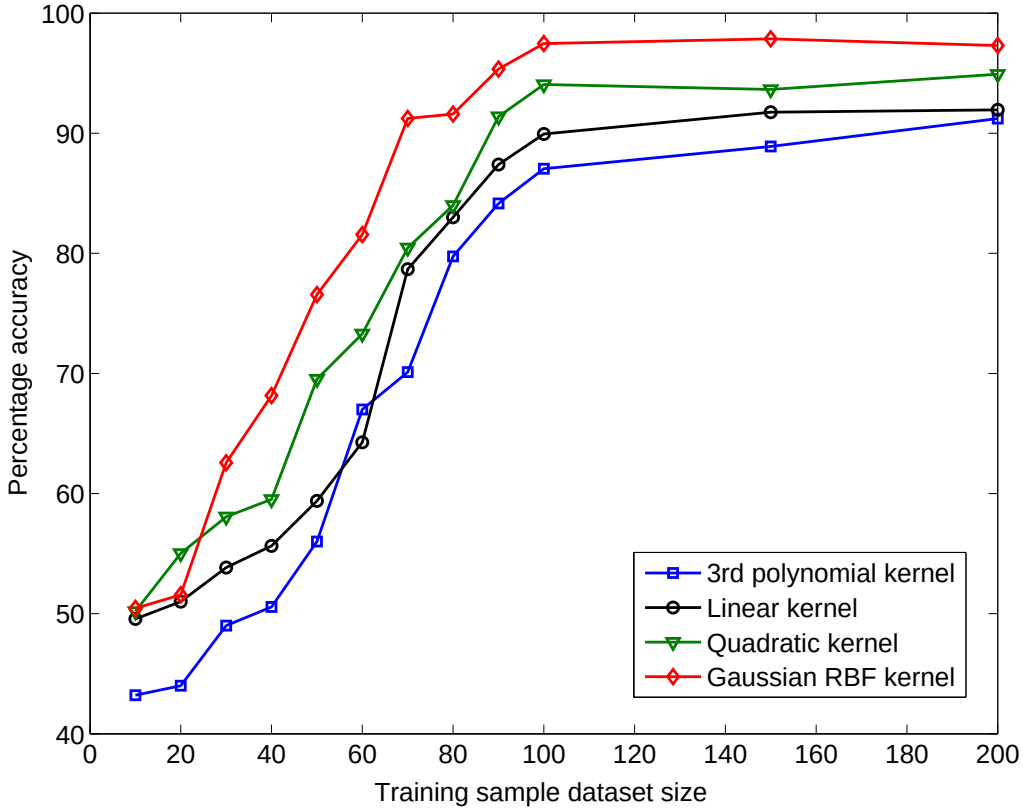


Figure 5.11: Percentage accuracy of L-2 link LPD classifier with training sample data set size for SVM kernels.

For both LPD classifiers, the Gaussian RBF kernel achieved the best performance. The results suggest that, with careful selection of training parameters, the LPD classifier can differentiate between a faulty link and a healthy link with up to 97% – 98% accuracy in these scenarios.

UD TCP variant	L-1 TB-1 detection accuracy (%)	L-2 TB-1 detection accuracy (%)
TCP Reno	93.26	92.33
TCP New Reno	98.33	97.11
TCP BIC	92.02	93.48
TCP CUBIC	94.49	93.7
HSTCP	93.35	95.72
TCP Vegas	92.56	95.79
TCP Westwood	90.63	91.93

Table 5.3: LPD classification accuracy for each each UD TCP variant when trained with TCP New Reno datasets for L-1 TB-1 and L2-TB-2

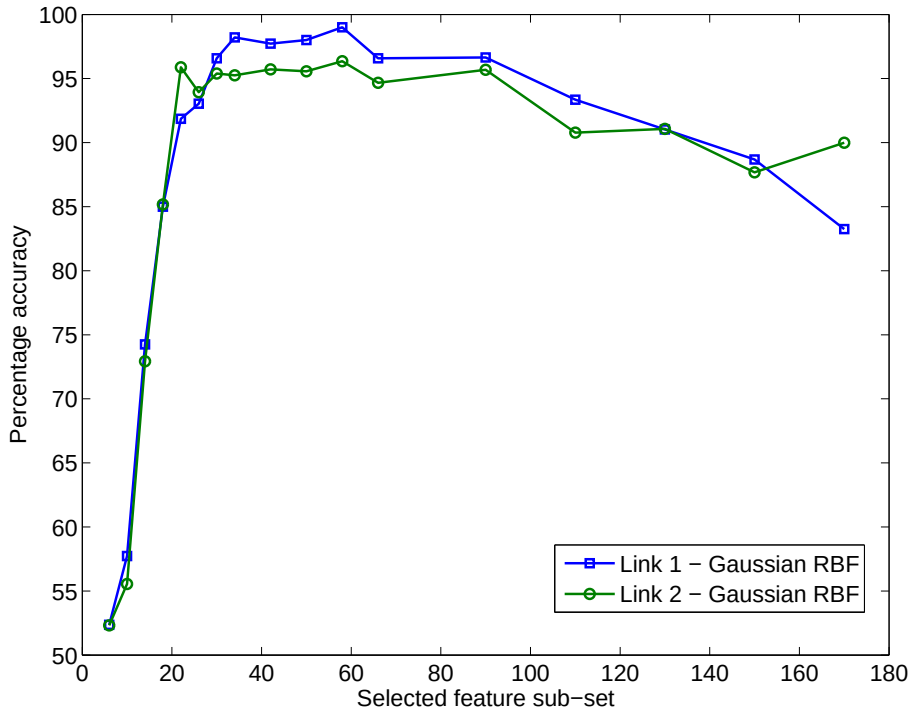


Figure 5.12: Percentage accuracy of both LPD classifiers with training feature sub-set.

In the next scenario, we used NSSs from only one type of TCP, TCP New Reno for training LPD classifiers. In most situations, it is difficult to collect all the different variations of TCPs to train the system. However, it is important for the classifier to have sufficient generalisation capability at the class boundary, so that it can compensate for the subtle variations in NSSs between TCP variants. The testing dataset included all eight TCP types, and the overall LPD classifier accuracies can be seen in Table 5.3. Figure 5.12 shows how the two LPD classifiers performed in this scenario with varying numbers of feature sub-sets. As the feature set increases, the complexity of the classifier increases and generalisation capability decreases. The figure clearly shows that detection accuracy starts to diminish as the feature set increases after reaching the initial peak. This is due to the classifier not being able to generalise sufficiently to compensate for NSS variations between TCP types. However, the overall results show that, with careful selection of the feature sub-set, the LPD classifier can successfully differentiate

faulty links from the healthy ones, even when trained with limited TCP variations. Given that the testing data for the healthy link class included NSSs from faulty UDs, the LPD classifier also managed to successfully distinguish between the NSSs variations caused by the UD faults and the link issues.

5.1.2 Diagnostic performance of CFD classifier

Both CFD classifier configurations, CFD-P and CFD-M, were evaluated using the same two datasets collected from two different links on test-bed 1, L-1 TB-1 and L-2 TB-2. However, for the CFD evaluation, we used only the data collected over healthy links, as the CFD classifier only operates after the LPD classifier ensures that the link is healthy. Both datasets contained 14 faulty cases and one healthy UD case for training the classifier and resulted in 15 FCMs for CFD-P and 105 FCMs for CFD-M.

In the first evaluation, the FCMs were trained using NSSs with all types of TCP variants. For every fault, we had 800 NSSs per link. We varied the number of training samples per fault between 10 – 200 to evaluate the impact the number of samples has on performance. CFD-P FCMs have two classes, one with a specific fault and another with all the other scenarios. Given that we had identical sample sizes across all the faults, each of the 15 FCMs in the CFD-P classifier had a $1:14$ class data ratio and the total training samples ranged between $10:140$ to $200:2800$. In contrast, each of the 105 FCMs in the CFD-M classifier were separated between two UD scenarios and had a $1:1$ class data ratio, which resulted in training datasets between $10:10$ to $200:200$. The rest of the samples were used as the validating data, and both training and testing samples were chosen at random for four iterations of the experiment to achieve more statistically robust outcomes. The results of the iterations were averaged to produce the final results.

Interpretation of the CFD-P classifier output was straight-forward as the validation NSSs were assigned to one of 15 classes, i.e. CF0-CF14. Fault association

from CFD-M output was interpreted using maximum voting, where each FCM has a single vote. The NSS is assigned to the class that receives the highest number of votes and then mapped onto a decision matrix with axis, actual labels and detected labels for cross-validation. Each FCM module was trained and tuned independently to have the best possible feature sub-set and SVM kernel. The metrics used in the performance analysis are defined as follows:

1. True-Positive Rate (TPR): Members of class X correctly classified as belonging to class X.

2. False-Positive Rate (FPR): Members of other classes incorrectly classified as belonging to class X.

3. True-Negative Rate (TNR): Members of other classes correctly classified as belonging to other classes.

4. False-Negative Rate (FNR): Members of class X incorrectly classified as not belonging to class X.

5. Accuracy: Total correct classifications achieved by the classifier.

CFD-P performance

Table 5.4 and Table 5.5 show the overall performance of the CFP-P classifier FCMs when trained with mixed TCP NSSs for the two datasets. The FCMs created had different optimum kernel functions and feature sub-sets to achieve the best accuracies. Evaluating performance across 10-200 training samples per fault (150-3000 total dataset), the FCM showed increasing accuracy with the training set, but the rate of improvement diminished, similar to the LPD classifier. We chose 100 samples per class for further evaluation, as this provided the best balance between accuracy and training overheads. The results for both datasets showed similar

patterns, but the L-2 data set has slightly less accuracy. This can be attributed to the impact the low bandwidth of the link has on the NSS consistency.

In addition, the performance evaluation included in-depth analysis of misclassification, as every NSS is sent through the full CFD when diagnosing an unknown problem. It is crucial to have a low FPR to avoid misidentification of faults. The results show that FCMs achieved 90% – 97% TPR and 1% – 9% FPR. The diagnostic accuracy of the FCMs, which accounts for both correct identifications and also correct omissions, also ranged between 90% – 97%, and the CFD-P classifier achieved an overall accuracy of 96.61% for the L-1 TB-1 dataset and 95.32% accuracy for the L-2 TB-1 dataset. The results also show that the FCMs consistently diagnosed problems with high accuracy, apart from FCM4, which was designated to detect the CF4 - DSACK error. As discussed earlier, DSACK error NSSs were

	Parameters		Performance				
Classifier	Optimum features	Kernel	TPR (%)	FPR (%)	TNR (%)	FNR (%)	Diagnostic accuracy
FCM0	36	Gaussian RBF	97.54	2.46	98.22	1.78	98.17
FCM1	38	Gaussian RBF	93.28	6.72	94.00	6.00	93.95
FCM2	26	Quadratic	98.39	1.61	99.32	0.68	99.26
FCM3	34	Linear	97.88	2.12	98.52	1.48	98.48
FCM4	66	Quadratic	90.09	9.91	90.86	9.14	90.81
FCM5	42	Linear	96.17	3.83	96.82	3.18	96.77
FCM6	50	Gaussian RBF	94.24	5.76	95.18	4.82	95.12
FCM7	48	Gaussian RBF	98.65	1.35	99.27	0.73	99.23
FCM8	22	Quadratic	95.65	4.35	96.58	3.42	96.52
FCM9	46	Linear	96.35	3.65	97.12	2.88	97.07
FCM10	34	Quadratic	97.88	2.12	98.56	1.44	98.51
FCM11	30	Gaussian RBF	96.33	3.67	96.33	3.67	96.33
FCM12	74	Gaussian RBF	93.50	6.50	94.14	5.86	94.10
FCM13	58	Gaussian RBF	97.51	2.49	98.44	1.56	98.38
FCM14	42	Quadratic	95.28	4.72	96.52	3.48	96.44
Overall							96.61

Table 5.4: CFD-P classifier FCM parameters and diagnostic performance for L-1 TB-1 dataset with mixed TCP training. Each FCMx classifier detects CFx fault. Trained with 100 samples per UD fault, 1500 total training sample set.

Classifier	Parameters		Performance				
	Optimum features	Kernel	TPR (%)	FPR (%)	TNR (%)	FNR (%)	Diagnostic accuracy
FCM0	34	Gaussian RBF	96.41	3.59	97.09	2.91	97.04
FCM1	28	Gaussian RBF	92.49	7.51	93.21	6.79	93.16
FCM2	22	Quadratic	96.94	3.06	97.87	2.13	97.81
FCM3	34	Linear	96.09	3.91	96.73	3.27	96.69
FCM4	50	Quadratic	89.01	10.99	89.78	10.22	89.73
FCM5	34	Linear	94.84	5.16	95.49	4.51	95.44
FCM6	58	Gaussian RBF	93.27	6.73	94.21	5.79	94.15
FCM7	32	Gaussian RBF	97.74	2.26	98.36	1.64	98.32
FCM8	30	Linear	94.40	5.60	95.33	4.67	95.27
FCM9	42	Quadratic	94.47	5.53	95.24	4.76	95.19
FCM10	34	Quadratic	96.39	3.61	97.07	2.93	97.02
FCM11	50	Gaussian RBF	95.80	4.20	95.80	4.20	95.80
FCM12	90	Linear	92.56	7.44	93.20	6.80	93.16
FCM13	74	Gaussian RBF	95.93	4.07	96.86	3.14	96.80
FCM14	50	Quadratic	93.61	6.39	94.33	5.67	94.28
Overall							95.32

Table 5.5: CFD-P classifier FCM parameters and diagnostic performance for L-2 TB-1 dataset with mixed TCP training. Each FCM*x* classifier detects CF*x* fault. Trained with 100 samples per UD fault, 1500 total training sample set.

harder to distinguish from SACK errors and consequently showed a relatively high FPR .

CFD-M performance

Tables 5.6 and 5.7 show the overall performance of the CFP-M classifier when trained with mixed TCP NSSs for the two datasets. The FCMs created had different optimum kernel functions and feature sub-sets to achieve the best accuracies. The analysis was not as straight-forward as in the CFD-P because the classification results from 105 FCMs first had to be consolidated, based on the majority of votes into class labels. Then, the detected classes were mapped to a decision matrix to calculate the TPRs, FPRs and final accuracies for each fault. When training CFP-M, each FCM was trained with a smaller dataset, since only two specific

faults are considered per FCM. Evaluating performance across 10-200 training samples per fault (20-400 total dataset), the FCMs showed increasing accuracy with the training set, but the rate of improvement diminished, similar to the LPD and CFD-P classifiers. We chose 100 samples per class for further evaluation, as this provided the best balance between accuracy and training overheads.

The results in Tables 5.6 and 5.7 from the CFD-M classifier show that there is only a slight difference in diagnostic accuracy between the CFD-M and CFD-P. For the same dataset L-1 TB-1, CFD-M achieved 94.88% overall accuracy, while CFD-P achieved 96.61%. Throughout the evaluation, we observed that CFD-P slightly outperformed CFD-M for each of the faults by a small margin. However, the CFD-M classifier still managed to achieve TPRs above 90% and FPRs below 10%, except for the case of CF4. We also observed that the selected feature sub-set for each FCM was noticeably smaller than that of CFD-P, given that the

UD scenario	TPR (%)	FPR (%)	TNR (%)	FNR (%)	Diagnostic accuracy
CF0	95.87	4.13	96.54	3.46	96.50
CF1	91.56	8.44	92.28	7.72	92.23
CF2	96.46	3.54	97.39	2.61	97.33
CF3	96.25	3.75	96.88	3.12	96.84
CF4	88.32	11.68	89.09	10.91	89.04
CF5	94.53	5.47	95.17	4.83	95.13
CF6	92.30	7.70	93.24	6.76	93.18
CF7	97.02	2.98	97.65	2.35	97.61
CF8	93.73	6.27	94.65	5.35	94.59
CF9	94.58	5.42	95.35	4.65	95.30
CF10	96.21	3.79	96.88	3.12	96.84
CF11	95.32	4.68	95.33	4.67	95.33
CF12	91.87	8.13	92.50	7.50	92.46
CF13	95.58	4.42	96.51	3.49	96.45
CF14	93.56	6.44	94.28	5.72	94.23
Overall					94.88

Table 5.6: CFD-M classifier diagnostic performance for L-1 TB-1 dataset with mixed TCP training. Trained with 100 samples per UD fault, 1500 total training sample set.

UD scenario	TPR (%)	FPR (%)	TNR (%)	FNR (%)	Diagnostic accuracy
CF0	94.74	5.26	95.41	4.59	95.37
CF1	90.77	9.23	91.49	8.51	91.44
CF2	95.01	4.99	95.94	4.06	95.88
CF3	94.46	5.54	95.09	4.91	95.05
CF4	87.24	12.76	88.01	11.99	87.96
CF5	93.20	6.80	93.84	6.16	93.80
CF6	91.33	8.67	92.27	7.73	92.21
CF7	96.11	3.89	96.74	3.26	96.70
CF8	92.48	7.52	93.40	6.60	93.34
CF9	92.70	7.30	93.47	6.53	93.42
CF10	94.72	5.28	95.39	4.61	95.35
CF11	94.79	5.21	94.80	5.20	94.80
CF12	90.93	9.07	91.56	8.44	91.52
CF13	94.00	6.00	94.93	5.07	94.87
CF14	91.89	8.11	92.61	7.39	92.56
Overall					93.62

Table 5.7: CFD-M classifier diagnostic performance for L-2 TB-1 dataset with mixed TCP training. Trained with 100 samples per UD fault, 1500 total training sample set.

within-class data is more consistent. The average feature sub-set size after feature selection for CFD-M FCMs was 32, while the CFD-P FCMs was 44 on average.

The complexity of the training process

It may appear logical that the total training time with the CFD-P design is larger than with the CFD-M, because it is necessary to train more binary classifiers. However, this is not true when the binary classifiers are SVMs. Indeed, the training time of an SVM increases more than linearly with the number of training samples. In addition, CFD-P requires a higher dimensionality model to achieve the same performance as CFD-M due to the complex, varied class data. Therefore, since each SVM involves a small number of training samples and is easier to solve, it is quicker to train the 105 SVMs of the CFD-M FCMs than the 15 SVMs of the CFD-P FCMs. Table 5.8 shows a comparison of training times for three training

		CFD-P	CFD-M
200 training samples per fault	Total FCM dataset	3000	400
	Average feature sub-set	45	35
	Total training time	5min 34s	1min 05s
	Support vectors	554	289
	Single NSS diagnosis time	368ms	96ms
100 training samples per fault	Total FCM dataset	1500	200
	Average feature sub-set	44	32
	Total training time	3min 28s	55s
	Support vectors	437	192
	Single NSS diagnosis time	259ms	74ms
50 training samples per fault	Total FCM dataset	750	100
	Average feature sub-set	42	30
	Total training time	2min 56s	37s
	Support vectors	357	102
	Single NSS diagnosis time	206ms	52ms

Table 5.8: CFD-M classifier diagnostic performance for L-2 TB-1 dataset with mixed TCP training. Trained with 100 samples per UD fault, 1500 total training sample set.

dataset sizes. In both cases, we parallelised the training of SVMs using parallel processing and multi-access, object store databases to achieve the best possible training times. The results clearly show that, even with the larger number of FCMs, CFD-M is significantly quicker to train than the CFD-P classifier.

The diagnosis complexity

Again, it may appear logical that diagnosis with the CFD-M classifier is more complex than with the CFD-P classifier, because it is necessary to evaluate more decision functions. However, as previously, this is not necessarily true with SVMs. Indeed, the complexity of SVM decision-making is directly linked to the number of support vectors (SVs), and although decision-making is more complicated in the CFD-M's case, it is reasonable to consider that the complexity is proportional to the total number of support vectors. As shown in Table 5.8, the CFD-P FCMs used more support vectors than the CFD-M and consequently, the time to diagnose a single NSS sample was longer in the CFD-P. Given that CFD-P marginally

UD scenario	L-1 TB-1 dataset CFD-P accuracy	L-1 TB-1 dataset CFD-M accuracy	L-2 TB-1 dataset CFD-P accuracy	L-2 TB-1 dataset CFD-M accuracy
CF0	96.79	94.85	95.85	92.41
CF1	92.31	90.67	91.01	89.93
CF2	97.42	96.18	96.05	94.81
CF3	97.32	95.11	94.97	94.04
CF4	89.43	87.58	88.00	86.43
CF5	94.51	93.42	93.57	91.50
CF6	92.45	92.06	92.92	90.76
CF7	96.97	95.86	96.45	94.73
CF8	95.12	93.44	93.92	91.96
CF9	95.38	93.83	92.21	90.93
CF10	96.33	95.19	95.34	93.73
CF11	95.04	92.34	94.28	92.96
CF12	91.80	91.32	90.86	89.88
CF13	96.92	95.11	95.18	93.12
CF14	94.47	91.26	92.05	91.42
Overall accuracy	94.82	93.22	93.51	91.91

Table 5.9: CFD-P and CFD-M classifiers diagnostic performance for TB-1 datasets when trained with TCP New Reno, and validated with all eight TCP types.

outperforms CFD-M on diagnostic accuracy, but CFD-M is quicker to train and quicker to diagnose, the selection of the CFD classifier becomes a design decision based on the specific IAND-*k* application. If speed of diagnosis and ability to quickly add more faults to the system is more important than a slight loss of accuracy, then CFD-M provides the best option. If the accuracy of the outcome is more important than the speed to a diagnosis or the time to re-train the system, CFD-P offers the best option.

Training only with TCP New Reno NSSs

Lastly, both CFD-P and CFD-M classifiers were evaluated using only TCP New Reno as the training dataset. The classifiers were then tested with NSSs with multiple TCP variants to evaluate the classifier's ability to generalise the class boundaries to compensate for variations in the NSSs. Table 5.9 shows the results

achieved in this experiment. The results were consistent with previous results achieved under similar training data constraints. The overall accuracy reduced by a small amount compared to the previous mixed TCP training dataset, due to the misclassification of ambiguous NSSs. However, even with the severely limited training dataset, the FCMs were capable of detecting UD faults with over 91% overall accuracy, sufficient for a functional diagnostic system.

5.1.3 IAND-*k* system characteristics

For the root cause diagnosis of UD soft failures, the proposed IAND-*k* system offers many advantages over the other inference methods currently available:

- The system offers a fully-automated, comprehensive framework which is extendible to diagnose a diverse range of faults, in contrast to the limited capabilities of other tools.

- The diagnostic capability of the system evolves with the diversity of the fault signature databases, instead of the inference algorithm. Users can collaborate to create common signature repositories, encompassing a wide range of faults, networks, and device platforms. Most rule-based systems lack the generality to operate effectively in a dynamic environment.

- The system relies solely on packet traces collected at end-points and can be implemented as an application. This provides flexibility for the operator to deploy the IAND-*k* system at any desired network location.

- Although the system is designed to diagnose users' computers from the edge of the operator's network, the same system can be used for diagnosing intermediate nodes in the network by deploying a trace collection module in a neighbouring node and training with suitable data.

5.2 IAND-*k*CC system for private clouds

With the rapid adoption of cloud applications by ordinary, non-expert users, "always available" network connections and consistently fast communication speeds are becoming critically important. The networking research community has converged on the common understanding that performance unpredictability and data-transfer bottlenecks are going to be significant obstacles to satisfactory cloud computing experience. In addition, automation is a mandatory requirement for achieving highly scalable cloud services, which presents a significant research challenge for the creation of comprehensive diagnostic solutions.

We propose the use of the fault signature-based inference technique to automate the diagnosis of user device bottlenecks in a private cloud environment. The IAND-*k* for cloud computing (IAND-*k*CC) system essentially uses a modular framework with the FCMs proposed in the previous section to identify the

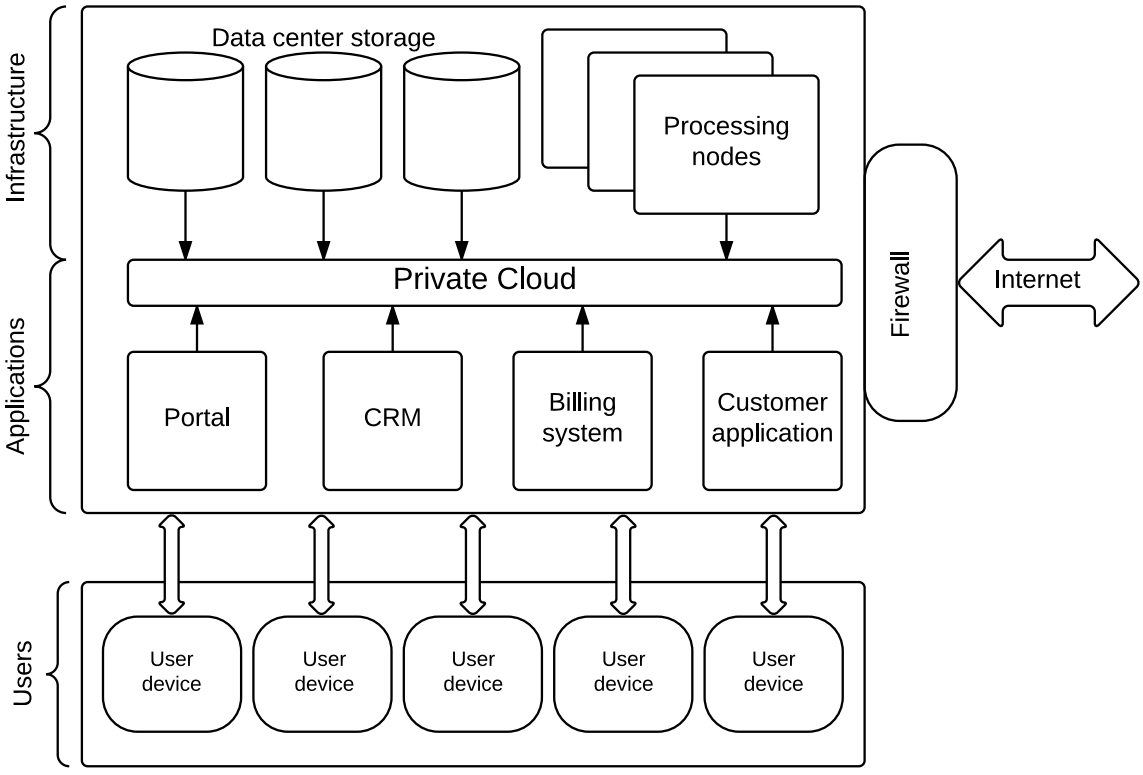


Figure 5.13: Private cloud computing environment.

faults at UDs that affect the performance of the whole cloud network.

