Lancaster University

Doctoral Thesis

Anomaly Detection for Resilience in Cloud Computing Infrastructures

Author:
Syed Noorulhassan Shirazi

Supervisors:
Professor David Hutchison & Dr. Andreas Ulrich Mauthe

A thesis submitted in fulfilment of the requirements for the degree of Doctor of Philosophy

in the

Communications and Networking
School of Computing and Communications

November 16, 2017

Declaration of Authorship

I, Syed Noorulhassan Shirazi, declare that this thesis titled, "Anomaly Detection for Resilience in Cloud Computing Infrastructures" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

"*The intellect is what arrives at what is correct through reasoning, and recognizes what has not yet happened through what has already taken place.*"

Ali Ibn Talib (AS)

LANCASTER UNIVERSITY

Abstract

Faculty of Science and Technology
School of Computing and Communications

Doctor of Philosophy

Anomaly Detection for Resilience in Cloud Computing Infrastructures

by Syed Noorulhassan Shirazi

Cloud computing is a relatively recent model where scalable and elastic resources are provided as optimized, cost-effective and on-demand utility-like services to customers. As one of the major trends in the IT industry in recent years, cloud computing has gained momentum and started to revolutionise the way enterprises create and deliver IT solutions. Motivated primarily due to cost reduction, these cloud environments are also being used by Information and Communication Technologies (ICT) operating Critical Infrastructures (CI). However, due to the complex nature of underlying infrastructures, these environments are subject to a large number of challenges, including mis-configurations, cyber attacks and malware instances, which manifest themselves as anomalies. These challenges clearly reduce the overall reliability and availability of the cloud, i.e., it is less resilient to challenges. Resilience is intended to be a fundamental property of cloud service provisioning platforms. However, a number of significant challenges in the past demonstrated that cloud environments are not as resilient as one would hope. There is also limited understanding about how to provide resilience in the cloud that can address such challenges. This implies that it is of utmost importance to clearly understand and define what constitutes the correct, normal behaviour so that deviation from it can be detected as anomalies and consequently higher resilience can be achieved. Also, for characterising and identifying challenges, anomaly detection techniques can be used and this is due to the fact that the statistical models embodied in these techniques allow the robust characterisation of normal behaviour, taking into account various monitoring metrics to detect known and unknown patterns. These anomaly detection techniques can also be applied within a resilience framework in order to promptly provide indications and warnings about adverse events or conditions that may occur. However, due to the scale and complexity of cloud, detection based on continuous real time infrastructure monitoring becomes challenging. Because monitoring leads to an overwhelming volume of data, this adversely affects the ability of the underlying detection mechanisms to analyse the data. The increasing volume of metrics, compounded with complexity of infrastructure, may also cause low detection accuracy. In this thesis, a comprehensive evaluation of anomaly detection techniques in cloud infrastructures is presented under typical elastic behaviour. More specifically, an investigation of the impact of live virtual machine migration on state of the art anomaly detection techniques is carried out, by evaluating live migration under various attack types and intensities. An initial comparison concludes that, whilst many detection techniques have been proposed, none of them is suited to work within a cloud operational context. The results suggest that in some configurations anomalies are missed and some configuration anomalies are wrongly classified. Moreover, some of these approaches have been shown to be sensitive to parameters of the

datasets such as the level of traffic aggregation, and they suffer from other robustness problems. In general, anomaly detection techniques are founded on specific assumptions about the data, for example the statistical distributions of events. If these assumptions do not hold, an outcome can be high false positive rates. Based on this initial study, the objective of this work is to establish a light-weight real time anomaly detection technique which is more suited to a cloud operational context by keeping low false positive rates without the need for prior knowledge and thus enabling the administrator to respond to threats effectively. Furthermore, a technique is needed which is robust to the properties of cloud infrastructures, such as elasticity and limited knowledge of the services, and such that it can support other resilience supporting mechanisms. From this formulation, a cloud resilience management framework is proposed which incorporates the anomaly detection and other supporting mechanisms that collectively address challenges that manifest themselves as anomalies. The framework is a holistic end-to-end framework for resilience that considers both networking and system issues, and spans the various stages of an existing resilience strategy, called (D^2R^2+DR). In regards to the operational applicability of detection mechanisms, a novel Anomaly Detection-as-a-Service ($ADaaS$) architecture has been modelled as the means to implement the detection technique. A series of experiments was conducted to assess the effectiveness of the proposed technique for $ADaaS$. These aimed to improve the viability of implementing the system in an operational context. Finally, the proposed model is deployed in a European Critical Infrastructure provider's network running various critical services, and validated the results in real time scenarios with the use of various test cases, and finally demonstrating the advantages of such a model in an operational context. The obtained results show that anomalies are detectable with high accuracy with no prior-knowledge, and it can be concluded that $ADaaS$ is applicable to cloud scenarios for a flexible multi-tenant detection systems, clearly establishing its effectiveness for cloud infrastructure resilience.