A private cloud is the phrase used to describe a cloud computing platform that is implemented within a corporate firewall, under the control of the IT department (see Figure 5.13). A private cloud is designed to offer the same features and benefits of public cloud systems, but removes a number of objections to the cloud computing model, including control over enterprise and customer data, worries about security, and issues connected to regulatory compliance. The private cloud model is closer to the more traditional model of individual local access networks (LANs) used in the past by enterprise, but with the added advantages of virtualisation.

A key advantage the private cloud offers is a greater degree of control; as a private cloud is only accessible by a single organisation, that organisation will have the ability to configure and manage it, consistent with their needs, to achieve a tailored network solution. Although private cloud networks can be realised in many technical architectures, most of the solutions provide a close link similar to traditional LAN services between the user devices and core servers. Most organisational users' devices will have constant communications with the cloud servers and between the devices themselves and reliable, optimised connections is critical to daily productivity. The IAND-*k*CC system in a private cloud computing environment can provide organisations with quick and automated diagnosis of the network performance issues experienced by users due to failures in their devices.

5.2.1 Deployment

The deployment of the IAND-*k*CC system in a private cloud is shown in Figure 5.14. The diagnostic server is located on a central server, while the trace collection module is deployed in the core access router of the data centre. This is typically the primary access conduit into the data centre for all users. However, depending on the architecture, trace capturing modules can be deployed in

distributed sets of edge routers. Users engage with the system when they are experiencing a network issue, and a trace capture module is then loaded into their device. This mechanism has been explained in detail in Section 3.2.

When the IAND-*k*CC system is first deployed, the system has to be trained with traces collected over the specific network. The training data can also be sourced using the LASE technique described in Section 3.6. However, since most enterprise networks do not change rapidly, once trained, the system can diagnose soft failures in devices without having to re-train regularly.

5.2.2 Performance evaluation

We deployed the IAND-*k*CC system using test-bed 2 described in Section 3.3. As shown in Figure 3.8, the UDs and the access router were connected through the live campus network. This set-up closely represents the private cloud scenario, and the live network provided real-world data to conduct a thorough analysis of performance.

This dataset to train and test the system included the same 15 UD scenarios shown in Table 5.1 and included all 8 types of TCPs at the UD. The data set was

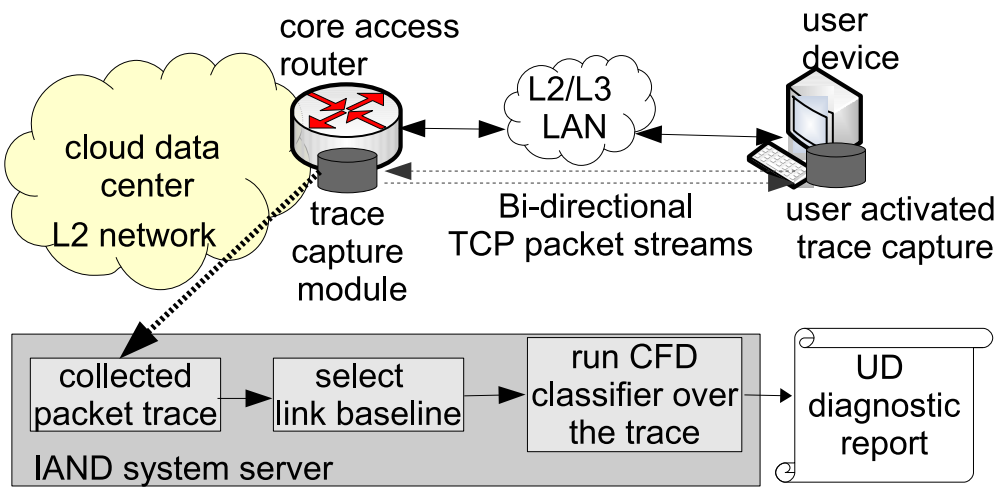


Figure 5.14: Deployment of the IAND-*k*CC system in a private cloud environment.

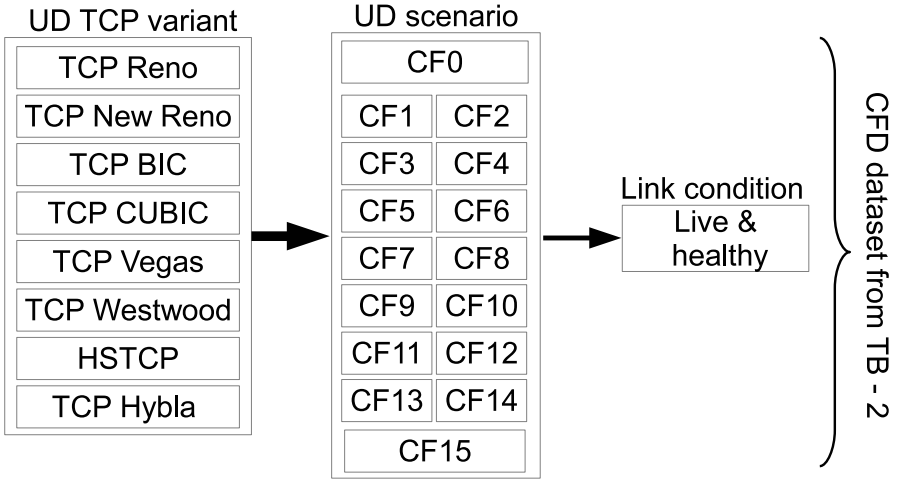


Figure 5.15: Datasets created for training and testing the IAND-*k*CC CFD classifiers. These datasets were collected from the TB-2, live network test-bed.

purely used to evaluate CFD classifier performance, with the assumption that links performed optimally without any major failures. Minor fluctuations are inherent in any network and the live network we used had cross traffic and normal congestion. The total dataset (TB-2) is shown in Figure 5.15 and had a total of 24,000 NSS samples, with 200 samples per scenario.

Both CFD-P and CFD-M configurations, with 15 FCMs and 105 FCMs respectively, were run to compare the overall diagnostic performance achieved per fault. The FCMs were trained with 100 samples per fault, and NSSs with a mix of TCP variants were used in training. The validation set included 700 samples per fault.

Table 5.10 shows the summary of the results. The patterns of accuracy variation with changing training datasets and the impact on FCM specific parameters were similar to that of the CFD evaluation discussed previously in this chapter. Table 5.10 shows the CFD-P classifier had an overall detection accuracy of 94.54% and CFD-M 92.85%. Both classifiers were able to identify all the classes with accuracies above 90%, with the exception of CF4, and achieved a TPR above 91% and FPR below 10%.

	CFD-P classifier					CFD-M classifier				
UD scenario	TPR (%)	FPR (%)	TNR (%)	FNR (%)	Detection accuracy	TPR (%)	FPR (%)	TNR (%)	FNR (%)	Detection accuracy
CF0	95.06	4.94	96.38	3.62	96.29	95.06	4.94	96.38	3.62	94.48
CF1	95.88	4.12	96.39	3.61	96.36	95.88	4.12	96.39	3.61	94.54
CF2	95.77	4.23	97.12	2.88	97.03	95.77	4.23	97.12	2.88	95.10
CF3	90.99	9.01	92.08	7.92	92.01	90.99	9.01	92.08	7.92	90.51
CF4	87.83	12.17	88.68	11.32	88.62	87.83	12.17	88.68	11.32	86.82
CF5	93.38	6.62	94.74	5.26	94.64	93.38	6.62	94.74	5.26	92.87
CF6	91.34	8.66	92.98	7.02	92.87	91.34	8.66	92.98	7.02	91.03
CF7	95.77	4.23	96.82	3.18	96.75	95.77	4.23	96.82	3.18	95.26
CF8	93.21	6.79	94.90	5.10	94.79	93.21	6.79	94.90	5.10	92.85
CF9	93.77	6.23	95.45	4.55	95.34	93.77	6.23	95.45	4.55	93.56
CF10	94.99	5.01	96.21	3.79	96.13	94.99	5.01	96.21	3.79	95.24
CF11	93.98	6.02	94.22	5.78	94.20	93.98	6.02	94.22	5.78	93.73
CF12	90.68	9.32	91.94	8.06	91.86	90.68	9.32	91.94	8.06	90.75
CF13	94.75	5.25	96.77	3.23	96.64	94.75	5.25	96.77	3.23	93.95
CF14	92.83	7.17	94.75	5.25	94.62	92.83	7.17	94.75	5.25	92.20
Overall accuracy					94.54					92.85

Table 5.10: Performance of the CFD classifiers on uniquely identifying UD faults in private cloud network.

Given that live networks contain dynamic links resulting in a larger noise component in NSSs, the results show that FCMs are capable of compensating for these variations and successfully diagnosing the issues. The diagnosis has been achieved with full automation, such that once the system is trained, no human intervention is needed until the diagnosis is completed. Once the failure is diagnosed, an administrator still has to fix the problem, and the system can be re-engaged to confirm the issue has been resolved.

5.3 Conclusion

In this chapter, we have introduced and evaluated the IAND-*k* system, an automated failure diagnostic system that uses intelligent inference from NSSs to automatically identify artefacts created by specific problems previously known. The system consists of a cascading set of supervised classifiers: (i) the LPD classifier, tasked with first filtering out whether the connection performance problem is caused by link faults or otherwise, (ii) the LFD classifier that diagnoses the exact root cause of a link failure, and (iii) the CFD classifier, tasked with diagnosing the specific UD faults that cause the connection problem. We specially focus on the IAND-*k* system's application on UD soft failure diagnosis (IAND-*k*UD). The LPD classifier uses a single FCM that incorporates NSS generation, feature selection and an L2, soft-margin, binary SVM for pattern classification. The CFD classifier performs a complex multi-class classification of UD faults, and we introduce two variants: CFD-P, which uses a parallel network of FCM modules, and CFD-M, which uses a matrix of FCM modules. The modular design of the CFD classifier offers extendibility to diagnose new faults by training new FCM modules independently.

We evaluated the system by diagnosing a number of common UD problems with various TCP implementations. Furthermore, we analysed the system's per-

formance in the absence of any UD faults, as well as when the training data is severely limited. Our results show that the LPD classifier can effectively identify and separate the link problems, without being affected by UD behaviour and TCP type. The CFD classifier results show that FCM modules collectively produce high diagnostic accuracy in all tested scenarios, and CFD-P slightly outperforms CFD-M in accuracy but takes longer to train and diagnose a new case.

Then we created the IAND-*k*CC, an application of the proposed IAND-*k* system, to automate the diagnosis of UDs in private cloud environments. We used data collected over a test-bed with a live network to evaluate the system's performance when operating in a real-world networking environment. The results show the IAND-*k*CC system is capable of diagnosing all the evaluated faults with a high degree of accuracy, while keeping false positives to a minimum.

The proposed IAND-*k* system provides a framework for an accurate diagnostic system that is scalable for a large array of user devices, easy to deploy, and extendible in diagnostic capability. To our knowledge, the IAND-*k* system is the first to use automated inferencing of TCP packet traces using SVMs for diagnosing the root causes of network performance issues.

CHAPTER SIX

KNOWN AND UNKNOWN SOFT-FAILURE DIAGNOSIS USING HYBRID CLASSIFIER ARCHITECTURE AND TRANSFORMED SIGNATURES

The ever-growing numbers of different types of connected devices and the complex dynamic nature of most networks mean that new types of performance problems and root causes are being created at an exponential rate. Whilst the diagnosis of a known fault is crucial, the ability to extend the scope of diagnostic capability to previously unknown faults with minimum human intervention adds more value to an automated fault diagnosis system.

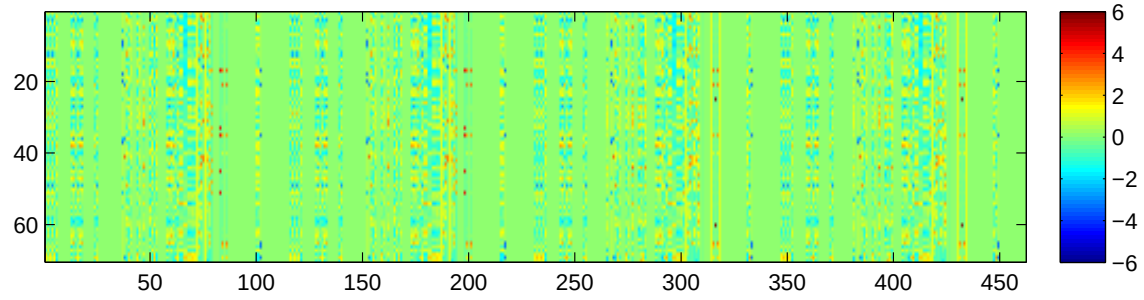
We have in previous chapters proposed IAND-*k*, an automated diagnostic systems based on supervised ML algorithms and normalised statistical signatures (NSSs) [221] and [248]. The ML algorithms are first trained using NSSs, which contain the artefacts embedded by the network faults and provide the fingerprint required for a diagnosis. However, due to the supervised nature of the system, the diagnostic capability is bound by the number of fault classes present and labelled in the training data set: known faults. In the case of diagnosing an unknown fault, this often leads to a false-positive error.

In this chapter, we present IAND-*h*, a diagnostic system based on a *hybrid classifier architecture* that combines both supervised and unsupervised ML algorithms

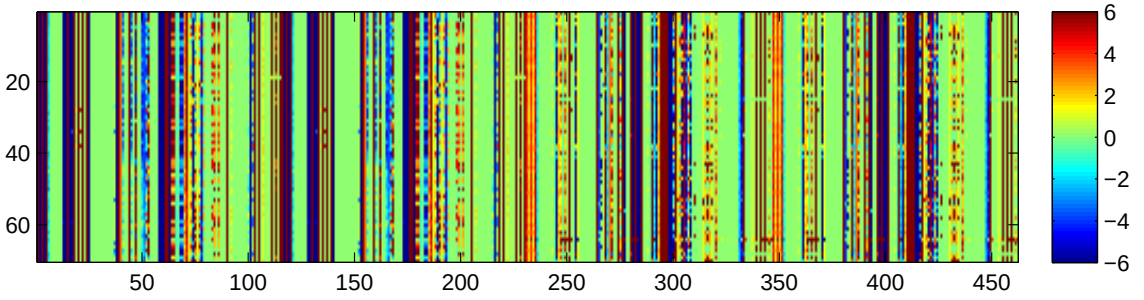
for the diagnosis of known and unknown faults in UDs. The IAND-*h* system architecture allows the unknown faults to be further analysed using several "clustering" algorithms [249, 250] to detect the presence of new faults. Clustering is the task of grouping sets of objects such that objects in the same group (cluster) are more similar than those in other groups.

NSSs introduced in previous chapters contain a large feature set (460 features) which enable them to characterise a broad range of faults with a single representation. However, as evident from Figure 6.1, not all features contribute equally to the separability of the classes, and even features within the same class show variations due to the inconsistent nature of the connection link. Having a large feature set can also lead to over-fitting ("curse of dimensionality" [241]) of the data when training an ML-based system; this will lead to poor generalisation and classification accuracy. Therefore, the dimensionality of the NSSs should be reduced while preserving the important information, to build a more effective diagnostic system.

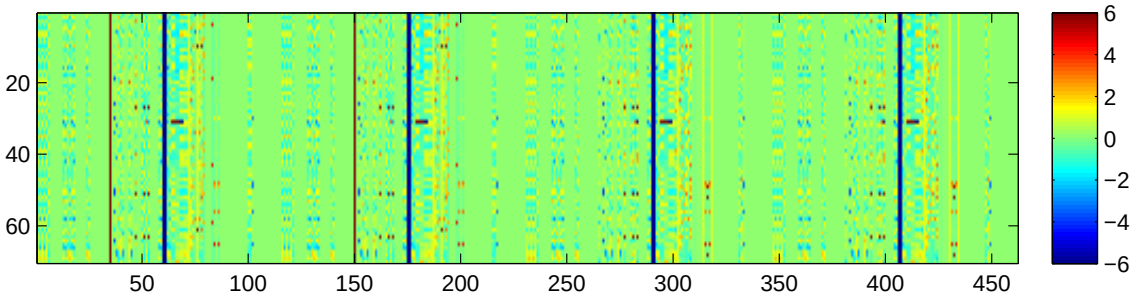
In this chapter, we present two new signatures, called EigenNSS and FisherNSS, both motivated by techniques used in facial recognition applications to achieve dimensionality reduction in NSSs. The new signatures transform the NSSs to lower dimensions without losing useful information. Principal component analysis (PCA) is used to transform the NSS into a new signature called EigenNSS, whereas Fisher's linear discriminant (FLD) analysis [84] is used to generate another signature type called FisherNSS. FisherNSS strives to maximise the ratio of the between-class scatter to the within-class scatter for better classification results. We report a detailed comparison of performance of EigenNSS and FisherNSS with data gathered from real-world networks. To our knowledge, this is the first time network soft failure signatures have been created using the proposed techniques.



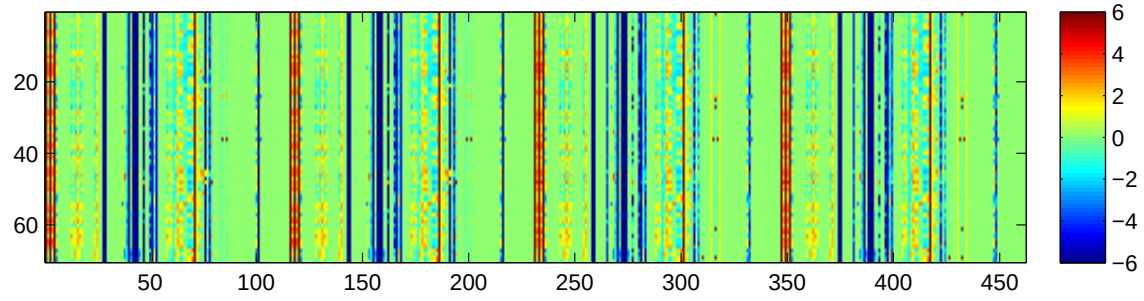
(a) Healthy UD



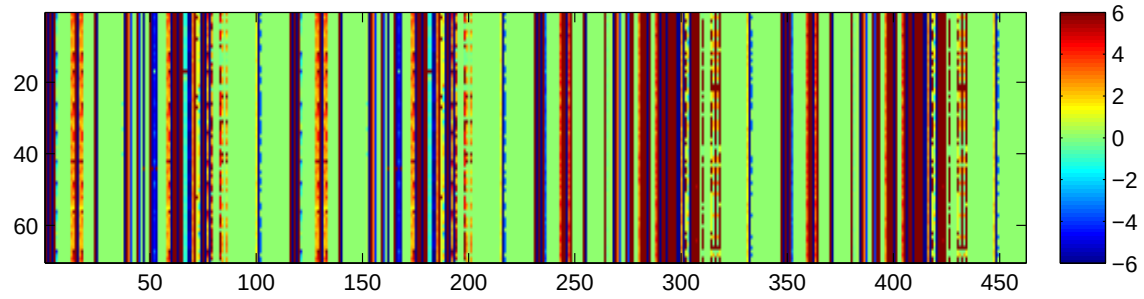
(b) Disabled SACK error



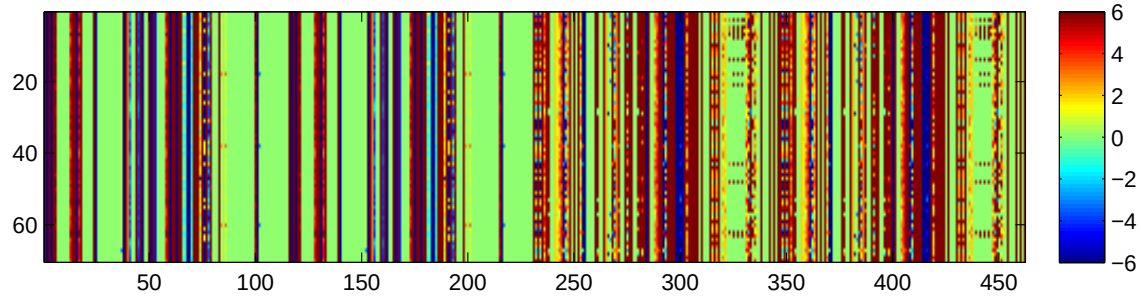
(c) TCP timestamps error



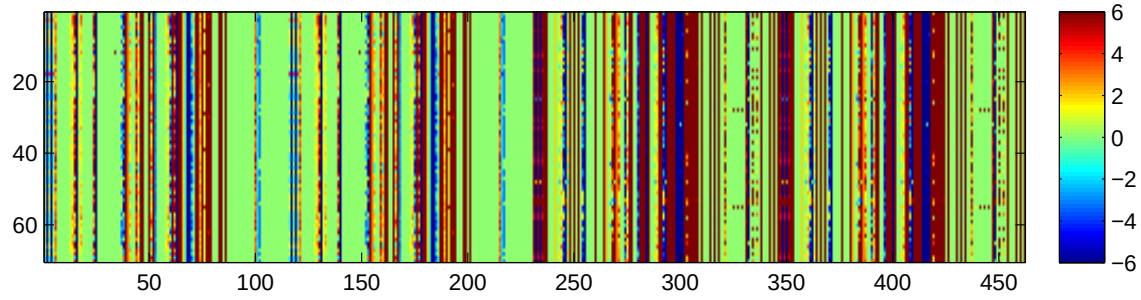
(d) Window scaling error



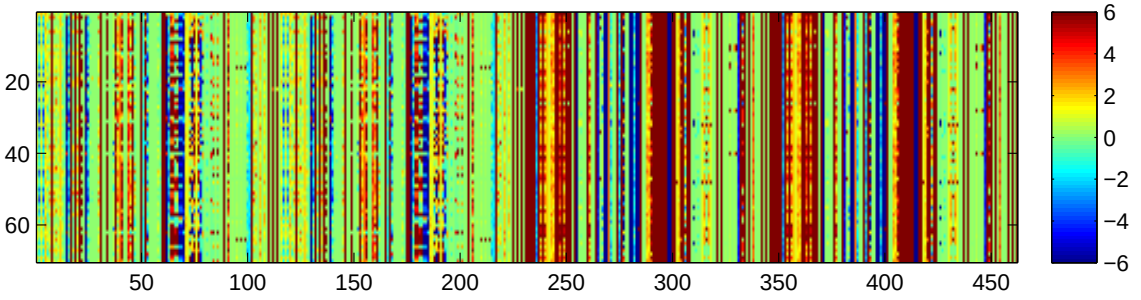
(e) Link-UD speed mismatch



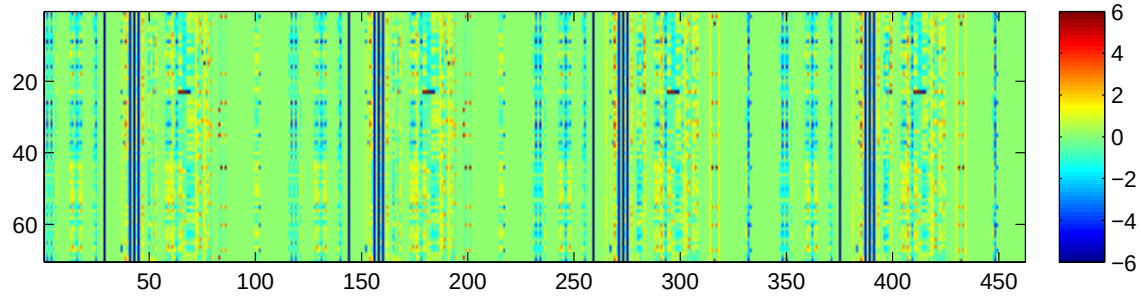
(f) Duplex mismatch



(g) UD firewall causing delay



(h) UD firewall causing packet loss



(i) Insufficient read and write buffers

Figure 6.1: Comparison of NSSs for common UD soft failures. Here, the x-axis (columns) of each figure represent 460 features of the 70 NSSs shown on the y-axis (rows). Each of the features has been normalised and scaled $[-6, 6]$ and each of the features is represented by a coloured vertical line projecting the scaled feature value to RGB space. Although, the NSSs of the same fault show minor variations due to stochastic nature of the networks, groupings of 70 NSSs of the same fault clearly shows that the NSSs are consistent in uniquely characterising a fault and can be used as the "fingerprint" for a diagnosis.

6.1 Operational overview

Deployment

The IAND-*h* diagnostic server is deployed as an application on the access router, as shown previously in Figure 1.4. The complexities that can affect the uniformity of the captured packet traces can be eliminated by narrowing down the path to the UD into an access link. The packet trace capturing mechanism introduced in Chapter 3 is used to capture the sample data required for diagnosis and the capture process is shown in Figure 6.2. Firstly, upon initiation by the end-user, the modules needed for file transfers and packet captures are loaded. Two TCP-based data transfers of a fixed size of 20 MB file (an upload and download) are conducted serially between the UD and server. The captured TCP packet traces are sent to the feature extraction module, where they are analysed and features are extracted to obtain statistical attributes (features), known as "raw" signatures.

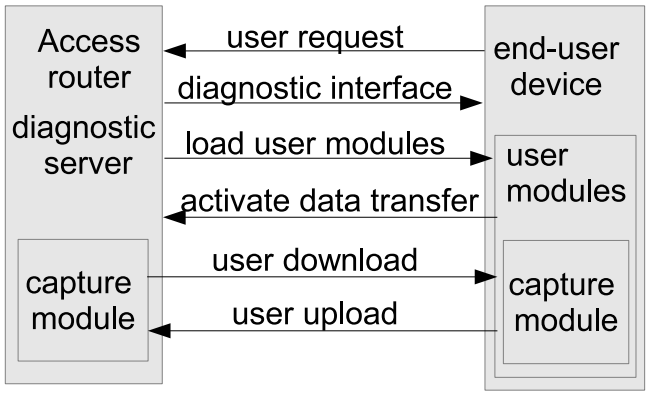


Figure 6.2: Deployment of the IAND-*h* system over the access network.

Operation

Figure 6.3 shows the operational stages of the diagnostic system. There are three main stages:

1. Training stage,

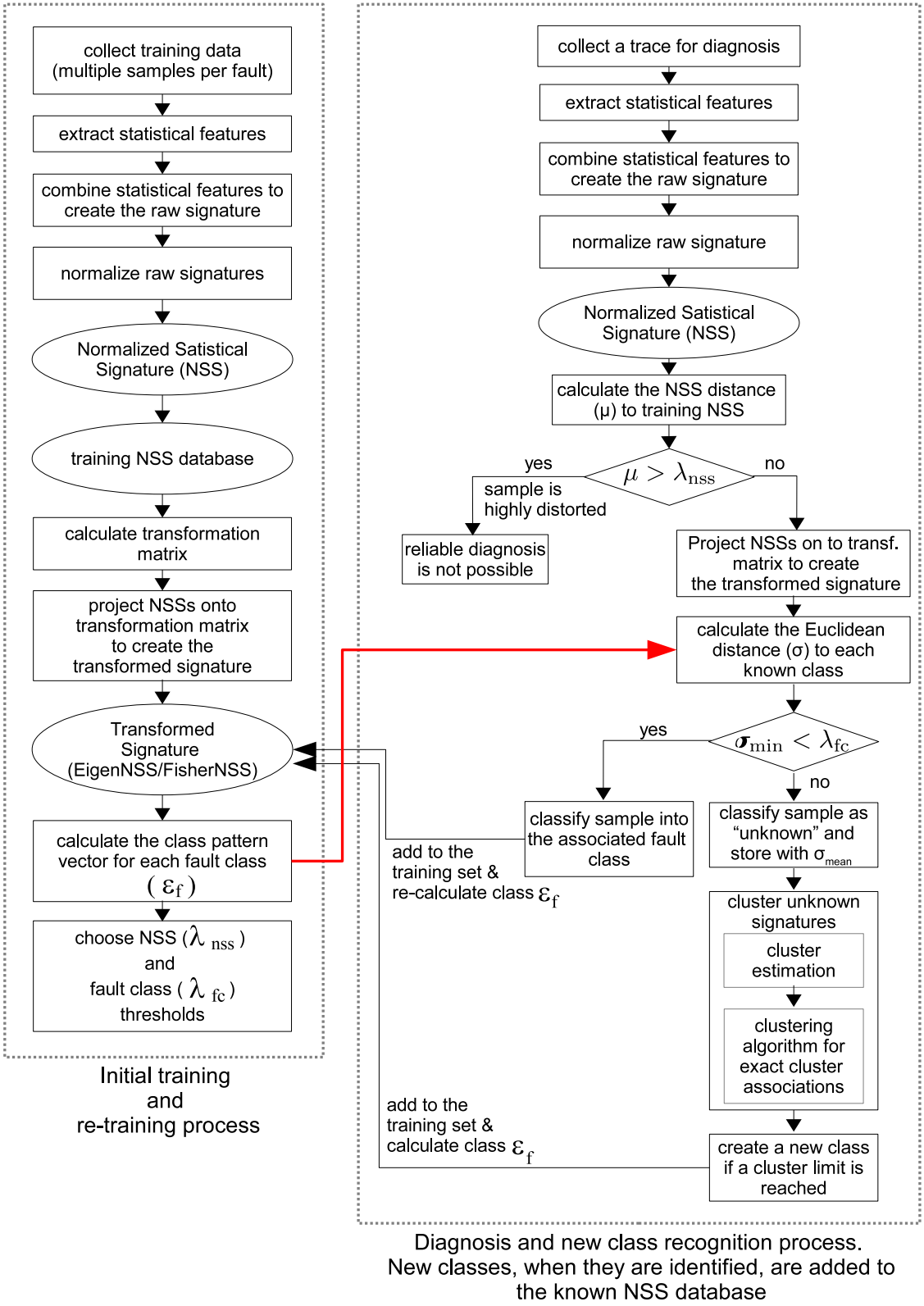


Figure 6.3: Operational overview of the IAND-*h* system.

2. Diagnosis stage,

3. New class recognition stage.

During the training stage, the packet traces are collected from UDs with known faults induced. Statistical attributes of the traces collected at client and server are then extracted and serially concatenated to form the *raw signature*. These raw signatures are then given a class label to indicate the specific fault they characterise. The raw signature is then normalised against the healthy performance baseline to create the NSS. We define a healthy UD similarly to Chapter 1.

The NSSs generated from multiple classes of faults are stored in a database and are used to calculate the *transformation matrices* of the database. The NSSs are projected onto either one of the transformation matrices to create the transformed signatures, EigenNSS and FisherNSS respectively. The transformed signatures of each class are used to calculate the pattern vector (ϵ_f) of that particular class. Finally, two thresholds are chosen for the system:

1. λ_{nss} for determining if a particular signature is valid, and

2. λ_{fc} for determining class associations.

Once the system is trained, UDs that are suspected to be faulty can be diagnosed by collecting packet traces and sending them through the same feature extraction and NSS generation process. The EigenNSS and FisherNSS can then be generated by projecting the NSSs onto either one of the transformation matrices created during the training stage. These transformed signatures are then used to calculate the pattern vector (ϵ) and Euclidean distance (σ)(ED) of the pattern vector from each known class. The NSS can also be used to calculate the square distance (μ) from the training NSS data set. Finally, depending on the minimum ED (σ_{min}) and square distance (μ), one of the three possible outcomes is decided by the system as follows:

if $\mu > \lambda_{\text{nss}}$ **then**

The sample is highly distorted.

Reliable diagnosis is not possible.

else

if $\sigma_{\text{min}} < \lambda_{\text{fc}}$ **then**

The sample is classified to the class associated
with the minimum distance. The sample is then
added to the training NSS database
and the ϵ_f of the class is recalculated
to include the new sample.

else

The sample contains a valid signature, but
it does not belong to any of the
known classes. Hence, the sample is
classified as an unknown class.

end if

end if

The above is called as Euclidean distance (ED) classification

When the IAND-*h* system detects a predetermined number of unknown signatures, a new class recognition phase begins. This predetermined number is usually defined to be twice the minimum threshold size at which a given cluster is accepted as a new class. These unknown signatures are first stored in a separate database with their pattern vectors. These signatures are then sent through a cluster estimation algorithm [249] which determines (i) if the data set has samples that can be clustered within the λ_{fc} bound, and (ii) the number of clusters that can be created. These clusters will be matched with their exact cluster memberships with the assistance of a clustering algorithm, which uses a fuzzy C-means clustering technique with iterative optimisation [250]. If any of the clusters reach the

minimum threshold size, they are considered as a new class and are added to the training database. The transformed signatures of the new class are sent to the class pattern vector calculation, while its NSSs are sent to the classifier training database. Although new classes can be added by calculating the class pattern vectors, the system can be re-trained in a short time when it is not performing any diagnostics if the training database is updated with the new NSSs. Re-training adds the new classes to supervised classifier and improves the final accuracy of the system. The IAND-*h* system also prompts administrators that a new fault class has been detected, and after investigative analysis of its actual root cause, a class label can be created.

6.2 EigenNSS: NSS transformation using principal components

To overcome the challenges of having high dimensionality in NSSs, we searched for a way to emphasise the significant global features that contain a maximum amount of information. Dimensionality reduction can be achieved by firstly finding the principal components (i.e eigenvectors of the covariance matrix) of the distribution of NSSs. Then, these eigenvectors can be ordered according to the amount of variations among the NSSs. Finally, the NSSs are projected onto a selected number (D) of eigenvectors that account for the highest variation. These projections are called "EigenNSS", and represent most of the information from the original samples in a D-dimensional space.

6.2.1 Generating EigenNSS

The following sub-section provides details of the EigenNSS generation and calculation process. Consider a training NSS set of $\mathbf{x}_1$, $\mathbf{x}_2$, ..., $\mathbf{x}_p$ where x is a m-

dimensional feature vector. For example, the set of NSSs shown in Figure 6.1 has 70 samples for each of the 10 classes (p = 70 x 10 = 700), in which each has 460 features (m = 460). The mean of the NSSs can be found by

$$\Phi = \frac{1}{p}\sum_{k=1}^{p}\mathbf{x}_k$$

The training NSSs are then mean-centred by

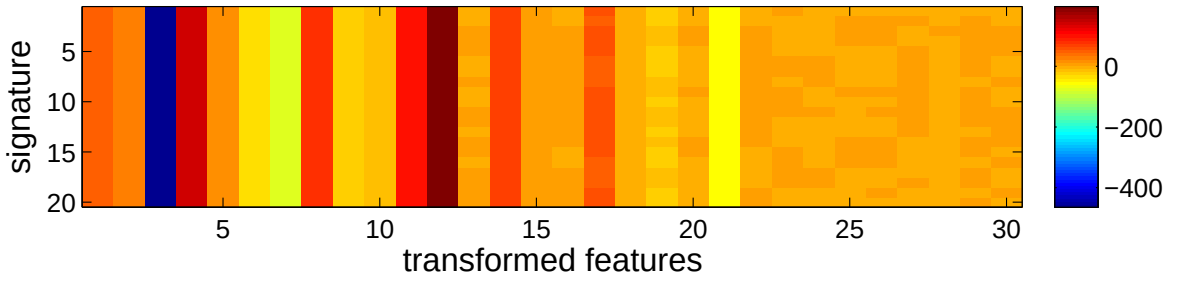
$$\bar{\mathbf{x}}_i = \mathbf{x}_i - \Phi$$

where $\bar{\mathbf{x}}_i$ is the mean-centred m-dimensional feature vector of the i^{th} instance. The resultant matrix $\Delta = \{\bar{\mathbf{x}}_1, \bar{\mathbf{x}}_2, ..., \bar{\mathbf{x}}_p\}$ has the dimensions of m x n and is subjected to principal component analysis where the eigenvectors, $\mathbf{v}_i$ and the corresponding eigenvalues n_i of Δ are determined by solving a well-known singular value decomposition (SVD) problem. However, only a pre-determined D number of eigenvectors v, that are arranged from the highest eigenvalue n, are selected to create the eigen matrix which has the dimensions of m x D. Finally, EigenNSSs are created by projecting the mean centred NSS matrix, Δ, onto the eigen matrix, Ψ_{PCA} as

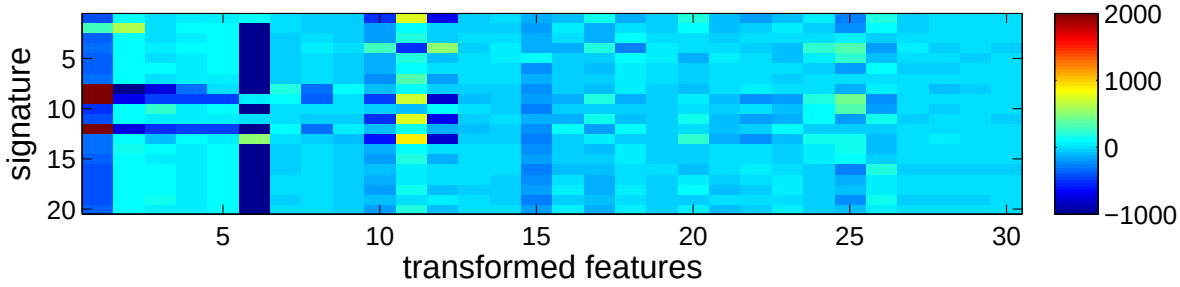
$$\Theta_{\text{PCA}} = {\Psi_{\text{PCA}}}^T \Delta$$

where $\Theta_{\text{PCA}} = \{\mathbf{e}_1, \mathbf{e}_2, ..., \mathbf{e}_p\}$ and $\mathbf{e}$ is a D-dimensional EigenNSS.

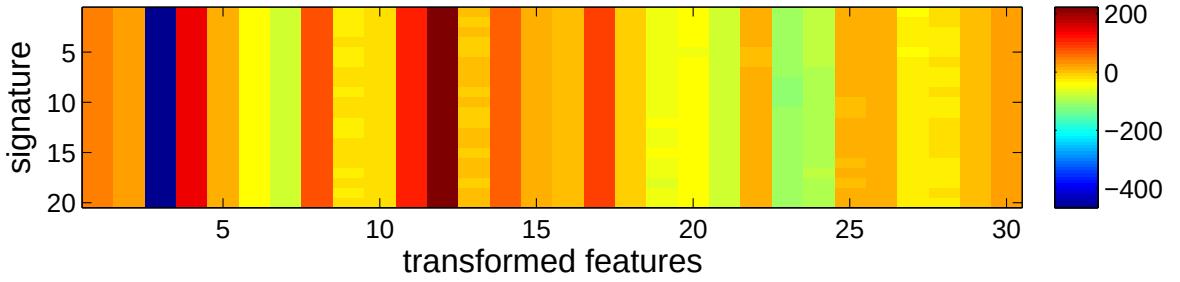
Figure 6.4 shows the comparison of EigenNSSs for various types of common UD faults with D=30. Note that the figure shows only 25 signature samples per class for clarity. As shown in the figure, the dimensionality of NSSs has been reduced while preserving the most important information for class separation. This is clearly visible, as the EigenNSS shows significant differences between classes compared to the NSSs. However, it can be seen that the EigenNSS samples within the same classes can vary due to the inconsistent nature of the network links.



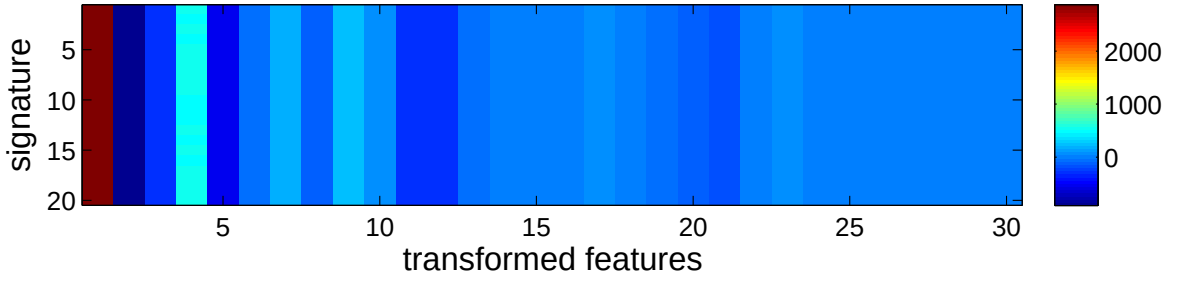
(a) Healthy UD



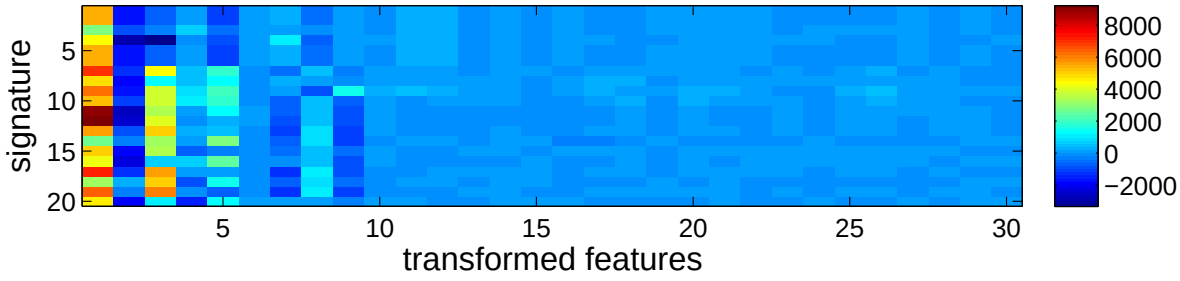
(b) Disabled SACK error



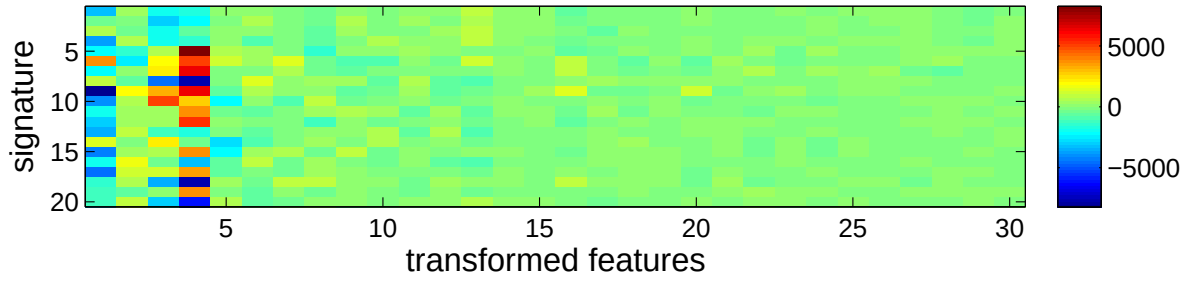
(c) TCP timestamps error



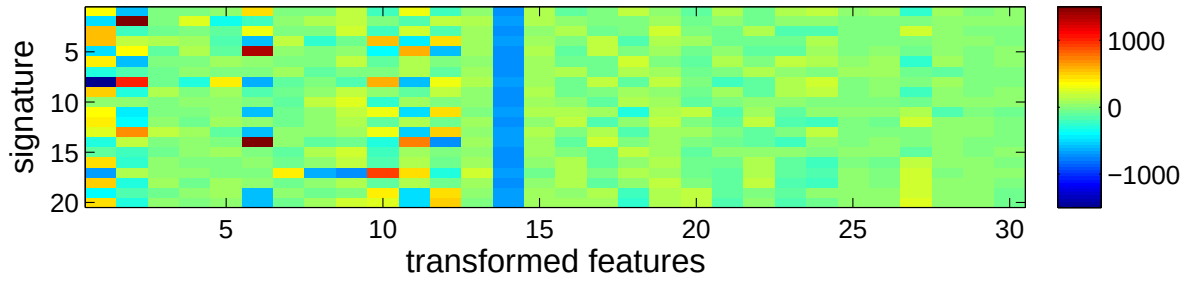
(d) Window scaling error



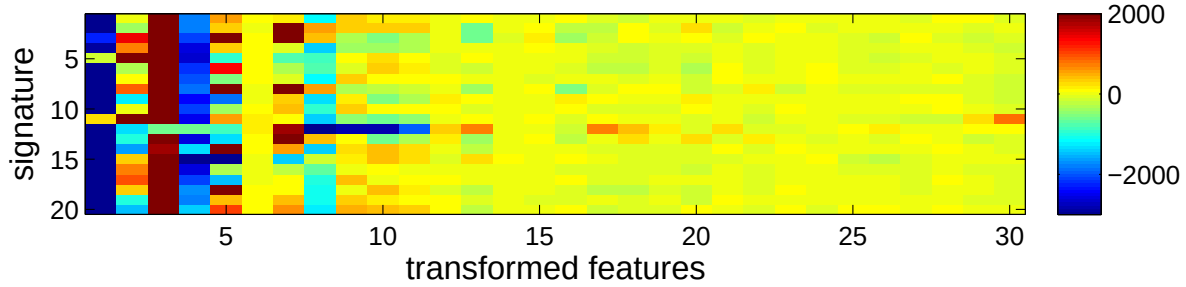
(e) Duplex mismatch Level 1



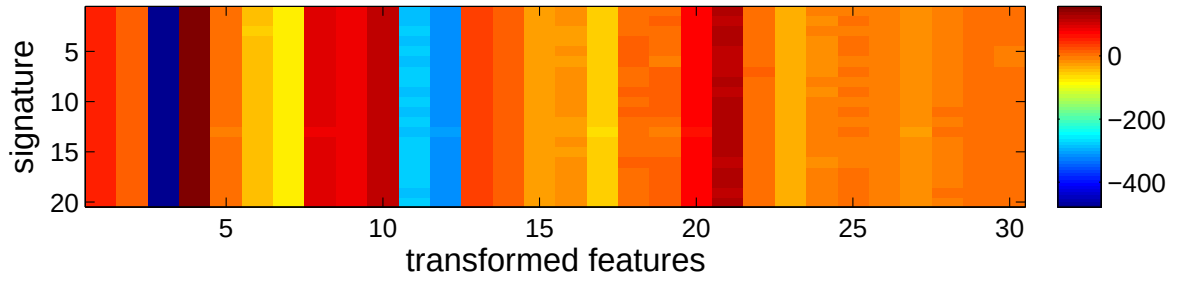
(f) Duplex mismatch Level 2



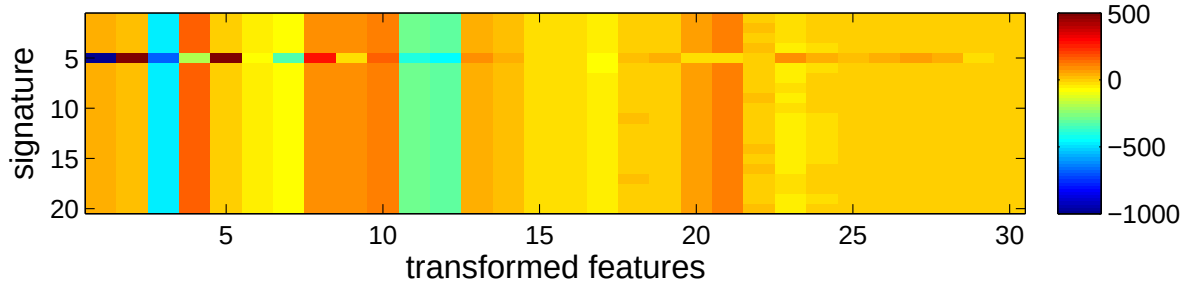
(g) UD firewall causing delay



(h) UD firewall causing packet loss



(i) Simultaneous insufficient read-write buffer



(j) Insufficient read buffer

Figure 6.4: Comparison of EigenNSSs for common UD soft failures. Here, groups of 20 signature samples (y-axis=1-20) per fault are shown to demonstrate the consistency of EigenNSS within a fault group.

6.3 FisherNSS: NSS transformation using Fisher's Linear Discriminant Analysis

As shown in the previous section, EigenNSS reduces the dimensionality of NSS. However, to account for the errors in data collection and the inconsistent nature of the networks, separation between the unwanted information is needed for a better classification outcome. This section shows the motivation and generation process of FisherNSS, a new transformed signature.

6.3.1 Fisher's Linear Discriminant

The inconsistent nature of the network affects the extracted features and may lead to poor fault detection. These inconsistencies (noise terms) are embedded inside the data, which makes it difficult to distinguish them from the actual information. As previously mentioned, Fisher's linear discriminant (FLD) aims to provide linear separability between classes in a data set, to facilitate correct fault diagnosis. This is achieved by using PCA to reduce the dimensions of the feature space, and then applying further reduction and linear separation with FLD.

FLD is an example of a *class-specific method*, that attempts to "shape" the scatter of feature values to facilitate reliable classification. To illustrate the benefits of a class-specific linear projection, we constructed a set of 10 2-dimensional (n=2) sample points. In Figure 6.5, a comparison of both PCA and FLD for a two-class problem is shown by projecting the constructed sample points from $2-D$ down to $1-D$ respectively. Comparing the projections, PCA smears the classes together so that they are no longer linearly separable in the projected space. Although the total scatter of FLD is smaller than that of PCA, FLD achieves greater between-class scatter, and consequently, results in better classification.