Acknowledgements

This thesis would not have been possible without the guidance and the help of several individuals who in one way or another contributed and extended their valuable assistance in the preparation and completion of this research.

First, I would like to express my gratitude to Prof. David Hutchison, for his technical guidance and moral support from the very early stage of this research. Thank you for believing in me and providing all the needed support. Secondly, I would like to thank Dr. Andreas Mauthe, my co-supervisor, without his knowledge and assistance, this research would not have been successful.

I would like to express special thanks to Dr. Steven Simpson who always there when I needed him. His assistance and personal guidance have been of great value on both an academic and personal level, for which I am extremely grateful. Thank you for giving me the confidence and help to work out difficult situations. Also, I would like to thank Prof. Plamen Angelov for providing insightful comments and suggestions he made in reference to Chapter 5 of this work.

I have been extremely lucky to work and co-author with many great colleagues. I would like to thank every one at InfoLab21, for the friendly atmosphere, in particular Dr. Syed Asad Ali Naqvi, Dr. Aurangzeb Khan Niazi, Dr. Egen Alberto Shchaffer Filho, Dr. Angelos Marnerides, Mike Watson, Dr. Ethem Ibrahim Bagci, Dr. Antonios Gouglidis, Dr. Arsham Farshad, Dr. Asmar Khan, Ms. Liz Redburn, Ms. Carol Airey and Ms. Barbara Hickson.

I would also like to thank the brilliant guys of EU-funded SECCRIT project for their assistance in many technical aspects surrounding this work. Thanks to all for the time spent together and for having shared your knowledge, this was a valuable and great experience for me. A special thanks go to my late uncle; Dr. Syed Tasnim Hussain Shah, greatly missed, who motivated me for PhD and always did his best to lend a helping hand whenever I needed it. I am especially grateful to Dr. Nassar Ikram for his guidance and support during my time at National Systems. I am grateful to my parents Mr. Syed Sajjad Hussain Shah and Mrs. Syeda Attiya Shaheen for their faith in me and allowing me to be as ambitious as I wanted. Also, I thank my brothers Qammar and Muwahib and my sisters Shazia, Tahira, Youkhaneez and Asma for their unlimited support and prayers. Lastly, my wife Kanza and our children, Khushan and Khubaib for unwavering love for me during these hard working years.

For those who have contributed but not mentioned, please accept my thanks.

Contents

List of Figures

List of Tables

List of Abbreviations

AD	**A**nomaly **D**etection
ADaaS	**A**nomaly **D**etection **a**s **a** **S**ervice
ATC	**A**ir **T**raffic **C**ontrol
AH	**A**nomaly **H**igh
AL	**A**nomaly **L**ow
AM	**A**nomalous **M**igration
AMQP	**A**dvanced **M**essage **Q**ueuing **P**rotocol
CC	**C**loud **C**omputing
CI	**C**ritical **I**nfrastructures
CPU	**C**entral **P**rocessing **U**nit
COTS	**C**ommercial **O**ff **T**he **S**helf
CRMF	**C**loud **R**esilience **M**anagement **F**ramework
COE	**C**oordination and **O**rganisation **E**ngine
D^2R^2+**DR**	**D**efend, **D**etect, **R**emediate, **R**ecover, **D**iagnose and **R**efine
DoS	**D**enial **o**f **S**ervice
DDoS	**D**istributed **D**enial **o**f **S**ervice
DF	**D**eployment **F**unction
DCE	**D**ata **C**ollection **E**ngine
ENISA	**E**uropean **N**etwork and **I**nformation **S**ecurity **A**gency
ECA	**E**vent **C**ondition **A**ction
FP	**F**alse **P**ositive
FN	**F**alse **N**egative
FPR	**F**alse **P**ositive **R**ate
FGA	**F**ine **G**rain **A**nalysis
GMM	**G**aussian **M**ixture **M**odel
HTTP	**H**yper **T**ext **T**ransfer **P**rotocol
HIPPA Act	**H**ealth **I**nsurance **P**ortability and **A**ccountability **A**ct of 1996
IT	**I**nformation **T**echnology
ICT	**I**nformation and **C**ommunications **T**echnologies
IPTV	**I**nternet **P**rotocol **TV**
IaaS	**I**nfrastructure as a **S**ervice
KVM	**K**ernel-based **V**irtual **M**achine
LAN	**L**ocal **A**rea **N**etworking
MDout	**M**igration direction outward
MDin	**M**igration direction inward
MT0	**M**igration targetedness; attack targets the VM that does not migrate