6.3.2 Generating FisherNSS

The following sub-section illustrates how the FisherNSS is generated, including the calculation process. Assuming we have a training EigenNSS set, Θ_{PCA} of $\mathbf{e}_1, \mathbf{e}_2, ..., \mathbf{e}_p$, where $\mathbf{e}$ is a D-dimensional transformed feature vector. For example, the set of EigenNSS shown in Figure 6.4 has N=25 samples for each of the 10 classes (p = 25 x 10 = 250), in which each has D=30 features. The mean of the entire EigenNSS set can be found by

$$\Phi_{\text{PCA}} = \frac{1}{p} \sum_{k=1}^{p} \mathbf{e}_k$$

The mean of the EigenNSS for the i^{th} class can be found by

$$\Phi_i = \frac{1}{N_i} \sum_{\mathbf{e}_k \in E_i} \mathbf{e}_k$$

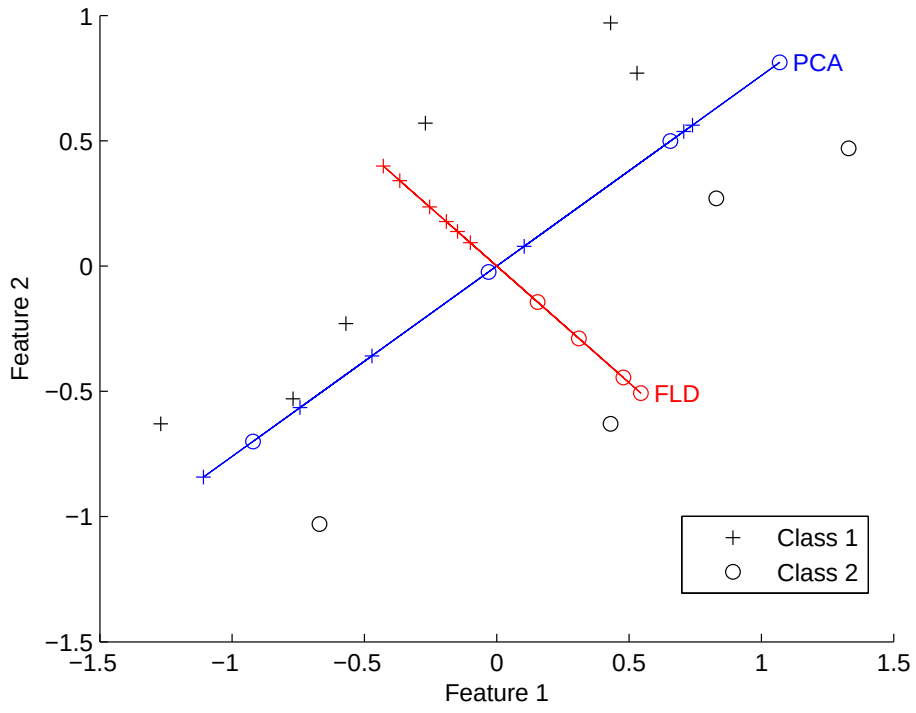


Figure 6.5: A comparison of principal component analysis (PCA) and Fisher's linear discriminant (FLD) for a two-class problem.

where Φ_i is the mean EigenNSS of class E_i and N_i is the number of samples in class E_i. The between-class scatter matrix can then be computed by

$$S_B = \sum_{i=1}^{c} (\Phi_i - \Phi_{\text{PCA}})(\Phi_i - \Phi_{\text{PCA}})^T$$

and the within-class scatter matrix by

$$S_W = \sum_{i=1}^{c} \sum_{\mathbf{e}_k \in E_i} (\mathbf{e}_k - \Phi_i)(\mathbf{e}_k - \Phi_i)^T$$

The optimal projection W_{opt} is chosen as the matrix with orthonormal columns (e.g. eigenvectors, w), which maximises the ratio of the determinant of the between-class scatter matrix to the determinant of the within-class scatter matrix, i.e.,

$$W_{\text{opt}} = \arg\max \frac{|W^T S_B W|}{|W^T S_W W|}$$

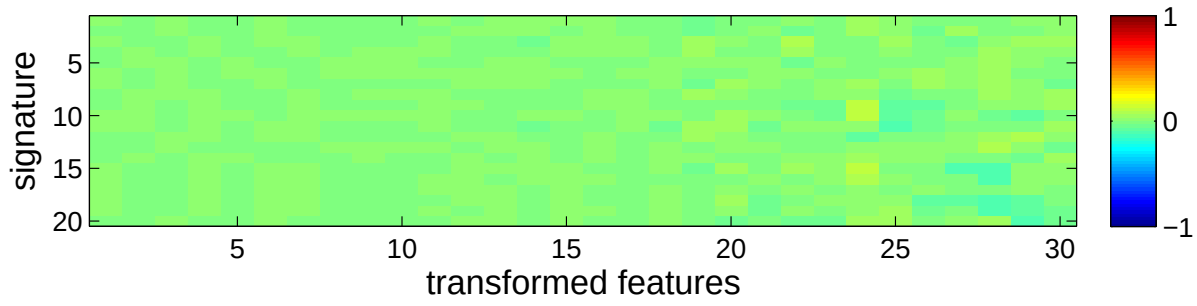
.

However, only a pre-determined D number of eigenvectors that are arranged from the highest eigenvalue are selected to create the Fisher matrix, which has the dimensions of D x D. Finally, FisherNSSs are created by projecting the EigenNSS matrix, Θ_{PCA} onto the Fisher matrix, Ψ_{FLD} as

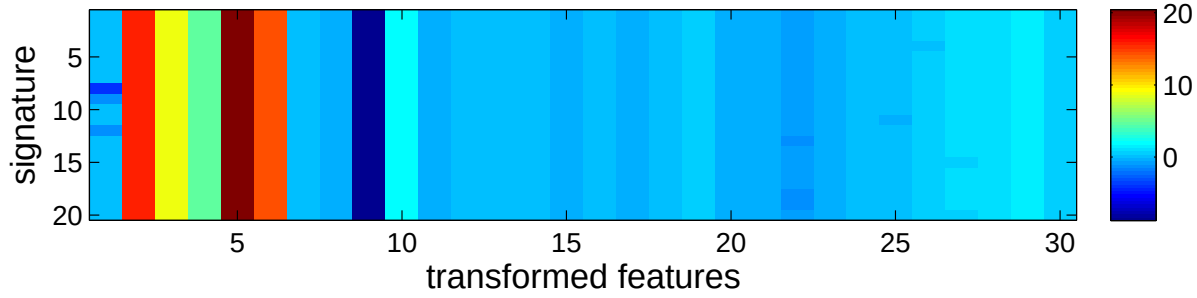
$$\Theta_{\text{FLD}} = {\Psi_{\text{FLD}}}^T \Theta_{\text{PCA}}$$

where $\Theta_{\text{FLD}} = \{\mathbf{f}_1, \mathbf{f}_2, ..., \mathbf{f}_p\}$ and $\mathbf{f}$ is an D-dimensional FisherNSS.

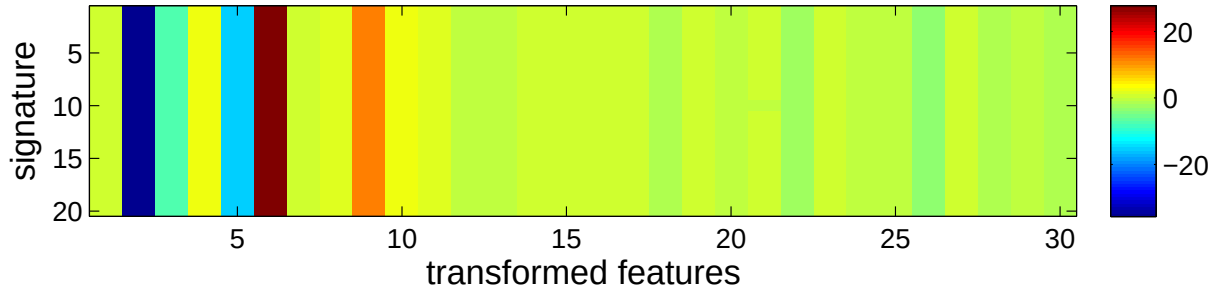
Figure 6.6 shows the FisherNSSs for various types of common UD faults with D=30. Similarly, only 25 signature samples per class are shown for clarity. As is evident from the figure, the signature samples in each class exhibit very little variation and appear to reduce the effects of network inconsistencies on the extracted signatures.



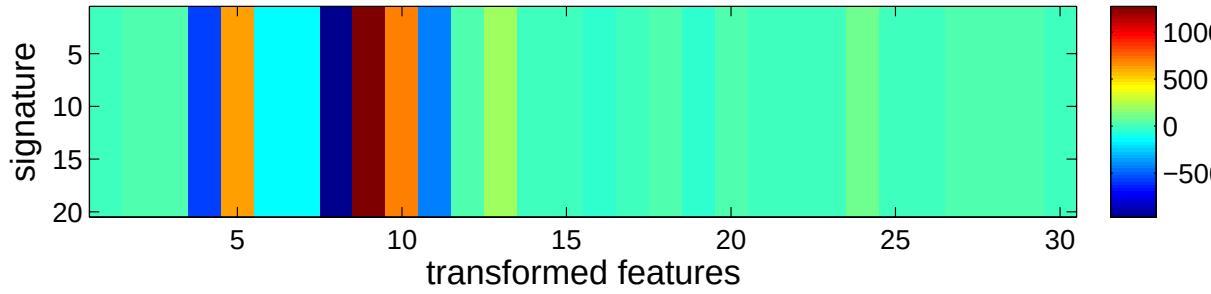
(a) Healthy UD



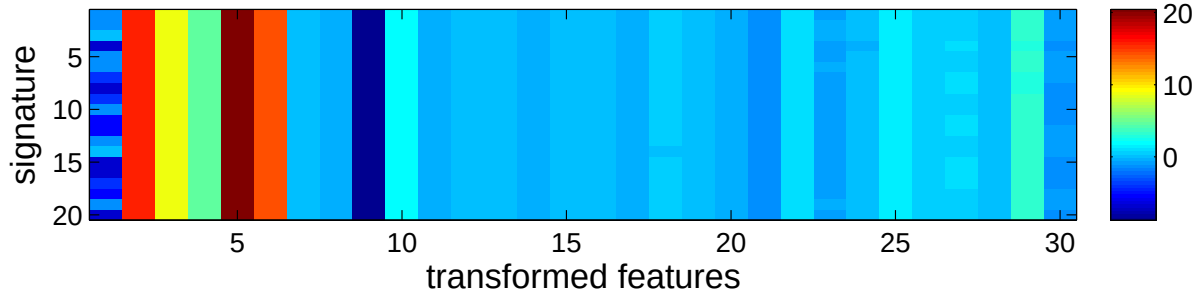
(b) Disabled SACK error



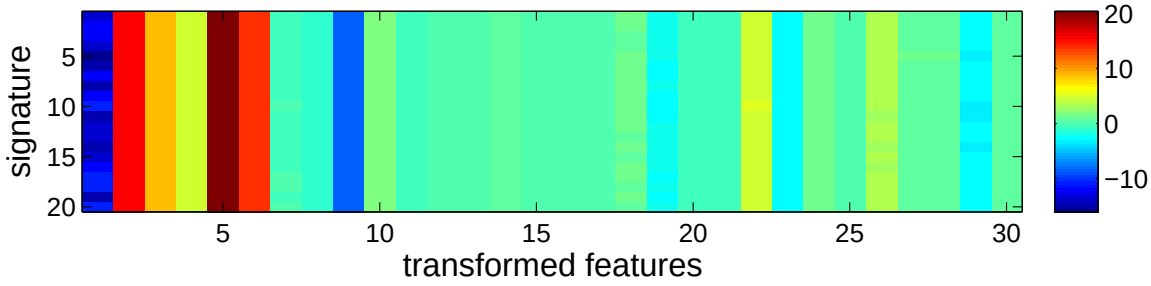
(c) TCP timestamps error



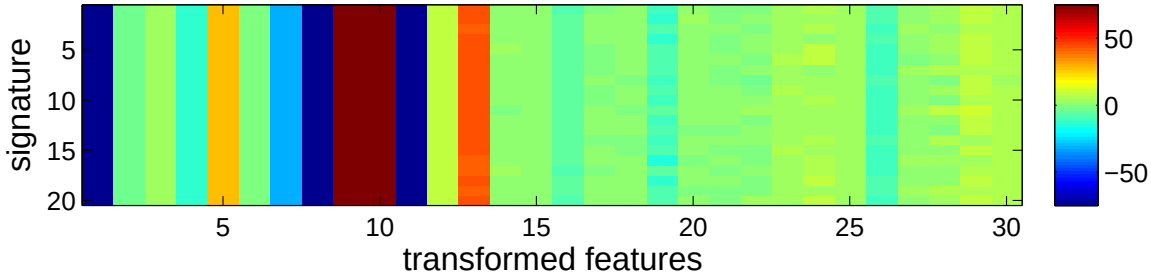
(d) Window scaling error



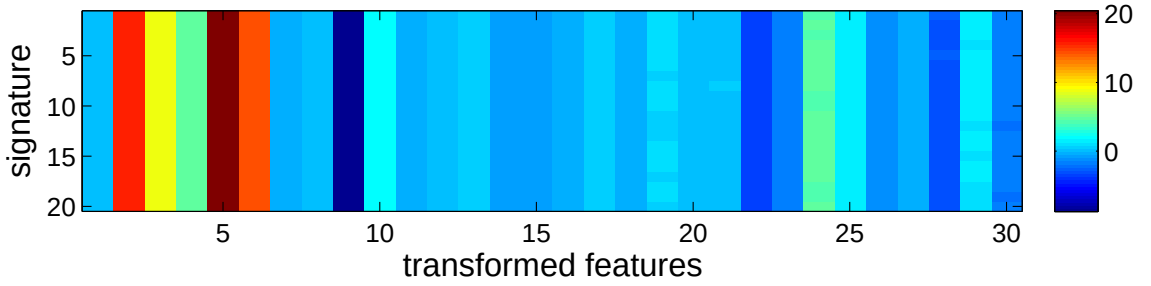
(e) Duplex mismatch Level 1



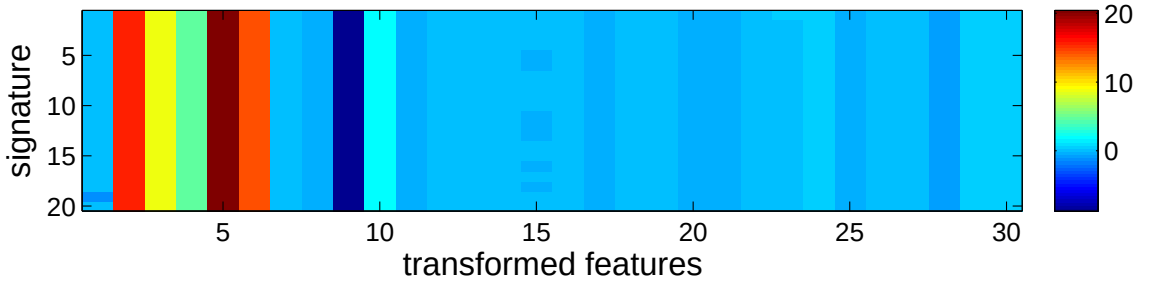
(f) Duplex mismatch Level 2



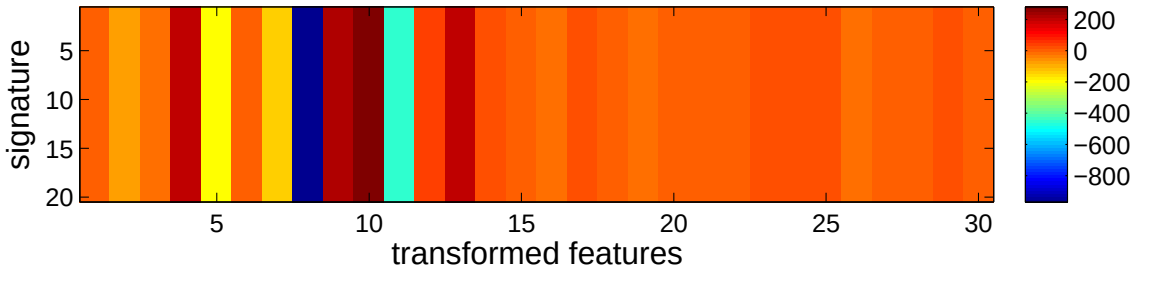
(g) UD firewall causing delay



(h) UD firewall causing packet loss



(i) Simultaneous insufficient read-write buffer



(j) Insufficient read buffer

Figure 6.6: Comparison of FisherNSSs for common UD soft failures.Here, groups of 20 signature (y-axis=1-20) samples per fault are shown to demonstrate the consistency of FisherNSSs within a fault group.

6.4 Training and diagnosis using transformed signatures

6.4.1 Training

The following sub-section shows the IAND-*h* system's training process using EigenNSS and FisherNSS respectively, where the pattern vectors (ϵ_f) of each class are calculated. We assume the EigenNSS matrix, Θ_{PCA} and the FisherNSS matrix, Θ_{FLD} contain n signature samples from multiple classes (faults, f). The D-dimensional class pattern vector, ϵ_f is calculated by averaging the reduced signatures as

$$\epsilon_{f,\text{PCA}} = \frac{1}{n}\sum_{k=1}^{n}\mathbf{e}_k$$

$$\epsilon_{f,\text{FLD}} = \frac{1}{n}\sum_{k=1}^{n}\mathbf{f}_k$$

where these pattern vectors, ϵ_f are used during the diagnosis stage to calculate the distance between any given unknown (test) signatures. These distances are used to determine the best class fit of the unknown signatures.

6.4.2 Diagnosis

Known faults

In this sub-section, the classification criteria required for the diagnosis of known faults are shown. The simplest method for determining the class association of a test sample is to calculate the ED, σ, between **e** and **f** respectively with each of the class pattern vectors as

$$\sigma_{f,\text{PCA}} = ||\mathbf{e} - \epsilon_{f,\text{PCA}}||^2$$

$$\sigma_{f,\text{FLD}} = ||\mathbf{f} - \epsilon_{f,\text{FLD}}||^2$$

The test sample is classified and associated with a class, f, when its respective minimum ED, σ_{min}, is below the chosen fault class threshold, λ_{fc}. Whilst in previous sections we have used more advance classification methods such as SVMs, here we have chosen to use a simpler form of a classifier to highlight the benefits of the new transformed signatures. However, any type of ML-based classification algorithm could be used to further enhance the performance.

Due to errors in data collection and the inconsistent nature of networks, some of the collected packet traces can be distorted. This may lead to a false detection, where the erroneous NSSs generated from these distorted packet traces are wrongly classified. Hence, we introduce another term for a more reliable outcome which is the squared distance, μ, between the NSS feature vector of the test sample, y, and the mean of the training NSSs, Φ.

$$\mu = ||y - \Phi||^2$$

The test sample is considered valid only if the minimum squared distance is less than the chosen NSS threshold, λ_{nss}.

Depending on the minimum values of σ_f and μ, the outcome of the diagnosis process is determined following the criteria previously described in Figure 6.3.

Unknown faults

Here, we present the new class recognition process for the diagnosis of unknown faults. A test signature sample is classified as "unknown" if it contains a valid signature but does not belong to any of the known classes. These unknown samples are sent through a cluster estimation algorithm to determine the number of clusters that can be created, and whether or not the samples can be clustered within the λ_{fc} bound.

Assume that we have a set of n unknown signature samples $\{x_1, x_2, ..., x_n\}$ in the database. We consider each of the unknown samples as a potential cluster centre and its respective measured potential, P_i, as a function of its distances to all the other unknown samples. The measure of potential for the i^{th} unknown samples is given as

$$P_i = \sum_{j=1}^{n} \mathbf{e}^{-\alpha||x_i - x_j||^2}$$

where

$$\alpha = \frac{4}{r_a^2}$$

and r_a is a positive constant, corresponding to the radius defining a neighbourhood, where unknown samples outside this radius have limited influence on the potential. After the potential of every unknown sample has been computed, we choose the unknown sample with the highest measured potential as the first cluster centre.

Let x_1^* be the location of the first cluster centre and P_1^* be its measured potential value. The potential of each unknown sample x_i is then revised by subtracting an amount of potential as a function of its distance from the first cluster centre. The revised potential of the i^{th} unknown sample can be calculated as

$$P_i = P_i - P_1^* \mathbf{e}^{-\beta||x_i - x_1^*||^2}$$

where

$$\beta = \frac{4}{r_b^2}$$

and r_b is a positive constant, corresponding to the radius defining the neighbourhood that will have measurable reductions in potential. The constant r_b is set to be somewhat greater than r_a, to avoid cluster centres being too closely spaced together. A good choice of values is $r_b = 1.5r_a$. The unknown samples near the first cluster centre will have greatly reduced potential, and are unlikely to be the

source of the next cluster centre. Therefore, the unknown sample with the highest remaining potential is selected to be the second cluster centre.

In general, the process of acquiring new cluster centres, x_k^* is repeated using the general formula for revising potentials as

$$P_i = P_i - P_k^* \mathbf{e}^{-||x_i - x_k^*||^2}$$

following these criteria:

if $P_k^* > \bar{\epsilon}\ P_1^*$ **then**

Accept x_k^* as a cluster centre and continue.

else if $P_k^* < \underline{\epsilon} P_1^*$ **then**

Reject x_k^* and end process.

else

Let d_{min} = shortest distances between x_k^* and

all previously found cluster centres.

if $\frac{d_{min}}{r_a} + \frac{P_k^*}{P_1^*} \geq 1$ **then**

Accept x_k^* as a cluster centre and continue.

else

Reject x_k^* and set the potential at x_k^* to 0.

Select the unknown sample with the

next highest potential as the new x_k^*

and re-test.

end if

end if

Here $\bar{\epsilon}$ represents the threshold for the potential above which we will definitely accept the unknown samples as cluster centres, whereas $\underline{\epsilon}$ represents a threshold below which we will definitely reject the unknown samples.

Once the cluster data are obtained, they are sent to a clustering algorithm

to determine the exact cluster membership. This algorithm uses fuzzy C-means clustering with iterative optimisation that minimises the cost function

$$J = \sum_{k=1}^{n} \sum_{i=1}^{c} \mu_{ik}^{m} ||x_k - v_i||^2$$

where n is the number of unknown test samples, c is the number of clusters (obtained from cluster estimation), x_k is the k^{th} unknown sample, c_i is the i^{th} cluster centre, μ_{ik} is the degree of membership of the k^{th} sample in the i^{th} cluster, and m is a constant greater than 1 (typically m = 2). The degree of membership, μ_{ik}, is defined by

$$\mu_{ik} = \frac{1}{\sum_{j=1}^{c} \left(\frac{||x_k - v_i||}{||x_k - v_j||}\right)^{2/(m-1)}}$$

FCM will converge to a solution for v_i that is either a local minimum or a saddle point of the cost function, J. The performance of the FCM solution depends strongly on the choice of the initial values used (e.g. the number of clusters, c, and the initial cluster centres, v_i), which are taken from the cluster estimation algorithm. Finally, the exact cluster membership can be computed by using the final iteration value of v_i.

6.5 Performance analysis

In this section, we evaluate the performance of the IAND-*h* system and analyse its diagnostic capability.

6.5.1 Data set

The test-beds introduced in Section 3.3 were used to collect data for evaluating the performance of new signatures and the IAND-*h* system. The proposed solution was deployed on test-bed 2 (see Figure 3.8), which used the live university

Fault	Description
CF1	Healthy
CF2	Disabled SACK error
CF3	Insufficient write buffer
CF4	Insufficient read buffer
CF5	Simultaneously insufficient read & write buffer
CF6	TCP timestamps are not working/in error
CF7	Window scaling error
CF8	Limited reordering threshold
CF9	Link-UD speed mismatch level 1
CF10	Link-UD speed mismatch level 1 & duplex mismatch
CF11	Link-UD speed mismatch level 2
CF12	Link-UD speed mismatch level 2 & duplex mismatch
CF13	UD firewall causing packet loss
CF14	UD firewall causing packet delay
CF15	Overloaded UD CPU
CF16	Overloaded UD memory
CF17	UD HDD i/o overloaded - faulty

Table 6.1: Key: List of faults

network with emulated UD faults (TB-2 dataset). Data collection was over a 3 weeks to capture more realistic data with fluctuations typically experienced with network utilisation. A total of 16 common UD faults that can affect network performance were emulated, as listed in Table 6.1. By including the "Healthy" UD case, a total of 17 classes were formed and used in this evaluation. Over the entire evaluation period, we collected 38055 traces from UDs emulating these 17 fault cases, each having equal numbers of samples. Data was also collected from UDs operating with eight different types of TCP, as they contribute to a variation in connection behaviour.

6.5.2 System performance

The IAND-*h* system was initially trained using only 13 classes as our evaluation needed to consider unknown faults. The faults CF2, CF8, CF10, and CF17 were kept as the unknown faults (refer to Table 6.1), and were introduced at random

intervals into the system during the testing stage. The data sets of the 13 classes were randomly divided into training and testing groups. We conducted the experiment over 8 iterative sessions and averaged the results in order to achieve statistical robustness.

The cluster threshold to add an unknown fault as a known fault was set to be equal to the number of per-class training samples used to initiate the system (e.g. if the system was initially trained with 50 samples per class, an unknown class was added as a known fault when its cluster membership reached 50). This cluster membership minimum threshold is a design choice and will dictate the confidence level in detecting the existence of a new fault. Having a large cluster membership before categorising a new fault improves the reliability of the system, but increases the time taken to offer users a valid diagnosis.

The proposed IAND-*h* system with FisherNSS and ED classifier (FisherNSS-

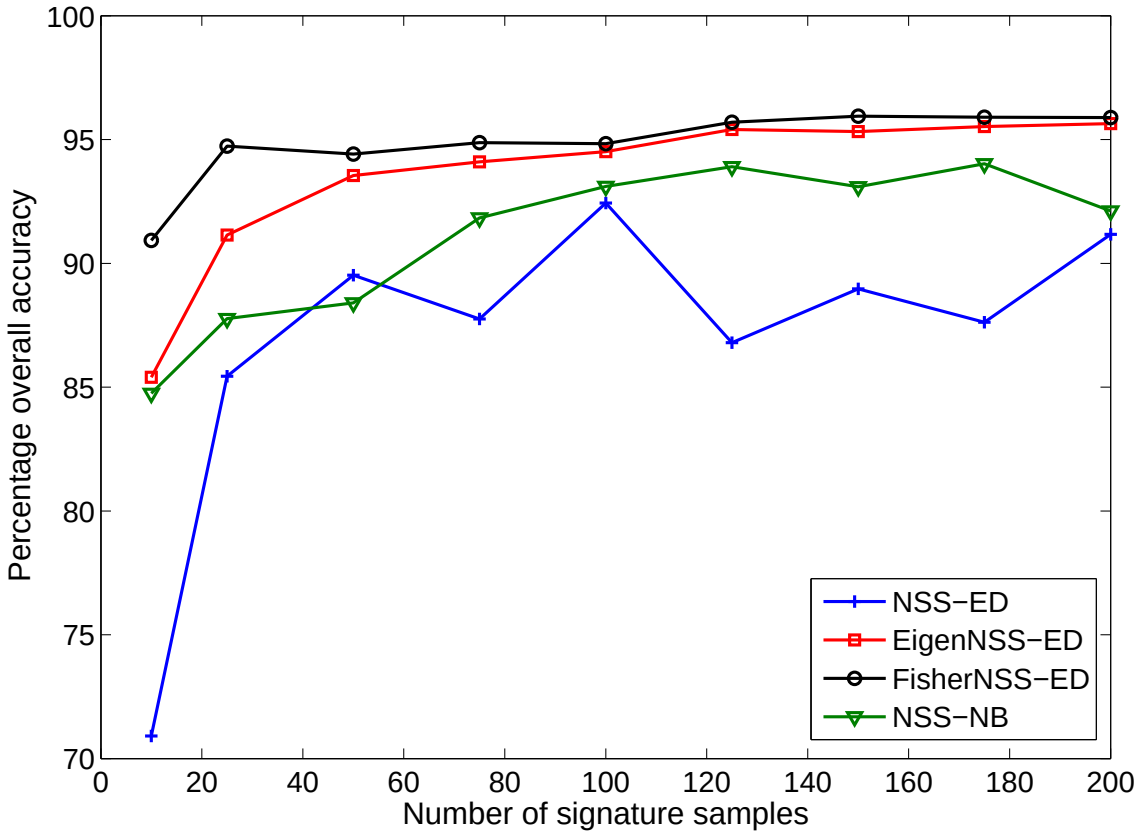


Figure 6.7: Comparison of overall accuracies of the IAND-*h* systems.

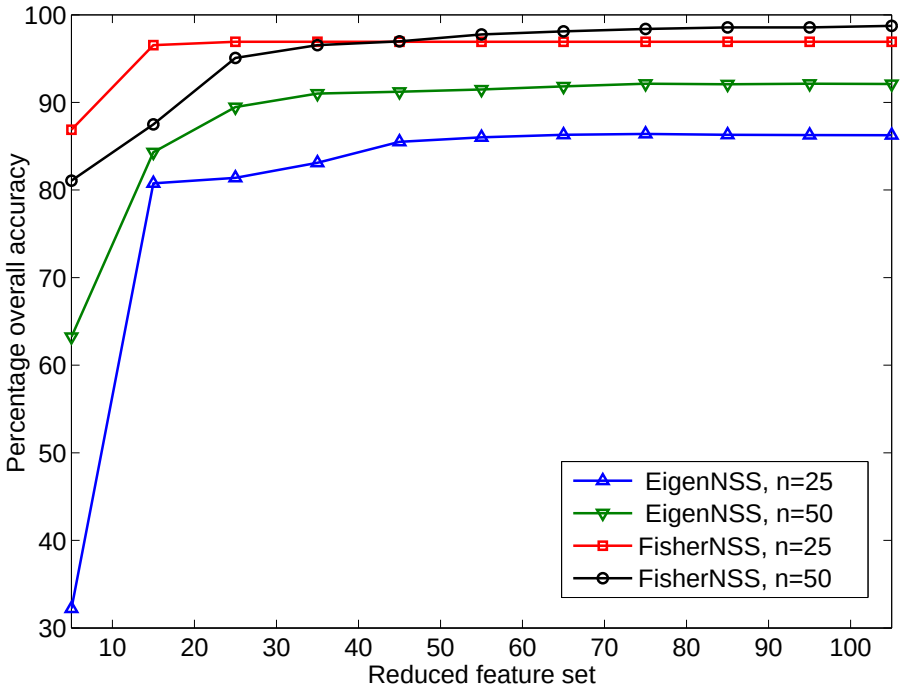


Figure 6.8: Overall accuracy of the systems against dimensionality (D) of the reduced signatures. Each graph represents a different per-class training dataset size.

ED) and the IAND-*h* system with EigenNSS and ED classifier (EigenNSS-ED) were tested against the IAND-*h* system with original NSS and ED classifier (NSS-ED) and a naive Bayes [144] multi-class classifier that used the original NSS data set (NSS-NB). Both FisherNSS-ED and EigenNSS-ED used dimensionality ($D = 60$). NSS-ED classifier is being used to compare the accuracy of IAND-*h* system against IAND-*k* which primarily uses NSS.

The system was trained with all 17 classes at the beginning using similar training and testing sets of data. Figure 6.7 compares the overall system accuracy between the 4 systems, where the x-axis represents the number of training samples used. For any given number of training samples, the figure shows that both EigenNSS-ED and FisherNSS-ED systems perform much better than the NSS-ED and NSS-NB systems. This is due to the "over-fitting" of classifiers used in the 460 feature NSSs, leading to degraded performance.

Figure 6.8 shows the overall accuracy of the FisherNSS-ED and EigenNSS-ED

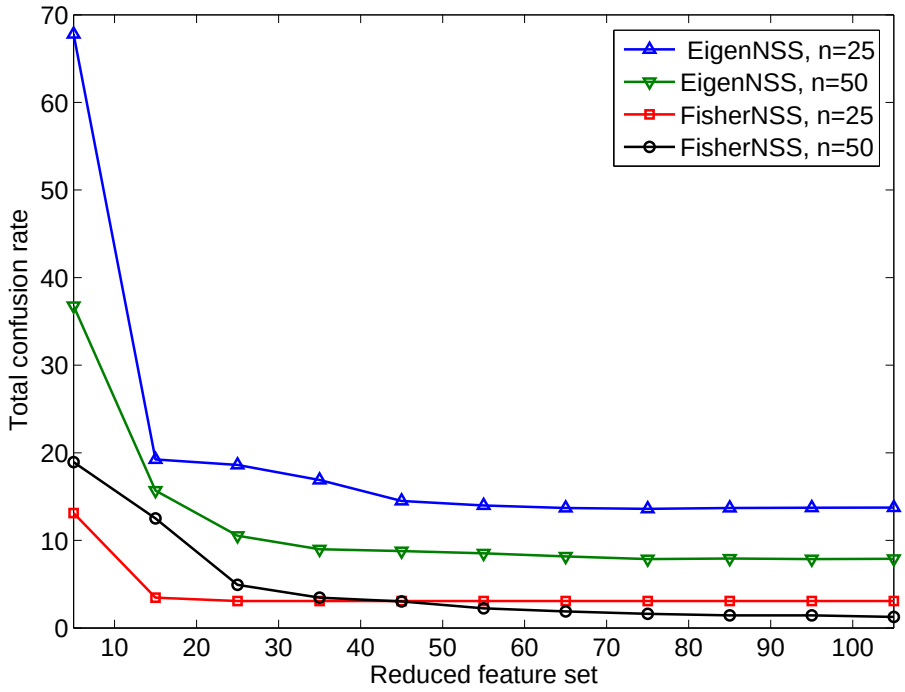


Figure 6.9: Overall confusion rate of the diagnostic systems against dimensionality (D) of the reduced signatures for different per-class training dataset size (n).

systems in recognising the testing samples which were previously unseen. Figure 6.9 shows the overall confusion rate of the systems, which indicates the ratio of wrongly classified samples to the total samples. For both figures, n represents the number of samples used for each class during training, and the x-axis shows the dimensionality (D) of the EigenNSS and FisherNSS respectively. From Figures 6.8 and 6.9, we see that the IAND-*h* system using FisherNSS achieved a higher overall accuracy than the EigenNSS for any D transformed signature features and n-values. Any additional features used to construct the transformed signature add only a marginal performance gain. Noting that the original NSS contained 460 features, these results show a successful dimensionality reduction of 96.74%. Figure 6.8 shows that as the number of samples used for training increases, the IAND-*h* system performance improves to a saturation limit.

The overall accuracy gap between both FisherNSS-ED and EigenNSS-ED systems becomes closer as the number of training samples increases. Most importantly, the FisherNSS-ED system performs better than the EigenNSS-ED system

when lower numbers of training samples are used. An overall accuracy of 80% and 20% confusion rate was achieved using EigenNSS with n=25 samples per class and D=15 reduced features for the 17 class system. Using the same n and D parameters with FisherNSS, an overall accuracy of 95% with 5% confusion rate was achieved. FisherNSS-ED is therefore deemed to be the better diagnostic system, due to the fact that it requires fewer training samples to obtain higher overall accuracy.

Table 6.2 summarises the performance of the 17 classes used in IAND-*h* system. The metrics used in the table have been previously defined in Section 5.1.2.

Table 6.2 also shows that all different types of faults can be uniquely identified independently with a high level of accuracy, as suggested by the high TPR and TNR. For example, faults CF6, CF7, CF12, CF13, CF14 and CF15 using both EigenNSS and FisherNSS have high TPRs of about 90% and above. Most of the faults show a low FPR and FNR, which indicates that the classifier has a low false detection rate.

Faults that were kept unknown to the systems, such as CF2, CF8, CF10 and CF17 have only a slightly smaller TPR compared to other faults. This shows that the systems have a high detection accuracy for unknown faults.

When EigenNSSs are used, some of the faults, such as CF1, CF2, CF3, CF5, CF8, CF9 and CF11 have relatively low TPRs which makes them less likely to be correctly classified as belonging to its respective class. However, in some fault cases, such as CF6, CF7, CF13 and CF15, EigenNSS has a better TPR than FisherNSS.

Fault	TPR		FPR		TNR		FNR	
	EigenNSS	FisherNSS	EigenNSS	FisherNSS	EigenNSS	FisherNSS	EigenNSS	FisherNSS
CF1	78.3	90.6	21.7	9.4	96.6	99.3	3.4	0.7
CF2	65.0	92.3	35.0	7.7	97.8	99.9	2.2	0.1
CF3	66.5	89.1	33.5	10.9	99.1	99.2	0.9	0.8
CF4	82.1	99.0	17.9	1.0	97.5	99.3	2.5	0.7
CF5	72.8	91.9	27.2	8.1	98.7	99.9	1.3	0.1
CF6	100	99.7	0	0.3	99.7	99.9	0.3	0.1
CF7	97.1	95.5	2.9	4.5	99.9	100.0	0.1	0.0
CF8	60.9	98.5	39.1	1.5	97.6	99.4	2.4	0.6
CF9	75.4	97.1	24.6	2.9	99.3	100.0	0.7	0.0
CF10	82.0	100	18.0	0	97.8	99.9	2.2	0.1
CF11	75.4	100	24.6	0	99.9	99.8	0.1	0.2
CF12	92.0	95.1	8.0	4.9	98.1	99.9	1.9	0.1
CF13	99.1	98.9	0.9	1.1	99.9	99.8	0.1	0.2
CF14	100	100	0	0	100	100.0	0	0.0
CF15	93.5	90.8	6.5	9.2	99.5	99.8	0.50	0.2
CF16	80.8	91.9	19.2	8.1	98.9	99.6	1.1	0.4
CF17	83.6	99.4	16.4	0.6	99.9	99.7	0.1	0.3

Table 6.2: Per-class estimation performance of the EigenNSS-ED and FisherNSS-ED systems at $D = 15$ and $n = 25$

Figures 6.10 and 6.11 show the confusion matrix of the EigenNSS-ED and FisherNSS-ED systems respectively. A confusion matrix is a typical form of visualisation to observe the performance of an algorithm, in this case, the multi-class classifiers. The rows represent target or expected (actual) classes and the columns represent the predicted classes. The diagonal elements of the matrix represent the correct classifications, whereas the other indices represent the incorrect instances. The ratio of each instance is colour-coded for clarity. In Figure 6.11, the FisherNSS-ED system successfully avoids large misclassification, satisfying a primary requirement of diagnostic system. However, Figure 6.10 shows a poorer performance for the EigenNSS-based system due to the greater chance of misclassification between classes.

Another important performance criterion for a system with an iterative training process is the time taken to train the system and evaluate a new sample (diagnosis). We included this in the experiment to compare how both the total train-

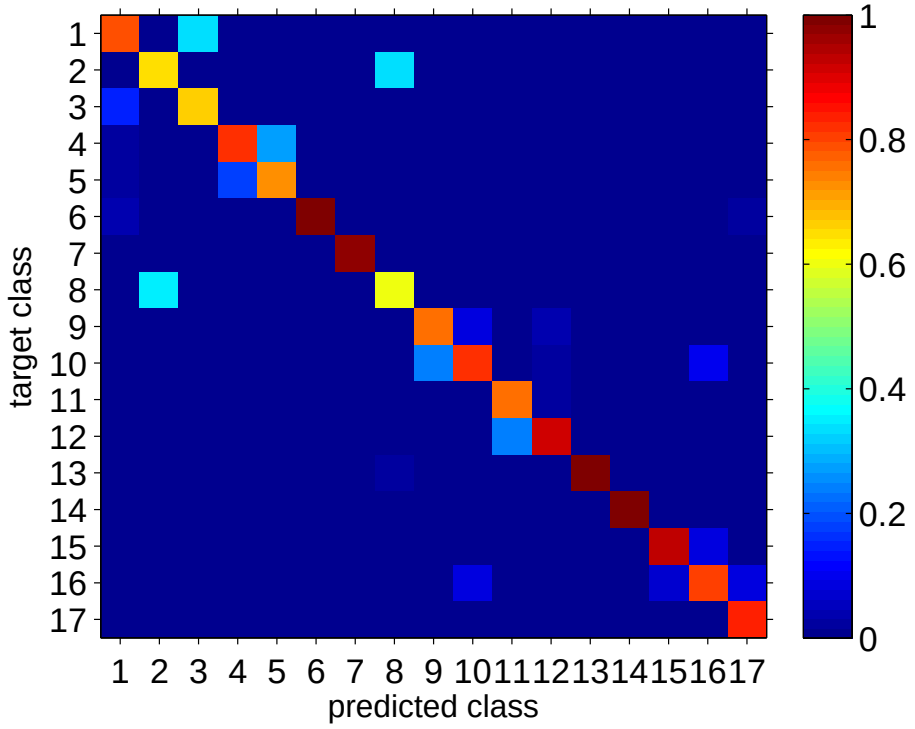


Figure 6.10: Confusion matrix for the 17 classes in EigenNSS-ED system.

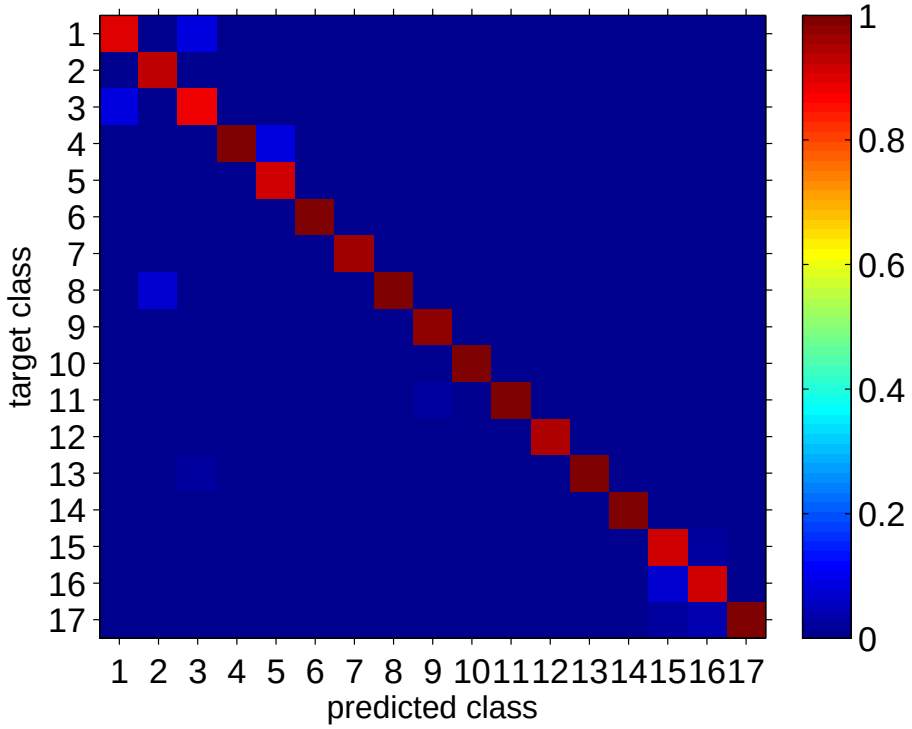


Figure 6.11: Confusion matrix for the 17 classes in FisherNSS-ED diagnostic system.

ing time and a single diagnosis time vary with training samples per class for the previously-mentioned classifiers. The NSS-NB classifier requires a much longer training time than the other classifiers, as shown in Figure 6.12. The training time for the EigenNSS-ED classifier (24 ms-435 ms) is faster than the FisherNSS-ED classifier (33 ms-365 ms) when $n = 10 - 200$. Since both the training times are in the order of milliseconds, this suggests that iterative training does not affect the practical usability of either the EigenNSS-ED or the FisherNSS-ED systems. The diagnosis time for a FisherNSS sample is in the order of microseconds (4 μs–59 μs), which is significantly faster than that of the other types of signatures, despite the fact that FisherNSS calculation involves more steps. This is followed by the EigenNSS-ED system, which has a diagnosis time of about 4 μs–66 μs. This shows that both systems are not limited to on-demand diagnosis, but could also be considered for "real-time" diagnosis applications. Real-time applications are often required to provide guaranteed responses within strict time constraints, usually

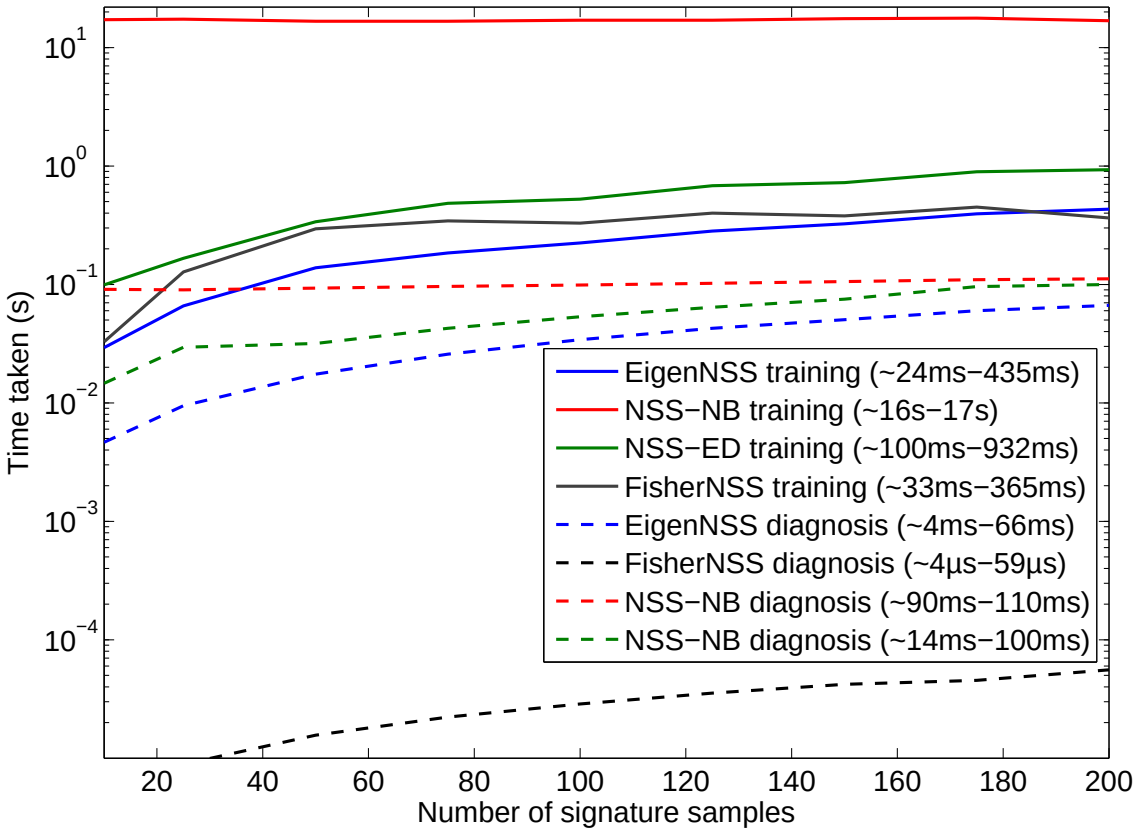


Figure 6.12: Training and single sample diagnosis time variations against increasing training dataset size for classifiers.

in the order of milliseconds and sometimes even microseconds.

6.6 Conclusions

We have proposed and evaluated an automated UD soft-failure diagnostic system, IAND-*h* based on a single hybrid multi-class classifier design. The IAND-*h* system is capable of diagnosing known and unknown faults by combining both supervised and unsupervised ML techniques. We presented the EigenNSS, a signature derived from NSS with limited set of useful features for a unique identification. We have also presented another signature transformation technique to reduce the dimensionality of NSSs and to remove network inconsistencies (unwanted information) even further from EigenNSS. This new transformed signature, FisherNSS, aims to maximise the ratio of the between-class scatter matrix to

the within-class scatter matrix to improve the classification process.

The IAND-*h* system was evaluated by diagnosing 17 UD faults collected over a live campus network and when using NSSs, IAND-*h* resulted in much lower accuracy compared to IAND-*k* system. However, IAND-*h* system achieved an overall accuracy of up to 95% using FisherNSS. With FisherNSS, faults such as CF4, CF6, CF7, CF8, CF9, CF10, CF11, CF12, CF13, CF14, and CF17 have a high True Positive Rate of about 95% and above. We achieved a dimensionality reduction of 96.74% and low confusion rate between classes. Although the IAND-*h* system classifiers with EigenNSS have the shortest training time of about 24 ms–435 ms, the FisherNSS classifiers are only marginally slower at about 33 ms-365 ms. Most importantly, FisherNSS samples have the shortest diagnosis time, in the order of microseconds of about 4μs–66μs, compared to all the other types of samples.

This work provides the foundation to extend the IAND ecosystem to a more sophisticated network environment with thousands of users, diverse client platforms and complex traffic patterns.

CHAPTER SEVEN

CONCLUSIONS AND FUTURE WORK

Users of modern communication networks demand high-speed delivery of information, have low tolerance for service interruptions or slow connections, and expect rapid resolution of any problems encountered. It is becoming infeasible to use traditional, manual methods for the timely identification and resolution of such problems. Automated, intelligent systems are needed for rapid diagnosis of the root causes, especially on UDs, to offer network customers the best quality of experience.

The research reported here aimed to create the necessary scientific base for the automatic diagnosis of network performance issues using intelligent inference methods and machine learning techniques. This thesis has introduced a novel approach to characterising network faults, diagnosing soft failures in network user devices and identifying faulty links, and has proposed a number of fully automated diagnostic systems with in-depth performance evaluations.

This chapter discusses the key contributions and insights of this research and presents avenues for future research.

The major achievements of the research are summarised below.

- The publication of 3 peer-reviewed journal papers (one additional journal publication is not included in the scope of the thesis), and 4 peer-reviewed conference papers.

- The proposal of the concept of Normalised Statistical Signature (NSS), a minimally invasive, robust method to remotely collect and characterise network faults using TCP trace statistics.

- The creation of a link-adaptive signature estimation (LASE) technique to dynamically generate signatures and avoid complexities that arise in rapidly changing network environments.

- The proposal of intelligent automated network diagnostic (IAND) ecosystem for automated diagnosis of network problems.

- The creation of fault classifier modules (FCMs) using support vector machine (SVM)-based pattern classification for identifying individual network faults.

- The proposal and evaluation of the IAND-*k*, a modular and scalable diagnostic system for automatic detection of previously known user device problems (IAND-*k*UD).

- The creation and evaluation of the IAND-*k* for cloud computing (IAND-*k*CC) system for diagnosing user device bottlenecks in cloud environments.

- The proposal of EigenNSS and FisherNSS, two forms of signatures derived from the original NSS for complexity reduction and improved separability.

- The proposal and evaluation of IAND-*h*, a single hybrid classifier architecture with the capability of diagnosing both known and unknown faults with real-time detection capability.

- The creation of one of the largest, accurately labelled active TCP trace datasets in the research field with 1.2 million samples.

7.1 Conclusions and Major contributions

Characterisation of network failures

Due to its position in the middle of the protocol stack, the TCP layer is directly affected by the behavioural characteristics of network elements. Performance bottlenecks or faults in user devices (UDs) alter the typical TCP packet stream and embed unique anomalies in the packet traces, which are called "fault artefacts". These artefacts are the key to diagnosing the root causes of network failures.

The first step in identification is effective and quantitative characterisation of the artefacts. We propose a methodology based on active TCP trace capturing and subsequent extraction of an aggregated statistical feature set to convert the TCP packet trace into a signature. The process transforms the fault artefacts embedded in the packet trace into a unique feature pattern on the signature, resulting in a unique signature for each type of fault. First, raw packet traces of a bi-directional TCP connection between the access server and the UD are captured at the sender device. The connection is user-initiated and performs file transfers of a constant size. We have determined that generally a 20 MB file size offers the best balance between performance and accuracy. The captured traces are then processed to extract per-connection aggregated statistics using a tcptrace-based [193] signature extraction module we have created. The aggregated statistical attributes we extracted provide a robust feature set that is representative of the UD condition, generalises TCP variant-specific connection behaviour, and is captured easily without the requirement to log into the user's device.

Fault signatures

Using the feature set extracted, we propose a normalised statistical signature (NSS), to uniquely characterise UD conditions causing network perfor-

mance issues. Using the NSS, we have demonstrated that each fault has a unique feature pattern that distinguishes them from one another and a healthy UD. Also, with 460 features, the NSSs can represent a wide range of faults within a single representation, unlike many other proposed techniques reported in research the literature.

The normalisation process also introduces a standardised baseline to effectively compare the signatures. We have presented the consistency of NSSs within a single type of fault through a unique visual representation in Figure 3.2. We also evaluated various features in the NSS under dynamic network conditions to identify different types of features in terms of consistency and robustness. We have identified stable-significant features, stable-null features, and unstable features, which need to be taken into consideration when using NSSs in highly dynamic networks or across multiple networks. The NSSs created here provide the foundation required to automate diagnosis.

Link-adaptive signature estimation

The properties of the link connecting the UD and the access router have a significant impact on the NSSs. To generalise the diagnostic capability under various link conditions (i.e. delays, bandwidths, packet losses, etc.), we require an extremely large number of training samples collected with all possible link parameter combinations.