MT1	**M**igration targetedness; VM experiencing the attack is migrating
NAD	**N**etwork **A**nomaly **D**etection
NAE	**N**etwork **A**nalysis **E**ngine
NS	**N**etwork **S**can
NM	**N**ormal **M**igration
OS	**O**perating **S**ystem
PS	**P**ort **S**can
PCA	**P**rincipal **C**omponent **A**nalysis
PE	**P**olicy **E**ngine
PaaS	**P**latform as a **S**ervice
QoS	**Q**uality of **S**ervice
RM	**R**esilience **M**anager
ROC	**R**eciever **O**perating **C**haracteristics
RTO	**R**ecovery **T**ime **O**bjective
RPO	**R**ecovery **P**oint **O**bjective
SLA	**S**ervice **L**evel **A**greement
SVM	**S**upport **V**ector **M**achine
SAE	**S**ystem **A**nalysis **E**ngine
SaaS	**S**oftware as a **S**ervice
TP	**T**rue **P**ositive
TN	**T**rue **N**egative
TPR	**T**rue **P**ositive **R**ate
TIMS	**T**enant **I**nfrastructure **M**anagement **S**ystem
QEMU	**Q**uick **EMU**lator
VM	**V**irtual **M**achine

To my wife Kanza, and our children Khushan and Khubaib whose support, love and encouragement have inspired me to pursue and complete the work.

List of Publications

Parts of the research documented in this thesis appeared in journals and conference proceedings as listed below:

1. Syed Noorulhassan Shirazi, Steven Simpson, Antonios Gouglidis, Andreas Ulrich Mauthe, and David Hutchison. Anomaly Detection in the Cloud using Data Density. In IEEE Cloud 2016, 9th IEEE International Conference on Cloud Computing, 4 2016.

2. Syed Noorulhassan Shirazi, Steven Simpson, Kanza Noor Syeda, Andreas Ulrich Mauthe and David Hutchison, 2016, Towards Policy Refinement for Resilience Management in Cloud. In 2016 8th International Workshop on Resilient Networks Design and Modeling (RNDM). IEEE, pp. 260-266, International Workshop on Resilient Networks Design and Modeling, 13-15 September.

3. Michael Watson, Syed Noorulhassan Shirazi, Angelos Marnerides, and Andreas Ulrich Mauthe. Malware Detection in Cloud Computing Infrastructures. IEEE Transactions on Dependable and Secure Computing, 13(2):192-205, 3 2016.

4. Syed Noorulhassan Shirazi, Steven Simpson, Simon Oechsner, Andreas Ulrich Mauthe, and David Hutchison. A Framework for Resilience Management in the Cloud. e & i Elektrotechnik und Informationstechnik, pages 1-11, 2015.

5. Ioannis M. Stephanakis, Ioannis P. Chochliouros, and Evangelos Sfakianakis and Syed Noorulhassan Shirazi. Anomaly Detection in Secure Cloud Environments using a Self Organizing Feature Map (SOFM) Model For Clustering sets of R-ordered Vector-Structured Features. ACM, 2015.

6. Syed Noorulhassan Shirazi, Steven Simpson, Angelos Marnerides, Michael Watson, Andreas Ulrich Mauthe, and David Hutchison. Assessing the Impact of Intra-Cloud Live Migration on Anomaly Detection. In Cloud Networking (CloudNet), 2014 IEEE 3rd International Conference on, pages 52-57, Oct 2014.

7. Syed Noorulhassan Shirazi, Alberto Schaeffer-Filho, and David Hutchison. Attack Pattern Recognition through Correlating Cyber Situational Awareness in Computer Networks, pages 125-134. Springer International Publishing, 3 2014.

8. Michael Watson, Syed Noorulhassan Shirazi, Angelos Marnerides, Andreas Ulrich Mauthe, and David Hutchison. Towards a Distributed, Self-organising Approach to Malware Detection in Cloud Computing. In Wilfried Elmenreich, Falko Dressler, and Vittorio Loreto, editors, Self-Organizing Systems, volume 8221 of Lecture Notes in Computer Science, pages 182-185. Springer Berlin Heidelberg.