To avoid the need for a large data set, we propose the concept of link-adaptive signature estimation (LASE). LASE uses a smaller data set, collected with different link parameters, to train feature estimator functions to identify the complex relationship between the features and link parameters. Once trained, the feature estimators can predict the feature values and thus the fault signatures over any link within the training parameter range.

We have demonstrated the effectiveness of LASE across a bandwidth range of 100 kB/s – 15 MB/s and a path latency range of 0 ms – 100 ms. We have created feature estimators using neural networks, support vector regression, polynomial models, and radial Bayes functions. A comprehensive comparison of the techniques has shown that neural networks and support vector regression offer the best performance and flexibility. We also identified four further types of features: predictable and unpredictable (chaotic), path-dependent and path-independent. The feature categorisation provides an input to the feature selection process to remove unnecessary features from the NSS. The proposed LASE technique can be extended to include more link parameters following the same methodology when links experience more complex dynamics.

Fault inference from signatures

Using SVM as the pattern classification algorithm, we introduce a "fault classifier module " (FCM), which implements the inference method for artefact identification (see Figure 4.1). Each FCM is designed to detect artefacts from a single type of fault using soft-margin non-linear binary SVM classifiers.

The FCM operates in two phases: first, the training phase creates an appropriate classifier model using trace data samples collected from known faults. The training phase includes signature extraction, data pre-processing and feature selection before SVM training, all vital components for achieving the best classification accuracy. Second, in the diagnostic phase, the trained classifier model is used to determine the artefacts hidden in an undiagnosed trace. We evaluated the FCM's detection capability using NSSs collected in our laboratory network and from live networks. We have used 16 specific types of UD faults to demonstrate that FCMs are capable of detecting spe-

cific issues with a high degree of accuracy, ranging from 90% – 99% in the laboratory network and 88% – 96% in the live network. These FCMs provide the basic building blocks necessary to build more complex and comprehensive diagnostic solutions. Most diagnostic systems in the research literature are purpose-built to diagnose a specific set of faults and require re-design of the algorithms to diagnose new faults. In contrast, systems built with FCMs can be extended by simply replicating an inference module and training with new fault data instead of changing the core inference algorithms in the entire system. Furthermore, the modules can be re-trained and tuned independently. FCMs offer a scalable approach to building a diagnostic system, where the system's diagnostic capability improves as the diversity of fault data increases.

Feature selection

Since not all the features contribute equally to the characterisation of a particular fault, the NSS feature set can be reduced to select only the best features for a specific classification task. The reduction of the feature set reduces the complexity and improves the generalisation capability of the classifier. We propose a two-step, hybrid feature selection technique based on the Student's t-test filter and an iterative wrapper to select the best possible feature set for the particular FCM. The performance evaluation of FCMs demonstrated that the selection of the correct feature set is critical to the success of each FCM. The FCM-based module architecture allows the selection of the best feature set for a specific binary classifier sub-problem, which improves the overall effectiveness of systems built using FCMs.

Automated failure diagnostic systems

We propose the IAND ecosystem, an automated network fault diagnostic system that has two systems: IAND-*k* and IAND-*h*.

IAND-*k* is modular system based on supervised machine learning using FCM as the foundation component to perform complex diagnosis tasks for detecting previously known problem . The system consists of a cascading set of classifiers: (i) the LPD classifier, tasked with first filtering out whether the connection performance problem is caused by link faults or otherwise, (ii) the LFD classifier that diagnoses the exact root cause of a link failure, and (iii) the CFD classifier, tasked with diagnosing the specific UD faults that cause the connection problem. We specifically focus on the UD diagnosis (IAND-*k*UD) and have left link problem diagnosis for future work.

The evaluation of the system with test-bed data has demonstrated the high true-positive and low false-positive rates of the system, achieving a high degree of overall accuracy. We demonstrated that the IAND-*k* system can consistently diagnose multitudes of issues, even when trained with severely limited datasets. We also propose two design choices for the CFD classifier and discuss the trade-off between accuracy and training/diagnosing time overheads between the CFD-P and CFD-M designs.

We also applied the IAND-*k* system to automate the diagnosis of UDs in private cloud environments (IAND-*k*CC). We have demonstrated the IAND-*k*CC system's capability to diagnose UD failures in a real-world dynamic network with a high degree of accuracy while keeping false positives to a minimum.

Transformed signatures for complexity reduction

To overcome the challenges of having high dimensionality in NSSs, we propose methods to emphasise significant global features that contain a maximum amount of information. We propose a transformed signature, EigenNSS by (i) firstly, finding the principal components (i.e eigenvectors of the covariance matrix) of the distribution of NSSs, (ii) then, ordering the eigen-

vectors according to the level of variations among the NSSs, and (iii) finally, projecting the NSSs onto a selected number of eigenvectors that account for the highest variation. EigenNSS aims to reduce the dimensionality of NSS. However, to account for the errors in data collection and the inconsistent nature of the networks, separation between this unwanted information is required for a better classification outcome. We propose FisherNSS, which uses Fisher's linear discriminant (FLD) to provide linear separability between classes by(i) first, using principal component analysis (PCA) to reduce the dimensions of the feature space, and (ii) then, applying further reduction and linear separation with FLD.Consequently, the FisherNSS provides a better class separation then EigenNSS. Experimental evaluation showed both EigenNSS and FisherNSS provided better overall accuracies compared to NSS when used in a diagnostic system. FisherNSS outperforms EigenNSS in terms of accuracy, but EigenNSS has a comparatively shorter training time.

IAND-*h*, a hybrid diagnostic system for known and unknown soft-failure diagnosis

The IAND-*k* diagnostic system is limited to diagnosing known types of faults due to the supervised nature of the learning task. We propose IAND-*h*, a hybrid classifier architecture that combines both supervised and unsupervised ML algorithms for the diagnosis of known and unknown faults in UDs, as a step to compensate for the aforementioned limitations. This new architecture allows the unknown faults to be further analysed using several "clustering" algorithms. The system uses a single multi-class classifier with Euclidean distance-based classification. The evaluation of the system has demonstrated a high level of detection accuracy, and faster training and diagnosis time, the latter feature making it suitable for on-demand as well as real time diagnostic applications.

Network failure datasets

As we do not have access to operator networks, the sample data required for training and testing the diagnostic systems were collected in network test-beds deployed in the university. The creation of accurate fault signatures and the collection of sufficient sample numbers proved to be one of the most challenging parts of the research. We used a combination of fault injection into devices, network emulation and live networks to collect a number of labelled active TCP trace datasets. The data included 17 different UD scenarios, including healthy UDs and faults, 8 different TCP variants at UDs and 3 different test-beds. Link emulation was used to precisely control link conditions in the laboratory test-bed, and live networks with increasing complexity were used in the other two test-beds to collect more realistic data with cross-traffic, congestion and variations. To our knowledge, no such dataset exists of UD failures and this is a key contribution to the networking and ML research community.

7.2 Future work

The results of the current work have clearly demonstrated the potential value of NSSs, fault inference using FCMs, and the potential of automated, end-to-end fault diagnostic systems. The work presented in this thesis opens a number of avenues for future research to explore.

Link fault diagnosis

The fault inference methodology and diagnostic systems proposed in this thesis focused on soft failure diagnosis in UDs. However, a comprehensive diagnostic system would benefit from the capability to diagnose root causes of link problems in conjunction with UD failures. Link failures add an additional level of complexity, as the links undergo complex dynamics and may

be variable in nature. Furthermore, complex multi-hop paths, especially with low bandwidth last-mile connections, add complexity to the creation of robust models, as the relationships between the link-path properties and observed effects may be complex in nature. However, having the ability to diagnose the exact causes of link problems using LFD classifier, especially in an automated scalable system, could add more value in the future.

Evaluation with complex networks

1. Wireless networks are a key part of our lives, and most devices today connect to access routers through wireless access points. The application of NSSs and diagnostic systems in wireless networking environment can provide valuable insights into how complex link characteristics affect the NSSs and overall classifier performance. From our initial evaluations, the proposed methodologies could be directly applied with additional constraints on the environment variations. However, further evaluations are required to identify potential issues.
2. The current analysis of the systems proposed can be expanded into further complex networking environments to evaluate the performance, identify the limitations and propose improvements. As mobile devices are becoming increasingly prominent, the proposed IAND ecosystem can be applied to a mobile-specific set of common problems.
3. Real-world ISP networks with multi-hop connections between the UD and the access routers are common, but complex network architectures. Application of the proposed NSSs and inference mechanism in such networks with real complex data could provide the insights required to improve the NSSs and capabilities of the pattern classifiers.
4. Public cloud infrastructure is another complex networking environ-

ment that could benefit from the IAND ecosystem. However, further evaluations are required to improve the system's dynamic adaptability to cater for the range of devices that utilise a public cloud service. The proposed systems can also be extended to offer dynamic device tuning to improve the user experience based on the current prevailing networking conditions, especially with real-time traffic such as video and chat. In this scenario, the classifiers can be changed into regressions, and dynamic device tuning can be achieved by training with sufficient data.

Improving the NSS

The proposed NSS can be expanded to include derived, higher-order parameter features. With increasing complexity of the networking environments, such features will become important, especially to provide more stability to the features. In particular, bandwidth normalised features, packet loss normalised features and specific ratios that remain unchanged with changing network conditions can be added to the NSS to improve the characterisation capabilities. Furthermore, application-specific traffic such as VoIP and P2P could supplement NSSs for specific applications. The traces from TCP can be supplemented with other common protocols such as RTP, IP and HTTP to capture a wider range of issues. Changing the artificially generated traffic pattern to include variations and injecting scenario-specific traffic into the signature can also improve the effectiveness of the NSS. These avenues can be explored to improve the NSSs applicability across more dynamic and diverse scenarios.

Dissemination of signature data to the research community

Over a two-year period, we have collected over 1.2 million trace samples and associated NSSs with accurate labels. We currently have over 5 TB of

data in secure storage. As no such dataset is available in the research literature, making the dataset available to the research community, both in networking and in machine learning can open up numerous opportunities. We are creating a platform, the Monash University network signature storage and analysis platform (N2SAP) to store the data for easy public access, with search functionality to select the required dataset and also with the future capability to run custom machine learning algorithms. We believe this data set will provide the research community with a baseline to develop an increasingly capable network soft failure diagnostic systems.

Deep machine learning for improved performance in complex networks

With additional complexities arising with complex networking environments and more complex NSSs, machine learning techniques will also need to evolve. Especially with the non-linearities in the real-world scenarios, deep learning provides a unique avenue of research to further expand the inference techniques introduced here. Deep learning is a set of algorithms in machine learning that attempts to model high-level abstractions in data by using model architectures composed of multiple non-linear transformations. Deep learning will allow (i) effective use of the knowledge extracted from unlabelled data, (ii) lessen the chance to converge to a local minima (iii) improved training performance.

Automated diagnostic pathways and resolutions

As we create complex diagnostic systems, the ability to automatically select the system in use or a specific diagnostic pathway through the ecosystem based on the diagnosis scenario and complexity presents an interesting research challenge. This will allow users to deploy IAND as a single, integrated solution without having to select specific components and the system itself will be capable of determining the best method to diagnose a problem.

Furthermore, the IAND ecosystem presented in this thesis focuses only on identification of the issue, but not on the resolution. Extending the capability to dynamically detect, identify and also resolve the network problem would offer a more comprehensive diagnostic solution. These capabilities will greatly enhance the automation and reduce the human intervention required.

APPENDIX ONE

EXTRACTED TCP FEATURES

The following table shows an exhaustive list of features extracted from each trace captured at the client and the server.

Feature	Description
total connection packets S↔C	Total packets in this connection
total packets S→C	Packets sent in each direction
total packets S←C	
ack pkts sent S→C	How many of the packets contained a valid ACK
ack pkts sent S←C	
pure acks sent S→C	The total number of ack packets seen that were not piggy-backed with data (just the TCP header and no TCP data payload) and did not have any of the SYN/FIN/RST flags set.
pure acks sent S←C	
sack pkts sent S→C	The total number of ack packets seen carrying TCP SACK blocks.
sack pkts sent S←C	
max sack blks/ack S→C	The maximum number of sack blocks seen in any sack packet.
max sack blks/ack S←C	
unique bytes sent S→C	How many data bytes were sent (not counting retransmissions)
unique bytes sent S←C	

actual data pkts S→C	How many packets contained any amount of data
actual data pkts S←C	
actual data bytes S→C	How many data bytes (including retransmissions)
actual data bytes S←C	
rexmt data pkts S→C	How many data packets were retransmissions
rexmt data pkts S←C	
rexmt data bytes S→C	How many bytes were retransmissions
rexmt data bytes S→C	
zwnd probe pkts S←C	The count of all the window probe packets seen. (Window probe packets are typically sent by a sender when the receiver last advertised a zero receive window, to see if the window has opened up now).
zwnd probe pkts S→C	
zwnd probe bytes S←C	The total bytes of data sent in the window probe packets.
zwnd probe bytes S→C	
outoforder pkts S←C	How many packets were out of order (or didn't see the first transmit!)
outoforder pkts S→C	
pushed data pkts S←C	The count of all the packets seen with the PUSH bit set in the TCP header.
pushed data pkts S→C	
SYN/FIN pkts sent S←C	How many SYNs and FINs were sent in each direction
SYN/FIN pkts sent S→C	
req 1323 ws/ts S←C	If the endpoint requested Window Scaling/Time Stamp options as specified in RFC 1323.
req 1323 ws/ts S→C	
adv wind scale S←C	The window scaling factor used.
adv wind scale S→C	
urgent data pkts S←C	The total number of packets with the URG bit turned on in the TCP header.
urgent data pkts S→C	
urgent data bytes S←C	The total bytes of urgent data sent. This field is calculated by summing the urgent pointer offset values found in packets having the URG bit set in the TCP header.

urgent data bytes S→C	
mss requested S→C	What what the requested Maximum Segment Size
mss requested S←C	
max segm size S→C	What was the largest segment
max segm size S←C	
min segm size S→C	What was the smallest segment
min segm size S←C	
avg segm size S→C	What was the average segment
avg segm size S←C	
max win adv S→C	What was the largest window advertisement
max win adv S←C	
min win adv S→C	What was the smallest window advertisement
min win adv S←C	
zero win adv S→C	How many times did zero-sized window advertisement were sent
zero win adv S←C	
avg win adv S→C	What was the average window advertisement sent
avg win adv S←C	
initial window pkts S→C	How many bytes in the first window (before the first ACK)
initial window pkts S←C	
initial window bytes S→C	How many packets in the first window (before the first ACK)
initial window bytes S←C	
ttl stream length S→C	What was the total length of the stream (from FIN to SYN)
ttl stream length S→C	
missed data S←C	How many bytes of data were in the stream that's missing?
missed data S→C	
truncated data S←C	The truncated data, calculated as the total bytes of data truncated during packet capture.
truncated data S→C	
truncated packets S←C	The total number of packets truncated as explained above.

truncated packets S→C	
data xmit time S←C	Total data transmit time, calculated as the difference between the times of capture of the first and last packets carrying non-zero TCP data payload.
data xmit time S→C	
idletime max S←C	Maximum idle time, calculated as the maximum time between consecutive packets seen in the direction.
idletime max S→C	
throughput S←C	What was the data throughput (Bytes/second)
throughput S→C	
RTT min S←C	What was the smallest RTT
RTT min S→C	
RTT max S←C	What was the largest RTT
RTT max S→C	
RTT avg S←C	What was the average RTT
RTT avg S→C	
RTT stdev S→C	What was the standard deviation of the RTT that I saw
RTT stdev S←C	
RTT from 3WHS S→C	The RTT value calculated from the TCP 3-Way Hand-Shake (connection opening), assuming that the SYN packets of the connection were captured.
RTT from 3WHS S←C	
RTT full_sz min S→C	The minimum full-size RTT sample.
RTT full_sz min S←C	
RTT full_sz max S→C	The maximum full-size RTT sample.
RTT full_sz max S←C	
RTT full_sz avg S→C	The average full-size RTT sample.
RTT full_sz avg S←C	
RTT full_sz stdev S→C	The standard deviation of full-size RTT samples.

RTT full_sz stdev S←C	
segs cum acked S→C	How many segments were cumulatively ACKed (the ACK that I saw was for a later segment? Might be a lost ACK or a delayed ACK
segs cum acked S←C	
duplicate acks S→C	How many duplicate ACKs
duplicate acks S←C	
triple dupacks S→C	How many triple duplicate ACKs
triple dupacks S→C	
max # retrans S←C	What was the most number of times that a single segment was retransmitted
max # retrans S→C	
min retr time S←C	What was the minimum time between retransmissions of a single segment
min retr time S→C	
max retr time S←C	What was the maximum time between retransmissions of a single segment
max retr time S→C	
avg retr time S←C	What was the average time between retransmissions of a single segment
avg retr time S→C	
sdv retr time S←C	What was the stdev between retransmissions of a single segment
sdv retr time S→C	
max owin S←C	The maximum outstanding unacknowledged data (in bytes) seen at any point in time in the lifetime of the connection.
max owin S→C	
min non-zero owin S←C	The minimum (non-zero) outstanding unacknowledged data (in bytes) seen.
min non-zero owin S→C	
avg owin S←C	The average outstanding unacknowledged data (in bytes), calculated from the sum of all the outstanding data byte samples (in bytes) divided by the total number of samples.
avg owin S→C	
wavg owin S←C	The weighted average outstanding unacknowledged data seen

wavg owin S→C

Table 1.1: Detailed list and descriptions of unique features extracted from TCP traces.

References

[1] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica *et al.*, "A View of Cloud Computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[2] A. Wright, "Ready for a Web OS?" *Communications of the ACM*, vol. 52, no. 12, pp. 16–17, Dec. 2009.

[3] J. C. Mudge, "Cloud Computing: Opportunities and Challenges for Australia," Australian Academy of Technological Sciences and Engineering (ATSE), Melbourne, Case Study Report, 2010. [Online]. Available: http://www.atse.org.au

[4] M. Mathur, "Elucidation of Upcoming Traffic Problems in Cloud Computing," *Recent Trends in Networks and Communications*, vol. 90, pp. 68–79, 2010.

[5] R. Maxion and F. Feather, "A Case Study of Ethernet Anomalies in a Distributed Computing Environment," *IEEE Transactions on Reliability*, vol. 39, no. 4, pp. 433 –443, Oct. 1990.

[6] D. R. Boggs, J. C. Mogul, and C. A. Kent, "Measured Capacity of an Ethernet: Myths and Reality," *SIGCOMM Computer Communication Review*, vol. 18, no. 4, pp. 222–234, Aug. 1988.

[7] D. Ritter and M. Seale, "A Multipurpose, Distributed LAN Traffic Monitoring Tool," *IEEE Network*, vol. 1, no. 3, pp. 32–39, 1987.

[8] M. Soha, "A Distributed Approach to LAN Monitoring Using Intelligent High Performance Monitors," *IEEE Network*, vol. 1, no. 3, pp. 13–20, 1987.

[9] ITU-T Rec. P.10/G. 100, "Vocabulary and effects of transmission parameters on customer opinion of transmission quality," International Telecommunications Union, Geneva, Switzerland, Tech. Rep., 2006. [Online]. Available: https://www.itu.int/rec/T-REC-P.10/en

[10] N. Anders, P. Priyesh, and P. Amanda, "Quality of Service Report," GFK Group, Nordwestring, UK, Technical Report, 2010. [Online]. Available:

http://stakeholders.ofcom.org.uk/binaries/consultations/topcomm/annexes/qos-report.pdf

[11] Cisco, "Wireless RF Interference Customer Survey Results," Cisco Systems, Inc, San Jose, CA, Survey Report 02-03, 2010. [Online]. Available: http://www.webtorials.com/main/resource/papers/cisco/paper163

[12] M. Mathis, J. Heffner, and R. Reddy, "Web100: Extended TCP Instrumentation for Research, Education and Diagnosis," *SIGCOMM Computer Communication Review*, vol. 33, no. 3, pp. 69–79, Jul. 2003.

[13] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel, "The Cost of a Cloud : Research Problems in Data Center Networks," *SIGCOMM Computer Communication Review*, vol. 39, no. 1, pp. 68–73, 2009.

[14] S. Sundaresan, W. de Donato, and N. Feamster, "Broadband Internet Performance: A View From the Gateway," *SIGCOMM Computer Communication Review*, vol. 41, no. 4, pp. 134–145, Aug. 2011.

[15] J. Garcia-Dorado, A. Finamore, M. Mellia, M. Meo, and M. Munafo, "Characterization of ISP Traffic: Trends, User Habits, and Access Technology Impact," *IEEE Transactions on Network and Service Management*, vol. 9, no. 2, pp. 142 –155, Jun. 2012.

[16] M. Thottan and C. Ji, "Anomaly Detection in IP Networks," *IEEE Transactions on Signal Processing*, vol. 51, no. 8, pp. 2191–2204, 2003.

[17] K. P. Kihlstrom, L. E. Moser, and P. M. Melliar-Smith, "Byzantine Fault Detectors for Solving Consensus," *The Computer Journal*, vol. 46, no. 1, pp. 16–35, 2003.

[18] P. J. Chuang and H. M. Hsu, "Effective Fault Diagnosis Approaches for Multipath Networks," *The Computer Journal*, vol. 42, no. 5, pp. 409–421, 1999.

[19] F. Turck, Y. Kiriha, and J.-K. Hong, "Management of the Future Internet: Status and Challenges," *Journal of Network and Systems Management*, vol. 20, pp. 616–624, 2012.

[20] S. Lee and H. S. Kim, "End-user Perspectives of Internet Connectivity Problems," *Computer Networks*, vol. 56, no. 6, pp. 1710–1722, Apr. 2012.

[21] R. Carlson, "Developing the Web100 Based Network Diagnostic Tool (NDT)," in *Proceedings of PAM 03*. San Diego, CA: Springer-Verlag, Berlin, Apr. 2003, pp. 152–161.

[22] W. Wu, P. DeMar, and M. Crawford, "Why Can Some Advanced Ethernet NICs Cause Packet Reordering?" *IEEE Communications Letters*, vol. 15, no. 2, pp. 253 –255, Feb. 2011.

[23] S. Shalunov and R. Carlson, "Detecting Duplex Mismatch on Ethernet," in *Proceedings of PAM 05*. Boston, MA: Springer-Verlag, Berlin, Oct. 2005, pp. 135–148.

[24] C. Callegari, S. Giordano, M. Pagano, and T. Pepe, "Behavior Analysis of TCP Linux Variants," *Computer Networks*, vol. 56, no. 1, pp. 462–476, Jan. 2012.

[25] M. Barbara, H. Russ, and B. George, *Bridging the Gap Workshop Report*, Internet2, Ann Arbor, Michigan, USA, Aug. 2005. [Online]. Available: grantome.com/grant/NSF/ACI-0443254

[26] K. Anatoly and S. Stanislav, "NetFlow Weekly Reports," Internet2: NetFlow, Ann Arbor, Michigan, USA, Tech. Rep., 2010. [Online]. Available: https://www.manageengine.com/products/netflow/netflowreports.html

[27] J. Postel, "Transmission Control Protocol," RFC 793 (Internet Standard), Internet Engineering Task Force, Sep. 1981, updated by RFCs 1122, 3168, 6093, 6528. [Online]. Available: http://www.ietf.org/rfc/rfc793.txt

[28] M. Siekkinen, G. Urvoy-Keller, E. W. Biersack, and T. En-Najjary, "Root Cause Analysis for Long-lived TCP Connections," in *Proceedings of the ACM conference on Emerging network experiment and technology (CoNEXT'05)*. Toulouse, France: ACM, Oct. 2005, pp. 200–210.

[29] A. X. Zheng, J. Lloyd, and E. Brewer, "Failure Diagnosis Using Decision Trees," in *Proceedings of the First International Conference on Autonomic Computing*, ser. ICAC '04. New York, NY, USA: IEEE Computer Society, 2004, pp. 36–43.

[30] M. Y. Chen, A. Accardi, E. Kiciman, J. Lloyd, D. Patterson, A. Fox, and E. Brewer, "Path-based Failure and Evolution Management," in *Proceedings of the 1st Conference on Symposium on Networked Systems Design and Implementation - Volume 1*, ser. NSDI'04. Berkeley, CA, USA: USENIX Association, 2004, pp. 23–23.

[31] M. Chen, E. Kiciman, E. Fratkin, A. Fox, and E. Brewer, "Pinpoint: Problem Determination in Large, Dynamic Internet Services," in *Proceedings of the International Conference on Dependable Systems and Networks (DSN 02)*. Washington, DC, USA: IEEE Computer Society, Jun. 2002, pp. 595 – 604.

[32] A. V. Mirgorodskiy, N. Maruyama, and B. P. Miller, "Problem Diagnosis in Large-scale Computing Environments," in *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*, ser. SC '06. New York, NY, USA: ACM, 2006, pp. 595 – 604.

[33] S. Kandula, R. Mahajan, P. Verkaik, S. Agarwal, J. Padhye, and P. Bahl, "Detailed Diagnosis in Enterprise Networks," *SIGCOMM Computer Communication Review*, vol. 39, pp. 243–254, August 2009.

[34] Y. Jin, N. Duffield, A. Gerber, P. Haffner, S. Sen, and Z.-L. Zhang, "NEVERMIND, the Problem is Already Fixed: Proactively Detecting and Troubleshooting Customer DSL Problems," in *Proceedings of the 6th International Conference on emerging Networking EXperiments and Technologies (Co-NEXT '10)*. ACM, 2010, pp. 7:1–7:12.

[35] A. A. Mahimkar, Z. Ge, A. Shaikh, J. Wang, J. Yates, Y. Zhang, and Q. Zhao, "Towards Automated Performance Diagnosis in a Large IPTV Network," *SIGCOMM Computer Communication Review*, vol. 39, pp. 231–242, August 2009.