9. Angelos Marnerides, Michael Watson, Syed Noorulhassan Shirazi, Andreas Ulrich Mauthe, and David Hutchison. Malware Analysis in Cloud Computing: Network and System Characteristics. In Globecom Workshops (GC Wkshps), 2013 IEEE, pages 482-487, Dec 2013.

10. Syed Noorulhassan Shirazi, Michael Watson, Angelos Marnerides, Andreas Ulrich Mauthe, and David Hutchison. A Multilevel Approach Towards Challenge Detection in Cloud Computing. In Cyberpatterns 2013 : Proceedings of the Second International Workshop on Cyberpatterns . Oxford: Oxford Brookes University. 2013. p. 103-107.

11. Syed Noorulhassan Shirazi, Alberto Schaeffer-Filho, and David Hutchison. Service Level Agreement Monitoring for Resilience in Computer Networks. In 13th Annual Post Graduate Symposium on The Convergence of Telecommunications, Networking and Broadcasting, Liverpool John Moores University, 2012.

Chapter 1

Introduction

Cloud computing has transformed and revolutionized the way enterprises create and deliver IT solutions. It is fast becoming the preferred standard for running company IT infrastructure, from the smallest of organisations to the largest of enterprises. Clearly, it has matured as a technology, to the extent that companies like Amazon, Google, Ebay etc., heavily invest in building up cloud infrastructures to support their service provisioning. Many organizations now face question as to when and how to "cloudify" their existing IT infrastructures to cloud computing environments. According to a recent survey conducted by Cloud Industry Forum (CIF)[1], 63% of UK businesses are planning to move their entire IT infrastructures to the cloud in near future. CIF recorded the opinion of 250 senior IT and business decision makers at the end of 2015 and found that 78% of UK organisations are already using the cloud. This is an increase of 53% from when the research was first conducted in 2011 and it is predicted that by 2018, this will increase to 85%[2].

This emerging popularity of cloud computing for service provision is mainly due to cost savings, scalability and on-demand resource provision enabled by virtualization technology. Moreover, because most cloud providers are specialised in hardware and software technologies, cloud users can be relived from keeping a large team of in-house experts for infrastructure maintenances. Motivated primarily by economies of scale, these cloud environments are also being used by sectors operating critical infrastructures [112, 107, 22, 110, 133].

In spite of these advantages, a shift to the cloud brings its own challenges. The migration of both data and applications outside the administrative domain of customers into a shared environment imposes transversal, functional problems across distinc platforms and technologies. A recent work [99] provides a contemporary discussion of the most relevant functional problems associated with cloud computing, mainly from the network perspective. In cloud environments users will have less control over the hardware, the software and also the data. The loss of control over data and lack of transparency

[1] https://www.cloudindustryforum.org/
[2] http://www.telegraph.co.uk/business/ready-and-enabled/cloud-computing/

give rise to many security concerns, which cause uncertainty for organizations that want to "cloudify" their IT assets. This is highlighted in recent report published by Vision which shows increasing customer reluctance to move to clouds due to security concerns [130]. Similarly, another recent study by Alert Logic says that application attacks aimed at cloud deployments grew 45% over one year [82]. The complexity of the underlying infrastructures also introduce a number of challenges, including misconfiguration, and malwares. Further, when cloud environments fail, they incur significant costs. For instance, on 29th June 2010, `amazon.com` experienced hours of anomalous behaviour whereby customers were unable to place orders. Based on their 2010 quarterly revenues, such downtime cost Amazon up to $1.75 million per hour. Most recently in another serious incident, on October 21st, 2016, `dyn.com` suffered a high speed Distributed Denial-of-Service (DDoS) attack which affected numerous websites, but the biggest victims are the enterprises that rely on Software as a Service (SaaS) for critical business operations. Moreover, in regards to cloud based critical services, if disrupted, would have serious impact on the health, safety, security or economic well-being of citizen or the effective functioning of governments [85, 20].

1.1 Resilience in Cloud

The potentially huge costs of failure of cloud services, in terms of money, safety, and security, drive the need for resilience in the cloud. Resilience is defined as:

"*The ability of a system to provide an acceptable level of service in the light of various challenges [132]*".