[36] B. Aggarwal, R. Bhagwan, T. Das, S. Eswaran, V. Padmanabhan, and G. Voelker, "NetPrints: Diagnosing Home Network Misconfigurations Using Shared Knowledge," in *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation*.

[37] C. Yuan, N. Lao, J.-R. Wen, J. Li, Z. Zhang, Y.-M. Wang, and W.-Y. Ma, "Automated Known Problem Diagnosis With Event Traces," in *ACM SIGOPS Operating Systems Review*, vol. 40, no. 4. ACM, 2006, pp. 375–388.

[38] Ookla, "Speedtest," 2014. [Online]. Available: URL: http://www.speedtest.net

[39] A. McKenzie and O. Telecommunications, "Oz broadband speed test," 2003. [Online]. Available: URL: http://www.ozspeedtest.com/

[40] D. Kundu and S. I. Lavlu, *Cacti 0.8 network monitoring*. Packt Publishing Ltd, 2009. [Online]. Available: URL: http://www.cacti.net

[41] K. Anagnostakis, S. Ioannidis, S. Miltchev, M. Greenwald, J. Smith, and J. Ioannidis, "Efficient Packet Monitoring for Network Management," in *Network Operations and Management Symposium, 2002 (NOMS 02)*. Florence, Italy: IEEE/IFIP, Apr. 2002, pp. 423 – 436.

[42] H. Huang, R. Jennings, Y. Ruan, R. Sahoo, S. Sahu, and A. Shaikh, "PDA: a Tool for Automated Problem Determination," in *Proceedings of LISA 07*. Dallas, TX: USENIX Association, CA, USA, Nov. 2007, pp. 13:1–13:14.

[43] S. D. Galup, R. Dattero, J. J. Quan, and S. Conger, "An Overview of IT Service Management," *Communications of the ACM*, vol. 52, no. 5, pp. 124–127, 2009.

[44] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly Detection: A Survey," *ACM Computing Surveys*, vol. 41, no. 3, pp. 15:1–15:58, Jul. 2009.

[45] W. S. Cleveland, D. Lin, and D. X. Sun, "IP Packet Generation: Statistical Models for TCP Start Times Based on Connection-rate Superposition," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 28, no. 1. ACM, 2000, pp. 166–177.

[46] R. Caceres, N. Duffield, A. Feldmann, J. D. Friedmann, A. Greenberg, R. Greer, T. Johnson, C. R. Kalmanek, B. Krishnamurthy, and D. Lavelle, "Measurement and Analysis of IP Network Usage and behavior," *IEEE Communications Magazine*, vol. 38, no. 5, pp. 144–151, 2000.

[47] P. Aukia, M. Kodialam, P. V. Koppol, T. Lakshman, H. Sarin, and B. Suter, "RATES: A Server for MPLS Traffic Engineering," *IEEE Network*, vol. 14, no. 2, pp. 34–41, 2000.

[48] J. Caberera, B. Ravichandran, and R. Mehra, "Statistical Traffic Modeling for Network Intrusion Detection," in *Proceedings of the 8th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, San Francisco, CA, USA, Sep. 2000, pp. 466–473.

[49] B. Zhang, J. Yang, J. Wu, and Z. Wang, "MBST: Detecting Packet-Level Traffic Anomalies by Feature Stability," *The Computer Journal*, vol. 56, no. 10, pp. 1176–1188, 2013.

[50] M. Hong, R. Gu, H. Wang, Y. Sun, and Y. Ji, "Identifying Online Traffic Based on Property of TCP Flow," *Journal of China Universities of Posts and Telecommunications*, vol. 16, no. 3, pp. 84–88, 2009.

[51] W. John and S. Tafvelin, "Analysis of Internet Backbone Traffic and Header Anomalies Observed," in *Proceedings of the ACM Internet Measurements Conference (IMC'07)*. San Diego, CA, USA: ACM, Oct. 2007, pp. 111–116.

[52] H. Dahmouni, S. Vaton, and D. Rossé, "A Markovian Signature-Based Approach to IP Traffic Classification," in *Proceedings of the Third Annual ACM Workshop on Mining Network Data*. San Diego, California, USA: ACM, New York, 2007, pp. 29–34.

[53] V. Paxson, M. Allman, S. Dawson, W. Fenner, J. Griner, I. Heavens, K. Lahey, J. Semke, and B. Volz, "Known TCP Implementation Problems," RFC 2525 (Informational), Internet Engineering Task Force, Mar. 1999. [Online]. Available: http://www.ietf.org/rfc/rfc2525.txt

[54] M. Hassani, M. Mohammadi, and S. Pashforoush, "A Novel Recursive Method for Analysis Performance of TCP Flow in Wireless LAN," in *Proceedings of the Second International Conference on Communication Software and Networks (ICCSN '10)*, Singapore, Feb 2010, pp. 387–391.

[55] H. Hisamatsu, G. Hasegawa, and M. Murata, "Performance Analysis of Large-scale IP Networks Considering TCP Traffic," *IEICE Transactions on Communications*, vol. 90, no. 10, pp. 2845–2853, 2007.

[56] I. Khalifa and L. Trajkovic, "An Overview and Comparison of Analytical TCP Models," in *Proceedings of the 2004 International Symposium on Circuits and Systems (ISCAS '04)*, vol. 5. Vancouver, Canada: IEEE, New York, May 2004, pp. 469–472.

[57] K. McCloghrie and M. Rose, "Management Information Base for Network Management of TCP/IP-based internets:MIB-II," RFC 1213 (Iinternet Standard), Internet Engineering Task Force, Mar. 1991, updated by RFCs 2011, 2012, 2013. [Online]. Available: http://www.ietf.org/rfc/rfc1213.txt

[58] K. L. Calvert, W. K. Edwards, N. Feamster, R. E. Grinter, Y. Deng, and X. Zhou, "Instrumenting Home Networks," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 1, pp. 84–89, 2011.

[59] R. Chandra, V. N. Padmanabhan, and M. Zhang, "WiFiProfiler: Cooperative Diagnosis in Wireless LANs," in *Proceedings of the 4th international conference on Mobile systems, applications and services*. Uppsala, Sweden: ACM, Jun. 2006, pp. 205–219.

[60] S. Zander, T. Nguyen, and G. Armitage, "Self-Learning IP Traffic Classification Based on Statistical Flow Characteristics," in *Passive and Active Network Measurement*, ser. Lecture Notes in Computer Science, C. Dovrolis, Ed. Springer Berlin Heidelberg, 2005, vol. 3431, pp. 325–328.

[61] S. Zander, T. T. Nguyen, and G. Armitage, "Automated traffic classification and application identification using machine learning," in *The IEEE Conference on Local Computer Networks, 2005.*, Sydney, Australia, June 2005, pp. 250–257.

[62] T. T. Nguyen and G. Armitage, "A Survey of Techniques for Internet Traffic Classification Using Machine Learning," *Communications Surveys Tutorials, IEEE*, vol. 10, no. 4, pp. 56–76, Oct. 2008.

[63] T. Nguyen and G. Armitage, "Training on multiple sub-flows to optimise the use of Machine Learning classifiers in real-world IP networks," in *Proceedings of the 31st IEEE Conference on Local Computer Networks*, Dubai, Nov 2006, pp. 369–376.

[64] T. T. Nguyen and G. Armitage, "Clustering to assist supervised machine learning for real-time IP traffic classification," in *IEEE International Conference on Communications, ICC '08*, Beijing, China, May 2008, pp. 5857–5862.

[65] P. A. Branch, A. Heyde, and G. J. Armitage, "Rapid identification of skype traffic flows," in *Proceedings of the 18th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, ser. NOSSDAV '09. Williamsburg, VA, USA: ACM, June 2009, pp. 91–96.

[66] P. Branch and J. But, "Rapid and generalized identification of packetized voice traffic flows," in *IEEE 37th Conference on Local Computer Networks (LCN)*, Clearwater, Florida, USA, Oct 2012, pp. 85–92.

[67] J. But, P. Branch, and T. Le, "Rapid Identification of BitTorrent traffic," in *IEEE 35th Conference on Local Computer Networks (LCN)*, Denver, Colorado, USA, Oct 2010, pp. 536–543.

[68] H. Hajji, "Statistical Analysis of Network Traffic for Adaptive Faults Detection," *IEEE Transactions on Neural Networks*, vol. 16, no. 5, pp. 1053–1063, Sep. 2005.

[69] M. V. J. Heikkinen and A. W. Berger, "Comparison of User Traffic Characteristics on Mobile-access Versus Fixed-access Networks," in *Proceedings of PAM 12*. Vienna, Austria: Springer-Verlag, Berlin, Mar. 2012, pp. 32–41.

[70] C. Manikopoulos and S. Papavassiliou, "Network Intrusion and Fault Detection: A Statistical Anomaly Approach," *IEEE Communications Magazine*, vol. 40, no. 10, pp. 76 – 82, Oct. 2002.

[71] K. Shafi and H. A. Abbass, "An Adaptive Genetic-based Signature Learning System for Intrusion Detection," *Expert Systems with Applications*, vol. 36, no. 10, pp. 12 036–12 043, Dec. 2009.

[72] Z. Chen, Y. Zhang, Z. Chen, and A. Delis, "A Digest and Pattern Matching-Based Intrusion Detection Engine," *The Computer Journal*, vol. 52, no. 6, pp. 699–723, Aug. 2009.

[73] J. V. Gomes, P. R. Incio, M. Pereira, M. M. Freire, and P. P. Monteiro, "Exploring Behavioral Patterns Through Entropy in Multimedia Peer-to-Peer Traffic," *The Computer Journal*, vol. 55, no. 6, pp. 740–755, 2012.

[74] M. Wolfgang. (2002, Nov.) Host Discovery with nmap. Palo Alto, CA, USA. [Online]. Available: http://nmap.org/book/man-host-discovery.html

[75] R. Beverly, "A Robust Classifier for Passive TCP/IP Fingerprinting," *Passive and Active Network Measurement*, vol. 3015, pp. 158–167, 2004.

[76] S. Erjongmanee and C. Ji, "Large-Scale Network-Service Disruption: Dependencies and External Factors," *IEEE Transactions on Network and Service Management*, vol. 8, no. 4, pp. 375 –386, Dec. 2011.

[77] T. Reidemeister, M. Jiang, and P. Ward, "Mining Unstructured Log Files for Recurrent Fault Diagnosis," in *Proceedings of the IFIP/IEEE International Symposium onIntegrated Network Management (IM '11)*. Dublin, Ireland: IEEE/IFIP, New York, May 2011, pp. 377 –384.

[78] S. Dawson and F. Jahanian, "Probing and Fault Injection of Dependable Distributed Protocols," *The Computer Journal*, vol. 38, no. 4, pp. 286–300, 1995.

[79] I. Cunha, R. Teixeira, and C. Diot, "Measuring and Characterizing End-to-End Route Dynamics in the Presence of Load Balancing," in *Proceedings of PAM 11*. Atlanta, GA: Springer-Verlag, Berlin, Mar. 2011, pp. 235–244.

[80] F. Feather, D. Siewiorek, and R. Maxion, "Fault Detection in an Ethernet Network Using Anomaly Signature Matching," *SIGCOMM Computer Communication Review*, vol. 23, no. 4, pp. 279–288, Oct. 1993.

[81] M. Roughan, S. Sen, O. Spatscheck, and N. Duffield, "Class-of-Service Mapping for QoS: A Statistical Signature-based Approach to IP Traffic Classification," in *Proceedings of SIGCOMM IMC 04*. Taormina, Italy: ACM, New York, Oct. 2004, pp. 135–148.

[82] T. Kihara, N. Tateishi, and S. Seto, "Evaluation of Network Fault-detection Method Based on Anomaly Detection With Matrix Eigenvector," in *Proceedings of APNOMS 11*. Taipei, Taiwan: IEEE, New York, Sep. 2011, pp. 1 –7.

[83] R. Vaarandi, "A Data Clustering Algorithm for Mining Patterns From Event Logs," in *Proceedings of the 3rd IEEE Workshop on IP Operations Management (IPOM '03)*, Kansas City, MO, USA, Oct 2003, pp. 119–126.

[84] P. Belhumeur, J. Hespanha, and D. Kriegman, "Eigenfaces vs. Fisherfaces: Recognition Using Class Specific Linear Projection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 19, no. 7, pp. 711–720, 1997.

[85] G. Khanna, M. Y. Cheng, P. Varadharajan, S. Bagchi, M. Correia, and P. Verissimo, "Automated Rule-Based Diagnosis Through a Distributed Monitor System," *IEEE Transactions on Dependable and Secure Computing*, vol. 4, no. 4, pp. 266–279, Oct 2007.

[86] H. J. Wang, J. Platt, Y. Chen, R. Zhang, and Y.-M. Wang, "PeerPressure for Automatic Troubleshooting," *ACM Sigmetrics Performance Evaluation Review*, vol. 32, no. 1, pp. 398–399, 2004.

[87] T. Ndousse and T. Okuda, "Computational Intelligence for Distributed Fault Management in Networks Using Fuzzy Cognitive Maps," in *Proceedings of the IEEE International Conference on Communications*, vol. 3. Dallas, TX, USA: IEEE, Jun. 1996, pp. 1558–1562.

[88] L. Lewis, "A Case-based Reasoning Approach to the Management of Faults in Communication Networks," in *Proceedings of the Twelfth Annual Joint Conference of the IEEE Computer and Communications Societies*. San Francisco, CA, USA: IEEE, Apr. 1993, pp. 1422–1429.

[89] A. Morales De Franceschi, L. F. Kormann, and C. B. Westphall, "Performance Evaluation for Proactive Network Management," in *Proceedings of the IEEE International Conference on Communications, 1996*, vol. 1. Dallas, TX, USA: IEEE, 1996, pp. 22–26.

[90] A. Gill, "Single-Channel and Multichannel Finite-State Machines," *IEEE Transactions on Computers*, vol. C-19, no. 11, pp. 1073–1078, Nov 1970.

[91] I. Katzela and M. Schwartz, "Schemes for Fault Identification in Communication Networks," *IEEE/ACM Transactions on Networking*, vol. 3, no. 6, pp. 753–764, 1995.

[92] I. Rouvellou and G. W. Hart, "Automatic Alarm Correlation for Fault Identification," in *Proceedings of the Fourteenth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'95)*, vol. 2. Boston, MA, USA: IEEE, Apr 1995, pp. 553–561.

[93] A. Lazar, W. Wang, and R. Deng, "Models and Algorithms for Network Fault Detection and Identification: A Review," in *Proceedings of the ICCS/ISITA '92*, vol. 3. Singapore: IEEE, Nov 1992, pp. 999–1003.

[94] L. Lewis and G. Dreo, "Extending Trouble Ticket Systems to Fault Diagnostics," *IEEE Network*, vol. 7, no. 6, pp. 44–51, 1993.

[95] M. Thottan and C. Ji, "Anomaly Detection in IP Networks," *IEEE Transactions on Signal Processing*, vol. 51, no. 8, pp. 2191–2204, Aug 2003.

[96] H. Wang, D. Zhang, and K. G. Shin, "Detecting SYN Flooding Attacks," in *Proceedings of the Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies*, vol. 3. New York, NY, USA: IEEE, Jun. 2002, pp. 1530–1539.

[97] P. Qiu and D. Hawkins, "A Nonparametric Multivariate Cumulative Sum Procedure for Detecting Shifts in All Directions," *Journal of the Royal Statistical Society: Series D (The Statistician)*, vol. 52, no. 2, pp. 151–164, Apr. 2003.

[98] A. S. Hadi, "A New Measure of Overall Potential Influence in Linear Regression," *Computational Statistics & Data Analysis*, vol. 14, no. 1, pp. 1–27, 1992.

[99] W. Tian and J. Liu, "Intrusion Detection Quantitative Analysis with Support Vector Regression and Particle Swarm Optimization Algorithm," in *Proceedings of the International Conference on Wireless Networks and Information Systems*. Shanghai, China: IEEE, Dec. 2009, pp. 133–136.

[100] D. Anderson, T. F. Lunt, H. Javitz, A. Tamaru, A. Valdes *et al.*, *Detecting Unusual Program Behavior Using the Statistical Component of the Next-generation Intrusion Detection Expert System (NIDES)*. SRI International, Computer Science Laboratory, 1995.

[101] P. N. Tan, M. Steinbach, and V. Kumar, *Introduction to data mining*. Pearson Addison Wesley, Boston, 2006, vol. 1.

[102] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern classification*. John Wiley & Sons, New York, 1999.

[103] C. De Stefano, C. Sansone, and M. Vento, "To Reject or Not to Reject: That is the Question-An Answer in Case of Neural Classifiers," *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, vol. 30, no. 1, pp. 84–94, 2000.

[104] D. Barbara, N. Wu, and S. Jajodia, "Detecting Novel Network Intrusions Using Bayes Estimators," in *Proceedings of the 1st SIAM International Conference on Data Mining*. Chicago, IL, USA: SIAM, Apr. 2001, pp. 1–17.

[105] G. Ratsch, S. Mika, B. Scholkopf, and K. Muller, "Constructing Boosting Algorithms from SVMs: An Application to One-class Classification," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 9, pp. 1184–1199, 2002.

[106] V. Roth, "Outlier Detection with One-class Kernel Fisher Discriminants," in *Advances in Neural Information Processing Systems (NIPS'04)*, Vancouver, Canada, Dec. 2004, pp. 1169–1176.

[107] J. R. Quinlan, *C4.5: Programs for Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993.

[108] S. Murthy, "Automatic Construction of Decision Trees from Data: A Multi-disciplinary Survey," *Data Mining and Knowledge Discovery*, vol. 2, no. 4, pp. 345–389, 1998.

[109] Y. Mao, "Automated Computer System Diagnosis by Machine Learning Approaches," UPenn CIS Dept, Tech. Rep., 2005.

[110] I. El Khayat, P. Geurts, and G. Leduc, "Machine-learnt Versus Analytical Models of TCP Throughput," *Computer Networks*, vol. 51, no. 10, pp. 2631–2644, 2007.

[111] F. Rosenblatt, *Principles of neurodynamics*. Spartan Book, New York, 1962.

[112] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning Representations by Back-propagating Errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[113] G. P. Zhang, "Neural Networks for Classification: A Survey," *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, vol. 30, no. 4, pp. 451–462, 2000.

[114] Z.-H. Zhou, "Rule Extraction: Using Neural Networks or for Neural Networks?" *Journal of Computer Science and Technology*, vol. 19, no. 2, pp. 249–253, 2004.

[115] L. Camargo and T. Yoneyama, "Specification of Training Sets and the Number of Hidden Neurons for Multilayer Perceptrons," *Neural Computation*, vol. 13, no. 12, pp. 2673–2680, 2001.

[116] C. Neocleous and C. Schizas, "Artificial Neural Network Learning: A Comparative Review," in *Methods and Applications of Artificial Intelligence*. Springer, 2002, pp. 300–313.

[117] S. J. Russell, P. Norvig, J. F. Canny, J. M. Malik, and D. D. Edwards, *Artificial Intelligence: A Modern Approach*. Prentice Hall, Englewood Cliffs, 1995, vol. 74.

[118] S. Chavan, K. Shah, N. Dave, S. Mukherjee, A. Abraham, and S. Sanyal, "Adaptive Neuro-fuzzy Intrusion Detection Systems," in *Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC'04)*, vol. 1. Las Vegas, Nevada, USA: IEEE, Apr. 2004, pp. 70–74.

[119] J. Ryan, M.-J. Lin, and R. Miikkulainen, "Intrusion Detection with Neural Networks," in *Advances in Neural Information Processing Systems*. Cambridge, MA: MIT Press, 1998, pp. 943–949.

[120] A. Hiramatsu, "ATM Communications Network Control by Neural Networks," *IEEE Transactions on Neural Networks*, vol. 1, no. 1, pp. 122–130, 1990.

[121] J. Burroni and C. Sarraute, "Using Neural Networks for Remote OS Identification," in *Proceedings of the Pacific Security Conference (PacSec 05)*, Tokyo, Japan, 2005, 2005.

[122] S. Jakubek and T. Strasser, "Fault-diagnosis Using Neural Networks with Ellipsoidal Basis Functions," in *Proceedings of the 2002 American Control Conference*, vol. 5. Anchorage, Alaska, USA: IEEE, May 2002, pp. 3846–3851.

[123] A. Hofmann and B. Sick, "Evolutionary Optimization of Radial Basis Function Networks for Intrusion Detection," in *Proceedings of the International Joint Conference on Neural Networks*, vol. 1. Dallas, TX, USA: IEEE, Jul. 2003, pp. 415–420.

[124] D. Fisch, A. Hofmann, and B. Sick, "On the Versatility of Radial Basis Function Neural Networks: A Case Study in the Field of Intrusion Detection," *Information Sciences*, vol. 180, no. 12, pp. 2421–2439, 2010.

[125] Y. Liu, "QPSO-Optimized RBF Neural Network for Network Anomaly Detection," *Journal of Information & Computational Science*, vol. 8, no. 9, pp. 1479–1485, 2011.

[126] V. N. Vapnik, *The Nature of Statistical Learning Theory*. New York, NY, USA: Springer-Verlag Inc., 1995.

[127] C. J. Burges, "A Tutorial on Support Vector Machines for Pattern Recognition," *Data Mining and Knowledge Discovery*, vol. 2, no. 2, pp. 121–167, 1998.

[128] N. Cristianini and J. Shawe-Taylor, *An Introduction to Support Vector Machines And Other Kernel-based Learning Methods*. New York, NY, USA: Cambridge University Press, 2000.

[129] S. J. Wright and J. Nocedal, *Numerical optimization*. New York, NY, USA: Springer, 1999, vol. 2.

[130] K. Veropoulos, C. Campbell, and N. Cristianini, "Controlling the Sensitivity of Support Vector Machines," in *Proceedings of the International Joint Conference on AI (IJCAI 99)*, Stockholm, Sweden, Jul. 1999, p. 5560.

[131] B. Schölkopf, C. J. Burges, and A. J. Smola, *Advances in kernel methods: support vector learning*. Cambridge, MA: MIT Press, 1999.

[132] M. G. Genton, "Classes of Kernels for Machine Learning: A Statistics Perspective," *The Journal of Machine Learning Research*, vol. 2, pp. 299–312, 2002.

[133] E. Eskin, "Anomaly Detection over Noisy Data Using Learned Probability Distributions," in *Proceedings of the Seventeenth International Conference on Machine Learning*, ser. ICML'00. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2000, pp. 255–262.

[134] K. Heller, K. Svore, A. D. Keromytis, and S. Stolfo, "One Class Support Vector Machines for Detecting Anomalous Windows Registry Accesses," in *Workshop on Data Mining for Computer Security (DMSEC)*, Melbourne, FL, Nov. 2003, pp. 2–9.

[135] A. Lazarevic, L. Ertöz, V. Kumar, A. Ozgur, and J. Srivastava, "A Comparative Study of Anomaly Detection Schemes in Network Intrusion Detection," in *Proceedings of the Third SIAM International Conference on Data Mining*. SIAM, 2003, pp. 25–36.

[136] G. La Mantia, D. Rossi, A. Finamore, M. Mellia, and M. Meo, "Stochastic Packet Inspection for TCP Traffic," in *Proceedings of the IEEE International Conference on Communications (ICC)*. Cape Town, South Africa: IEEE, May 2010, pp. 1–6.

[137] A. Este, F. Gringoli, and L. Salgarelli, "Support Vector Machines for TCP Traffic Classification," *Computer Networks*, vol. 53, no. 14, pp. 2476–2490, 2009.

[138] R. Alshammari and A. N. Zincir-Heywood, "Machine Learning Based Encrypted Traffic Classification: Identifying SSH and Skype," in *Proceedings of the IEEE Symposium on Computational Intelligence for Security and Defense Applications (CISDA'09)*. Ottawa, Canada: IEEE, Jul. 2009, pp. 1–8.

[139] M. Mirza, J. Sommers, P. Barford, and X. Zhu, "A Machine Learning Approach to TCP Throughput Prediction," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 35, no. 1. ACM, 2007, pp. 97–108.

[140] D. Basak, S. Pal, and D. Patranabis, "Support Vector Regression," *Neural Information Processing-Letters and Reviews*, vol. 11, no. 10, pp. 203–224, 2007.

[141] T. Koski and J. Noble, *Bayesian networks: An Introduction*, 1st ed. New York, NY, USA: John Wiley & Sons, 2009, vol. 924.

[142] R. Daly, Q. Shen, and S. Aitken, "Review: Learning Bayesian Networks: Approaches and Issues," *The Knowledge Engineering Review*, vol. 26, no. 2, pp. 99–157, 2011.

[143] I. Rish, "An Empirical Study of the Naive Bayes Classifier," in *Proceedings of the Workshop on Empirical Methods in Artificial Intelligence (IJCAI'01)*, vol. 3, no. 22. IBM New York, 2001, pp. 41–46.

[144] Y. Yang and G. I. Webb, "On why Discretization Works for Naive-Bayes Classifiers," in *AI 2003: Advances in Artificial Intelligence*. Springer, 2003, pp. 440–452.

[145] P. Xie, J. H. Li, X. Ou, P. Liu, and R. Levy, "Using Bayesian Networks for Cyber Security Analysis," in *2010 IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. Chicago, IL, USA: IEEE, Jun. 2010, pp. 211–220.

[146] A. Valdes and K. Skinner, "Adaptive, Model-Based Monitoring for Cyber Attack Detection," in *Recent Advances in Intrusion Detection*, ser. Lecture Notes in Computer Science, H. Debar, L. M, and S. Wu, Eds., vol. 1907. Springer Berlin Heidelberg, 2000, pp. 80–93.