The *acceptable level of service* in the above definition depends on user expectation. Today, users want instantaneous access to information that is available around the clock – the *always-on, always-available* service. However, due to the presence of various challenges, resilience is often evaluated through two key metrics: 1) Recovery time objective (RTO) is the duration of time within which a system must be restored after a disruption to avoid a break in business continuity, and 2) Recovery point objective (RPO) is the nearest point to where a system may be recovered after a disruption [86]. Further, resilience is concerned not only with the *Availability* of services but also with maintaining the *Confidentiality* and *Integrity* of the information in the face of challenges.

Resilience in terms of the three criteria of Confidentiality, Integrity, and Availability (CIA triad), is intended to be a fundamental property of cloud service provisioning platforms. However, the innate and often desirable properties of cloud environments such as elasticity, dynamicity, and scalability

make problematic implementing the standard resilience solutions used in conventional enterprise systems, specifically:

- Elasticity, in cloud computing refers to the dynamic adaptation of capacity to meet demand. It is defined as "the degree to which a system is able to adapt to workload changes by provisioning and de-provisioning resources in an autonomic manner, such that at each point in time the available resources match the current demand as closely as possible" [61, 60]. Providing resilience and service guarantees in the presence of various challenges requires resources in terms of redundant storage for backup-recovery and additional network capacity etc. However, due to varying customer workloads and the requirement of elasticity in the cloud, resource allocation and availability is constantly changing making it difficult to provide resilience and service guarantees. Service providers are faced with the challenge of determining how an application on the cloud should be configured, and how much resources in terms of CPUs, memory, storage, bandwidth, etc. should first be allocated and then added or removed pro actively for that application to satisfy its Service Level Agreements (SLA) and requirements for dependability and responsiveness, without under provisioning or over provisioning resources.

- Virtualization in cloud computing also introduces new sources of threats and failure. Complex interactions between the applications, and workloads sharing the same physical infrastructure make it difficult to predict demands on that infrastructure. Also, the provisioning of virtual resources reduces control over the underlying physical hardware. The difficulty of predicting interactions and adapting the system accordingly, makes it hard to provide dependability guarantees in terms of availability and responsiveness as well as resilience to external perturbations such as security attacks [75].

- Cloud environments achieve higher utilization of physical resources through the consolidation of workloads. However, this makes them more vulnerable to threats resulting from unpredictable resource demands as well as operational failures, such as, hardware and software failures, unforeseen load fluctuations, and network attacks.

- The geographical distribution in a cloud based service framework, introduces various dependencies that impede the satisfaction of SLA. This distribution may be between clients and services, as well as the physical infrastructure behind services. Cloud providers are faced with the question of how to assure the availability and reliability (i.e resilience) of services that are distributed over the Internet? The authors in [145]

have shown, through a series of stress tests, that response times of infrastructure-on-demand services offered by Amazon, Google and Microsoft varied by a factor of twenty depending on the time of day the services were accessed. Geographical distribution may also cause legal and privacy implications when data is hosted on outsourced and shared infrastructure that is in a different legal jurisdiction than the owner of the data. Privacy clauses in legislation such as the Health Insurance Portability and Accountability Act (HIPAA)[3] and the Telecommunications Act of 1996[4], as well as financial regulations such as the Sarbanes-Oxley Act[5] obstruct the applicability of cloud solutions in their respective industries, and may adversely affect resilience.

Cloud environments are multifaceted, where services may be provided by an assortment of heterogeneous platforms, and resources are shared between entities that engage in a wide range of behaviours. The applications running in the cloud may employ best practices to varying degrees, thus, increasing the risk of unforeseen threats. These risks are exacerbated due to multiple administrative domains between the application and infrastructure operators, which reduce end-to-end system visibility. Problem detection and diagnosis is made more difficult when error propagation information is not readily visible. The detection of problems is an integral part of all resilience strategies, such as D^2R^2+DR [132]. However, due to the diversity of platforms and applications in a cloud environment domain, application, or platform specific detection strategies will not work. What is needed is a generalised method of detecting problems.

1.2 Anomaly Detection in Cloud

Anomaly detection systems are generalised systems that identify events that appear to be anomalous with respect to normal system behaviour. They get a model of normal system behaviour and issue alerts whenever the behaviour changes from the norm. The underlying assumption in anomaly detection is that such changes are normally caused by malicious or disrupting events. Anomaly detection has been studied within diverse research areas and application domains.

Despite the usefulness of anomaly detection, applying it operationally in the cloud computing context involves many other challenges with regards to performance and scalability. The on-demand provisioning nature of cloud, necessitates that anomaly detection in the cloud be based on real-time monitoring. Further, real-time monitoring demands can also change significantly

[3] http://www.dhcs.ca.gov/formsandpubs/laws/hipaa/Pages/1.10HIPAATitleInformation.aspx

[4] https://www.fcc.gov/general/telecommunications-act-1996

[5] http://www.soxlaw.com/

over time. Hence, the detection should not only achieve high scalability, but also embrace changes in monitoring demands. The detection must also provide good multi-tenancy support to ensure that multiple tenants/providers can use anomaly detection at the same time. During an attempt to to address these challenges, the analysis discovered that the applicability and performance of state-of-the-art anomaly detection techniques in cloud environments were not well understood at the time, and there were no surveys discussing their merits and demerits in the cloud context.

To address this problem an experimental evaluation of state-of-the-art anomaly detection techniques is carried out to assess their monitoring and detection capabilities in multi-tenant cloud infrastructures. More specifically, the impact of elasticity is investigated, on contemporary anomaly detection techniques, namely Principle Component Analysis (PCA), K-means (KM), Naïve Bayesian (NB) and Expectation-Maximization algorithm for Gaussian Mixture Models (EMGM). These techniques are evaluated by quantifying the impact of live migration, under various attack types and intensities, on their results. Live intra-cloud virtual machine (VM) migration was studied because it is a widely used technique for efficient resource management employed within modern cloud infrastructures. Additionally, to understand how migration is represented or visible at hypervisor level, and then to what extent VM migration, which is generally representative for normal traffic, can be incorrectly classified as an anomaly – false positive. Conversely, certain anomalies may be missed because of VM migration – false negative. Further, new and intrinsic capabilities of cloud computing introduce a number of novel security concerns, such as, Hypervisor attacks and application deployment misconfiguration. Further to analyse the implication of these novel security concerns on the performance of anomaly detection mechanisms.

The experimental evaluation shows that elasticity of the cloud makes the traffic distribution observed by anomaly detection techniques unstable and makes it difficult to predict normal behaviour. The degradation in the performance for these detectors is evident in their Receiver Operating Characteristics (ROC) curves, when intra-cloud live migration is initiated while VMs are under attack, such as, netscan (NS), portscan (PS) or denial-of-service (DoS). Most anomaly detection techniques require problem-specific parameters to be predefined in advance, impairing their use in real-time detection [72, 141, 97]. Further, due to the scale and complexity of cloud computing, conventional anomaly detection methods require monitoring metrics with high dimensionality. In this context, real-time infrastructure monitoring becomes challenging, because it leads to overwhelming volumes of data impairing the detector's efficacy [125]. State-of-the-art anomaly detection techniques are not robust to the properties of cloud infrastructures, such as

elasticity and limited knowledge of the services that are running. If assumptions built into the technique about the cloud data do not hold, it could result in the anomaly detection technique producing high false positive and negative rates. In such cases, the reliability of the technique is reduced, which may prevent it from meeting the requirements of a larger resilience strategy.

1.3 Objectives

The section above have outlined some important open issues related to the security and dependability of cloud environments, and the need for developing novel anomaly detection techniques to facilitate resilience in the cloud. This leads to consider the following key research question:

Can the problem of anomaly detection in cloud be effectively addressed in a way that contributes to a systematic and coordinated resilience approach?

The above general research question was addressed by means of the following specific research objectives:

- Derive an anomaly detection technique that is both effective against cloud specific security concerns, and robust to the challenges of elasticity, evolving workload patterns, and limited knowledge about services in the cloud.
- Derive a generalised method of implementing resilience in cloud computing environments, which incorporates anomaly detection to offers policy driven resilience strategies at different levels of cloud infrastructure.
- Operationalise and assess, in the real world, the effectiveness of anomaly detection as part of a resilience strategy in the cloud context.

In service to these objectives, the following technical contributions is produced:

Real time Anomaly Detection Technique (ADT)

The knowledge, about challenges and shortcomings of existing anomaly detection techniques in the cloud context, gained through experimental study has been used to develop and design a novel anomaly detection algorithm based on data density. The density is computed recursively, so the technique is memory-less, light weight, and unsupervised, and therefore suitable for cloud environments where metrics are monitored in large volumes and in real-time. The proposed algorithm is a self-learning process that builds a dynamically evolving information model of normality. This allows it to be

robust to the challenges of limited knowledge about services, and the evolving workload patterns of multiple applications. The efficacy of the proposed technique is demonstrated in the face of elastic behaviour, using a dataset created in the cloud testbed. The dataset consists of feature vectors obtained from a physical cloud testbed network experiencing migration under controlled traffic conditions. The dataset models scenarios combining normal network use with network-based attacks of various types and intensities. The obtained results, which include precision, recall, accuracy, F-score and G-score, show that network level attacks are detectable with better accuracy by using the proposed algorithm than by means of conventional anomaly detection methods.