[147] N. Fonseca and M. Crovella, "Bayesian Packet Loss Detection for TCP," in *Proceedings of the 24th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'05)*, vol. 3. Miami, FL, USA: IEEE, Mar. 2005, pp. 1826–1837.

[148] A. Bashar, G. Parr, S. McClean, B. Scotney, and D. Nauck, "Knowledge Discovery Using Bayesian Network Framework for Intelligent Telecommunication Network Management," in *Proceedings of the 4th International Conference on Knowledge Science, Engineering and Management*, ser. KSEM'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 518–529.

[149] B. Agrawal, N. Kumar, and M. Molle, "Controlling spam emails at the routers," in *2005 IEEE International Conference on Communications,(ICC'05)*, vol. 3. Seoul, South Korea: IEEE, May 2005, pp. 1588–1592.

[150] M. Tariq, A. Zeitoun, V. Valancius, N. Feamster, and M. Ammar, "Answering What-if Deployment and Configuration Questions with WISE," in *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 4. ACM, 2008, pp. 99–110.

[151] R. M. Khanafer, B. Solana, J. Triola, R. Barco, L. Moltsen, Z. Altman, and P. Lazaro, "Automated Diagnosis for UMTS Networks Using Bayesian Network Approach," *IEEE Transactions on Vehicular Technology*, vol. 57, no. 4, pp. 2451–2461, 2008.

[152] R. Barco, P. Lázaro, L. Díez, and V. Wille, "Continuous Versus Discrete Model in Autodiagnosis Systems for Wireless Networks," *IEEE Transactions on Mobile Computing*, vol. 7, no. 6, pp. 673–681, 2008.

[153] S. Furao and O. Hasegawa, "An Incremental Network for On-line Unsupervised Classification and Topology Learning," *Neural Networks*, vol. 19, no. 1, pp. 90–106, 2006.

[154] K. Yamanishi and J.-i. Takeuchi, "Discovering Outlier Filtering Rules from Unlabeled Data: Combining a Supervised Learner with an Unsupervised Learner," in *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '01. New York, NY, USA: ACM, 2001, pp. 389–394.

[155] T. Cover and P. Hart, "Nearest Neighbor Pattern Classification," *IEEE Transactions on Information Theory*, vol. 13, no. 1, pp. 21–27, 1967.

[156] S. Boriah, V. Chandola, and V. Kumar, "Similarity Measures for Categorical Data: A Comparative Evaluation," in *In Proceedings of the eighth SIAM International Conference on Data Mining*, vol. 30, no. 2. SIAM, 2008, pp. 243–254.

[157] A. A. Abbasi and M. Younis, "A Survey on Clustering Algorithms for Wireless Sensor Networks," *Computer Communications*, vol. 30, no. 14, pp. 2826–2841, 2007.

[158] P. Berkhin, "A Survey of Clustering Data Mining Techniques," in *Grouping Multidimensional Data*. Springer Berlin Heidelberg, 2006, pp. 25–71.

[159] R. Xu and D. Wunsch, "Survey of Clustering Algorithms," *IEEE Transactions on Neural Networks*, vol. 16, no. 3, pp. 645–678, 2005.

[160] S. C. Johnson, "Hierarchical Clustering Schemes," *Psychometrika*, vol. 32, no. 3, pp. 241–254, 1967.

[161] J. A. Hartigan and M. A. Wong, "Algorithm AS 136: A K-means Clustering Algorithm," *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, vol. 28, no. 1, pp. 100–108, 1979.

[162] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, and A. Y. Wu, "An efficient K-means Clustering Algorithm: Analysis and Implementation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 7, pp. 881–892, 2002.

[163] S. Lloyd, "Least Squares Quantization in PCM," *IEEE Transactions on Information Theory*, vol. 28, no. 2, pp. 129–137, 1982.

[164] X. Xu, M. Ester, H.-P. Kriegel, and J. Sander, "A Distribution-Based Clustering Algorithm for Mining in Large Spatial Databases," in *Proceedings of the 14th International Conference on Data Engineering*. Orlando, FL, USA: IEEE, Feb. 1998, pp. 324–331.

[165] A. Lakhina, M. Crovella, and C. Diot, "Mining Anomalies Using Traffic Feature Distributions," in *ACM SIGCOMM Computer Communication Review*, vol. 35, no. 4. ACM, 2005, pp. 217–228.

[166] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A Density-based Algorithm for Discovering Clusters in Large Spatial Databases with Noise," in *Proceedings of 2nd International Conference on Knowledge Discovery and Data Mining (KDD'96)*, vol. 96, Portland, OR, USA, Aug. 1996, pp. 226–231.

[167] X. Zhu, "Semi-supervised Learning Literature Survey," *Computer Sciences Technical Report 1530*, vol. 52, pp. 55–66, 2010.

[168] N. Grira, M. Crucianu, and N. Boujemaa, "Unsupervised and Semi-supervised Clustering: A Brief Survey," in *7th ACM SIGMM international workshop on Multimedia information retrieval*, New York, NY, USA, Aug. 2004, pp. 1–12.

[169] N. Cardwell, S. Savage, and T. Anderson, "Modeling TCP Latency," in *Proceedings of the Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'00)*, vol. 3. Tel Aviv, Israel: IEEE, Mar. 2000, pp. 1742–1751.

[170] M. Duke, R. Braden, W. Eddy, and E. Blanton, "A Roadmap for Transmission Control Protocol (TCP) Specification Documents," RFC 4614 (Informational), Internet Engineering Task Force, Sep. 2006, updated by RFC 6247. [Online]. Available: http://www.ietf.org/rfc/rfc4614.txt

[171] K. Fall and S. Floyd, "Simulation-based Comparisons of Tahoe, Reno and SACK TCP," *ACM SIGCOMM Computer Communication Review*, vol. 26, no. 3, pp. 5–21, 1996.

[172] M. Allman, V. Paxson, and W. Stevens, "TCP Congestion Control," RFC 2581 (Proposed Standard), Internet Engineering Task Force, Apr. 1999, made obsolete by RFC 5681, updated by RFC 3390. [Online]. Available: http://www.ietf.org/rfc/rfc2581.txt

[173] V. Jacobson, "Congestion Avoidance and Control," in *ACM SIGCOMM Computer Communication Review*, vol. 18, no. 4. ACM, 1988, pp. 314–329.

[174] V. Jacobson, R. Braden, and D. Borman, "TCP Extensions for High Performance," RFC 1323 (Proposed Standard), Internet Engineering Task Force, May 1992. [Online]. Available: http://www.ietf.org/rfc/rfc1323.txt

[175] D. Leith and R. Shorten, "H-TCP: TCP for High-speed and Long-distance Networks," in *Proceedings of 2nd Workshop Protocols Fast Long Distance Networks (PFLDNeT)*, Argonne, IL, USA, Feb. 2004, pp. 1–16.

[176] Y. Tian, K. Xu, and N. Ansari, "TCP in Wireless Environments: Problems and Solutions," *IEEE Communications Magazine*, vol. 43, no. 3, pp. S27–S32, 2005.

[177] R. Braden, "Requirements for Internet Hosts - Communication Layers," RFC 1122 (Internet Standard), Internet Engineering Task Force, Oct. 1989, updated by RFCs 1349, 4379, 5884, 6093, 6298, 6633, 6864. [Online]. Available: http://www.ietf.org/rfc/rfc1122.txt

[178] L. S. Brakmo and L. L. Peterson, "TCP Vegas: End to End Congestion Avoidance on a Global Internet," *IEEE Journal on Selected Areas in Communications*, vol. 13, no. 8, pp. 1465–1480, 1995.

[179] S. Floyd, T. Henderson, and A. Gurtov, "The NewReno Modification to TCP's Fast Recovery Algorithm," RFC 3782 (Proposed Standard), Internet Engineering Task Force, Apr. 2004, made obsolete by RFC 6582. [Online]. Available: http://www.ietf.org/rfc/rfc3782.txt

[180] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, "TCP Selective Acknowledgment Options," RFC 2018 (Proposed Standard), Internet Engineering Task Force, Oct. 1996. [Online]. Available: http://www.ietf.org/rfc/rfc2018.txt

[181] B. Moraru, F. Copaciu, G. Lazar, and V. Dobrota, "Practical analysis of TCP implementations: Tahoe, Reno, NewReno," in *RoEduNet International Conference*, vol. 1, Sibiu, Romania, Jan. 2003, pp. 125–138.

[182] L. Xu, K. Harfoush, and I. Rhee, "Binary Increase Congestion Control (BIC) for Fast Long-distance Networks," in *Proceedings of the IEEE INFOCOM 04 Conference*, vol. 4, Hong Kong, Mar. 2004, pp. 2514 – 2524.

[183] K. Munir, M. Welzl, and D. Damjanovic, "Linux beats windows! or the worrying evolution of TCP in common operating systems," in *PFLDnet Workshop.*, Los Angeles, USA, Feb. 2007, pp. 43–48.

[184] S. Ha, I. Rhee, and L. Xu, "CUBIC: A New TCP-friendly Hgh-speed TCP Variant," *ACM SIGOPS Operating Systems Review*, vol. 42, no. 5, pp. 64–74, 2008.

[185] S. Floyd, "HighSpeed TCP for Large Congestion Windows," RFC 3649 (Experimental), Internet Engineering Task Force, Dec. 2003. [Online]. Available: http://www.ietf.org/rfc/rfc3649.txt

[186] S. Mascolo, C. Casetti, M. Gerla, M. Y. Sanadidi, and R. Wang, "TCP Westwood: Bandwidth Estimation for Enhanced Transport Over Wireless Links," in *Proceedings of the Seventh ACM International Conference on Mobile Computing and Networking (MobiCom'2001)*. Rome, Italy: ACM, Jul. 2001, pp. 287–297.

[187] C. Caini and R. Firrincieli, "TCP Hybla: A TCP Enhancement for Heterogeneous Networks," *International Journal of Satellite Communications and Networking*, vol. 22, no. 5, pp. 547–566, 2004.

[188] D. Hayes and G. Armitage, "Revisiting TCP Congestion Control Using Delay Gradients," in *Networking 2011*, ser. Lecture Notes in Computer Science, J. Domingo-Pascual, P. Manzoni, S. Palazzo, A. Pont, and C. Scoglio, Eds. Springer Berlin, Heidelberg, 2011, vol. 6641, pp. 328–341.

[189] ——, "Improved coexistence and loss tolerance for delay based TCP congestion control," in *IEEE 35th Conference on Local Computer Networks (LCN)*, Denver, Colorado, USA, Oct 2010, pp. 24–31.

[190] K. Tan, J. Song, Q. Zhang, and M. Sridharan, "A Compound TCP Approach for High-Speed and Long Distance Networks," in *Proceedings of the 25th IEEE International Conference on Computer Communications.*, April 2006, pp. 1–12.

[191] L. A. Grieco and S. Mascolo, "TCP Westwood and Easy RED to Improve Fairness in High-Speed Networks," in *Proceedings of the 7th IFIP/IEEE International Workshop on Protocols for High Speed Networks*, ser. PIHSN '02. London, UK, UK: Springer-Verlag, 2002, pp. 130–146.

[192] T. J. Shepard, "TCP Packet Trace Analysis," Cambridge, MA, USA, Tech. Rep., 1991. [Online]. Available: http://publications.csail.mit.edu/lcs/pubs/pdf/MIT-LCS-TR-494.pdf

[193] S. Ostermann, "tcptrace: A TCP Connection Analysis Tool," 2000. [Online]. Available: http://www. tcptrace. org

[194] A. Orebaugh, G. Ramirez, and J. Beale, *Wireshark & Ethereal network protocol analyzer toolkit*. Syngress, URL: http://www.wireshark.org, 2006. [Online]. Available: URL: http://www.wireshark.org

[195] B. Veal, K. Li, and D. Lowenthal, "New Methods for Passive Estimation of TCP Round-trip Times," in *Proceedings of the 6th International Conference on Passive and Active Network Measurement (PAM'05)*. Boston, MA, USA: Springer, Mar. 2005, pp. 121–134.

[196] S. Jaiswal, G. Iannaccone, C. Diot, J. Kurose, and D. Towsley, "Inferring TCP Connection Characteristics Through Passive Measurements," in *Proceedings of the IEEE INFOCOM 04 Conference*, vol. 3, Hong Kong, Mar. 2004, pp. 1582 –1592 vol.3.

[197] S. Katti, D. Katabi, C. Blake, E. Kohler, and J. Strauss, "M&M: A Passive Toolkit for Measuring, Tracking and Correlating Path Characteristics," MIT Computer Science and Artificial Intelligence Laboratary, Cambridge, MA, USA, Technical Report, Apr. 2004. [Online]. Available: http://mit.dspace.org/bitstream/handle/1721.1/30462/MIT-CSAIL-TR-2004-022.pdf?sequence=2

[198] M. Mirza, J. Sommers, P. Barford, and X. Zhu, "A Machine Learning Approach to TCP Throughput Prediction," *IEEE/ACM Transactions on Networking*, vol. 18, no. 4, pp. 1026 –1039, Aug. 2010.

[199] S. Katti, D. Katabi, C. Blake, E. Kohler, and J. Strauss, "MultiQ: Automated Detection of Multiple Bottleneck Capacities Along a Path," in *Proceedings of the ACM SIGCOMM'04 Conference on Internet Measurement*. Taormina, Sicily, Italy: ACM, Oct. 2004, pp. 245–250.

[200] P. Benko and A. Veres, "A Passive Method for Estimating End-to-End TCP Packet Loss," in *Proceedings of the IEEE Global Telecommunications Conference GLOBECOM'02*. Taipei, Taiwan: IEEE Communications Society, Nov. 2002, pp. 2609 – 2613 vol.3.

[201] V. Paxson, "Automated Packet Trace Analysis of TCP Implementations," in *Proceedings of the ACM SIGCOMM'97 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*. Cannes, France: ACM, Sep. 1997, pp. 167–179.

[202] M. Mellia, M. Meo, L. Muscariello, and D. Rossi, "Passive Analysis of TCP Anomalies," *Computer Networks*, vol. 52, no. 14, pp. 2663–2676, 2008.

[203] M. Dondo and J. Treurniet, "Investigation of a Neural Network Implementation of a TCP Packet Anomaly Detection System," Defence R&D Canada - Ottawa, Tech. Rep, May 2004. [Online]. Available: http://handle.dtic.mil/100.2/ADA436375

[204] J. Treurniet, "Detecting Low-profile Scans in TCP Anomaly Event Data," in *Proceedings of the 2006 International Conference on Privacy, Security and Trust (PST' 06)*. New York, NY, USA: ACM, Oct. 2006, pp. 1–8.

[205] R. Yuan, Z. Li, X. Guan, and L. Xu, "An SVM-based Machine Learning Method for Accurate Internet Traffic Classification," *Information Systems Frontiers*, vol. 12, no. 2, pp. 149–156, 2010.

[206] M. Mellia, M. Meo, L. Muscariello, and D. Rossi, "Passive Identification and Analysis of TCP Anomalies," in *Proceedings of the 2006 IEEE International Conference on Communications (ICC'06)*, vol. 2. Istanbul, Turkey: IEEE Communications Society, Jun. 2006, pp. 723 –728.

[207] Y. Zhao, Y. Chen, and D. Bindel, "Towards Unbiased End-to-End Network Diagnosis," in *ACM SIGCOMM Computer Communication Review*, vol. 36, no. 4. ACM, 2006, pp. 219–230.

[208] T. J. Hacker, B. D. Athey, and J. Sommerfield, "Experiences Using Web100 for End-to-End Network Performance Tuning," in *Proceedings of the 4th Visible Human Project Conference*, Keystone, CO, USA, Nov. 2002, pp. 555–563.

[209] J. Pahdye and S. Floyd, "On Inferring TCP Behavior," in *Proceedings of the ACM SIGCOMM'01 Conference on Applications, technologies, architectures, and protocols for computer communications*. ACM, Aug. 2001, pp. 287–298.

[210] M. Mathis, J. Heffner, P. O'Neil, and P. Siemsen, "Pathdiag: Automated TCP diagnosis," in *Proceedings of PAM 08*. Cleveland, OH: Springer-Verlag, Berlin, Apr. 2008, pp. 152–161.

[211] A. Medina, M. Allman, and S. Floyd, "Measuring the Evolution of Transport Protocols in the internet," *ACM SIGCOMM Computer Communication Review*, vol. 35, pp. 37–52, Apr. 2005.

[212] S. Jaiswal, G. Iannaccone, J. Kurose, and D. Towsley, "Formal Analysis of Passive Measurement Inference Techniques," in *Proceedings of the IEEE INFOCOM'06 Conference*. Barcelona, Spain: IEEE Communications Society, Apr. 2006, pp. 1 –12.

[213] M. Mellia, M. Meo, and L. Muscariello, "TCP anomalies: Identification and Analysis," in *Distributed Cooperative Laboratories: Networking, Instrumentation, and Measurements*. Amsterdam: Springer US, 2006, pp. 113–126.

[214] Web10g, "Web10Gig - Taking TCP Instrumentation to the Next Level," Web10g, San Jose, CA, Annual Report, 2011. [Online]. Available: https://web10g.org/index.php/papers-presentations?id=19

[215] L. Stewart and J. Healy, "Characterising the Behaviour and Performance of SIFTR v1. 1.0 report," Centre for Advanced Internet Architectures (CAIA), Swinburne University of Technology, Victoria, Australia, Tech. Rep., 2007. [Online]. Available: http://caia. swin. edu.au/reports/070824A/CAIA-TR-070824A.

[216] A. Coon, J. Radick, and H. Moore, "Automated Detection of TCP Anomalies," *Patent No: 11/888002, Hewlett Packard Company, Fort Collins, CO*, no. 11/888002, Jul. 2007.

[217] P. Krishnan, "System and Method for Automatically Diagnosing Protocol Errors from Packet Traces," *Patent No: 11/752379, Sunnyvale, CA, US*, no. 11/752379, Jul. 2007.

[218] D. Thaler, A. Gavrilescu, and T. Qian, "Method and Apparatus for Performing Network Diagnostics," *Patent No:11/118206, Microsoft Corporation, Redmond, WA*, no. 11/118206, Mar. 2008.

[219] T. Shon and J. Moon, "A Hybrid Machine Learning Approach to Network Anomaly Detection," *Information Sciences*, vol. 177, no. 18, pp. 3799–3821, Sep. 2007.

[220] L. Kuang and M. Zulkernine, "An Anomaly Intrusion Detection Method Using the CSI-KNN Algorithm," in *Proceedings of the 2008 ACM Symposium on Applied Computing*. New York, NY, USA: ACM, 2008, pp. 921–926.

[221] C. Widanapathirana, Y. Şekercioğlu, M. Ivanovich, P. Fitzpatrick, and J. Li, "Automated Inference System for End-To-End Diagnosis of Network Performance Issues in Client-Terminal Devices," *International Journal of Computer Networks & Communications*, vol. 4, no. 3, pp. 37–56, 2012.

[222] C. Widanapathirana, J. Li, Y. Şekercioğlu, M. Ivanovich, and P. Fitzpatrick, "Diagnosing Client Faults Using SVM-based Intelligent Inference from TCP Packet Traces," in *Proceedings of the 6th International Conference on Broadband Communications and Biomedical Applications (IB2COM 2011)*, Melbourne, Australia, Nov. 2011, pp. 68–73.

[223] L. Braun, A. Didebulidze, N. Kammenhuber, and G. Carle, "Comparing and Improving Current Packet Capturing Solutions Based on Commodity Hardware," in *Proceedings of SIGCOMM IMC 10*. Melbourne, Australia: ACM, New York, Nov. 2010, pp. 206–217.

[224] M. Mellia, A. Carpani, and R. L. Cigno, "TStat: TCP Statistic and Analysis Tool," in *Proceedings of QoS-IP 03*. Milano, Italy: Springer-Verlag, Berlin, Feb. 2003, pp. 145–157.

[225] X. Sun, H. Wang, J. Li, and Y. Zhang, "Satisfying Privacy Requirements Before Data Anonymization," *The Computer Journal*, vol. 55, no. 4, pp. 422–437, Apr. 2012.

[226] L. DiCioccio, R. Teixeira, and C. Rosenberg, "Measuring Home Networks with HomeNet Profiler," in *Passive and Active Measurement*. Springer, Berlin Heidelberg, 2013, vol. 7799, pp. 176–186.

[227] B. Settles, "Active Learning Literature Survey," *Computer Sciences Technical Report 1648, University of Wisconsin, Madison*, vol. 52, pp. 55–66, 2010.

[228] R. P. Lippmann, D. J. Fried, I. Graf, J. W. Haines, K. R. Kendall, D. McClung, D. Weber, S. E. Webster, D. Wyschogrod, and R. K. Cunningham, "Evaluating Intrusion Detection Systems: The 1998 DARPA Off-line Intrusion Detection Evaluation," in *Proceedings of the DARPA Information Survivability Conference and Exposition, 2000 (DISCEX'00)*, vol. 2. IEEE, 2000, pp. 12–26.

[229] R. Sommer and V. Paxson, "Outside the Closed World: On Using Machine Learning for Network Intrusion Detection," in *Proceedings of the IEEE Symposium on Security and Privacy (SP)*. IEEE, 2010, pp. 305–316.

[230] J. P. Theiler and D. M. Cai, "Resampling Approach for Anomaly Detection in Multispectral Images," in *Proceedings of APIE AeroSense 2003*. Orlando, FL, USA: International Society for Optics and Photonics, Apr. 2003, pp. 230–240.

[231] N. Abe, B. Zadrozny, and J. Langford, "Outlier Detection by Active Learning," in *Proceedings of the 12th International Conference on Knowledge Discovery and Data Mining (ACM SIGKDD)*. ACM, 2006, pp. 504–509.

[232] I. Steinwart, D. Hush, and C. Scovel, "A Classification Framework for Anomaly Detection," *Journal of Machine Learning Research*, vol. 6, pp. 211–232, Dec. 2005.

[233] P. Benko, G. Malicsko, and A. Veres, "A Large-scale, Passive Analysis of End-to-End TCP Performance Over GPRS," in *Proceedings of the IEEE INFOCOM 04 Conference*, vol. 3, Hong Kong, Mar. 2004, pp. 1882 –1892.

[234] M. Oo and M. Othman, "How Good Delayed Acknowledgement Effects Rate-Based Pacing TCP over Multi-hop Wireless Network," in *Proceedings of the 2009 International Conference on Signal Processing Systems*, Singapore, May 2009, pp. 464 –468.

[235] L. Rizzo, "Dummynet: A Simple Approach to the Evaluation of Network Protocols," *SIGCOMM Computer Communication Review*, vol. 27, no. 1, pp. 31–41, Jan. 1997. [Online]. Available: http://doi.acm.org/10.1145/251007.251012

[236] M. Carbone and L. Rizzo, "Dummynet Revisited," *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 2, pp. 12–20, 2010.

[237] A. Björck, *Numerical Methods for Least Squares Problems*. SIAM, Philadelphia, USA, 1996.

[238] P. Wasserman, *Advanced Methods in Neural Computing*. Wiley, New York, USA, 1993.

[239] D. Marquardt, "An Algorithm for Least-Squares Estimation of Nonlinear Parameters," *Journal of Society for Industrial and Applied Mathematics*, vol. 11, no. 2, pp. 431–441, 1963.

[240] F. Korn, B. Pagel, and C. Faloutsos, "On the Dimensionality Curse and the Self-similarity Blessing," *IEEE Transactions on Knowledge and Data Engineering*, vol. 13, no. 1, pp. 96–111, 2002.

[241] P. Sterlin, "Overfitting Prevention with Cross-validation," Master's thesis, University Pierre and Marie Curie (Paris VI), Paris, France, 2007.

[242] A. Janecek, W. Gansterer, M. Demel, and G. Ecker, "On the Relationship Between Feature Selection and Classification Accuracy," in *Proceedings of the Journal of Machine Learning and Research Conference*, vol. 4, 2008, pp. 90–105.

[243] E. P. Xing, M. I. Jordan, and R. M. Karp, "Feature Selection for High-dimensional Genomic Microarray Data," in *Proceedings of the Eighteenth International Conference on Machine Learning*, ser. ICML '01. Williamstown, MA, USA: Morgan Kaufmann Publishers Inc., Jun. 2001, pp. 601–608.

[244] S. Das, "Filters, Wrappers and a Boosting-Based Hybrid for Feature Selection," in *Proceedings of the Eighteenth International Conference on Machine Learning*, ser. ICML '01. Williamstown, MA, USA: Morgan Kaufmann Publishers Inc., Jun. 2001, pp. 74–81. [Online]. Available: http://dl.acm.org/citation.cfm?id=645530.658297

[245] W. Zhu, X. Wang, Y. Ma, M. Rao, J. Glimm, and J. S. Kovach, "Detection of Cancer-specific Markers amid Massive Mass Spectral Data," *Proceedings of the National Academy of Sciences*, vol. 100, no. 25, pp. 14 666–14 671, 2003.

[246] S. Abe, *Support Vector Machines for Pattern Classification*. Springer, New York, USA, 2010.

[247] H. W. Kuhn and A. W. Tucker, "Nonlinear Programming," in *Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability*. Berkeley, Calif: University of California Press, 1951, pp. 481–492.

[248] C. Widanapathirana, J. Li, Y. Şekercioğlu, M. Ivanovich, and P. Fitzpatrick, "Intelligent Automated Diagnosis of Client Device Bottlenecks in Private Clouds," in *Proceedings of IEEE UCC 11*. Melbourne, Australia: IEEE, New York, Dec. 2011, pp. 261 –266.

[249] S. L. Chiu, "Fuzzy Model Identification Based on Cluster Estimation," *Journal of Intelligent and Fuzzy Systems*, vol. 2, no. 3, pp. 267–278, 1994.

[250] J. C. Bezdek, "Fuzzy Mathematics in Pattern Classification," *PhD Dissertion, Applied Mathematics Center, Cornell University*, 1973.