Cloud Resilience Management Framework (CRMF)

Having defined a novel anomaly detection algorithm for the cloud environment, the next challenge was to determine how this anomaly detector could be incorporated in a resilience strategy.

To address this challenge, a cloud resilience management framework is presented, which models and then applies anomaly detection and other supporting mechanisms through an existing resilience strategy (D^2R^2+DR) in the cloud operating context. These mechanisms collectively address the challenges that manifest themselves as anomalies. The resilience framework uses an end to end feedback loop that allows remediation to be integrated with the existing cloud management system. By controlling the various mechanisms and by allowing anomaly detection to be deployed at different levels of the cloud, the framework enables policy-driven remediation services and cloud infrastructure management to ensure that desired security and resilience requirements of customers and tenants are being met.

Anomaly Detection as a Service (ADaaS)

For operationalising the proposed anomaly detection mechanisms, the "Anomaly Detection as a Service ($ADaaS$)" model is designed and implemented, which a cloud provider can offer as part of their infrastructure to their tenants and customers, to counteract threats. The $ADaaS$ model is further evaluated in a real world cloud environment hosting critical services of a European Utility provider. Deploying $ADaaS$ in a real setting shows that the proposed solution can capture real world applications interactions and can yield useful results.

1.3.1 Testbeds

In the context of the above stated objectives, different cloud testbeds have been set up to carry out the experimentation, implementation and evaluation.

For the experimental study to analyse the effect of migration the Xen[6] based testbed is established with two hosts serving as nodes for running the multiple VMs. The idea behind using Xen is to have a lightweight, special-purpose hypervisor that is designed for producing small VMs which can populate cloud with minimal hardware, as needed by the experiments. Migration is used as a test case of elasticity for experiments and is achieved with *libvirt*. The dataset obtained from this testbed is used to quantify the impact of migration on anomaly detection as well as evaluation of the proposed anomaly detection techniques (Chapter 5).

For making the anomaly detection mechanisms operational, the anomaly detection technique is implemented as a service (see Chapter 6) for OpenStack. OpenStack is a popular open source cloud computing platform, and it is chosen because it offers various APIs needed to implement the service. Therefore for evaluation of the the implemented service OpenStack testbed is set up; this consists of three physical hosts which serve as a controller and two compute nodes, running the latest version of OpenStack (Mitaka). The compute node hosts multiple VMs to simulate real cloud operations and also to generate background traffic during experiments.

A similar testbed was hosted in one of Europe's leading telecommunication providers' research labs to further evaluate the integration of the service to the real cloud environment. The two use cases were developed to identify the needs of tenant and testbed is used to demonstrate how anomaly detection as a service can deal with the needs of tenants related to security and resilience.

1.4 Thesis Structure

The novel contributions to literature, and how they relate to one another and to the structure of the thesis, are presented in Figure 1.1.

The remainder of this thesis is organized as follows: Chapter 2 covers relevant background information for this research and provides a detailed literature study on anomaly detection in cloud environments. Chapter 3 describes the experimental evaluation of state-of-the-art anomaly detection techniques in the context of live virtual machine migration, and under various attack types and intensities. Informed by the issues identified in the experimental evaluation (chapter 3), Chapter 4 describes the design and evaluation of a novel real-time data density based anomaly detection technique that addresses many of the shortcomings of state-of-the-art anomaly detection methods in the cloud context. In Chapter 5 the design and qualitative evaluation of a cloud resilience management framework is presented that uses novel anomaly detection technique as its detection component. The design of

[6] https://www.xenproject.org/users/cloud.html

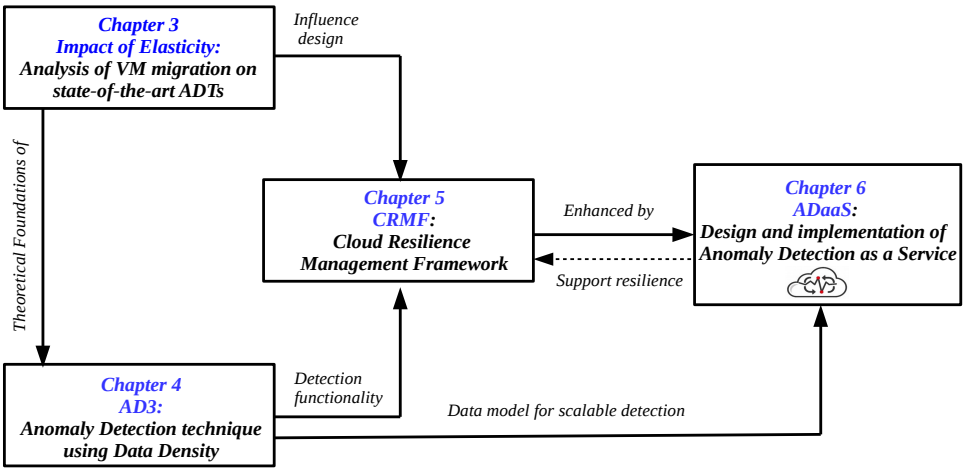


FIGURE 1.1: The research components

the resilience management framework is also informed by the challenges identified through the experimental evaluation. Chapter 6 presents the design, implementation, and experimental results of the *ADaaS* model that operationalises anomaly detection techniques and enables a cloud provider to offer detection services based on security and resilience requirements of its customers and tenants. Finally, Chapter 7 summarizes the research conducted and provides future work directions.

Chapter 2

Background and Related Work

The objective of this chapter is threefold. First, to describe core concepts and background information in the field of cloud computing by providing an overview of cloud classifications, models and features. Second, to discuss security and resilience as a cloud need and consider the positioning of resilience mechanisms (such as anomaly detection) in an overall cloud architecture. Finally, to give background on anomaly detection and summarise the state of the art of available anomaly detection for cloud environments.

2.1 Cloud Computing

In recent years, cloud computing has emerged as a widely accepted paradigm in computing systems, where scalable and elastic IT-related capabilities are provided as a service to external customers using Internet. Although service delivery is not a new concept, cloud computing differs in many ways from traditional IT outsourcing practices. It helps enterprises to create and deliver IT solutions in a more flexible and cost-effective way and therefore it is considered as a major evolution of e-buisness. This paradigm has largely been adopted in different context and applied to a large set of technologies. According to NIST (National Institute of standards and Technology) [94], which provided the definition and most accepted description of the general characteristics of cloud computing, *"Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configureable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction"*.

The popularity of the cloud computing is mainly due to cost savings, scalability and on-demand resource provisioning enabled by virtualization. Today, it underpins a significant portion of the web and is the actual means of deploying services and applications at scale. Despite the major evolution and all the benefits not all organisations are ready to adopt cloud computing for their IT infrastructures and services. Because the concepts and technologies of cloud computing are complex and potentially disruptive [77]. Beside economic considerations, those of security and resilience, risk management,

trust and control are important issues that prevent organisations from fully adopting cloud computing [99].

2.1.1 Cloud Characteristics

The essential characteristics which define cloud computing system can be summarised as:

- **On-demand self-service** – the ability to allow consumers to use cloud services as needed without any human interaction with the cloud provider.
- **Broad network access** – means computing capabilities are available over the network and accessed through standard mechanisms on heterogeneous platforms (e.g. smart phones, laptops, and computers) as well as other traditional or cloud based software services.
- **Resource pooling** – clouds providing theoretically infinite computing resources (physical and virtual) on demand to the end users using a multi-tenancy model. This precludes the need to plan for provisioning and there is a degree of location independence in that the customer generally has no control or knowledge over the exact location of the provided resources.
- **Rapid elasticity** – the ability that resources can be rapidly and elastically provisioned, in most cases automatically, and rapidly released to scale down and scale up.
- **Measured service** – cloud provider control and monitor the different aspects of the cloud services by leveraging a metering capability according to the type of service. This is critical for billing, access control, resource optimization and capacity planning.

2.1.2 Cloud Deployment Classifications

Depending on how a cloud infrastructure is operated and managed, cloud computing has four deployment models which can be classified based on its architectural layout.

Public cloud – where a cloud offers one or more of these kinds of services in a pay-as-you-go manner to the public such as, public cloud providers are Amazon Web Services, Rightscale, Google, and Microsoft Azure. They are the most popular type due to their rapid setup time and low capital expenditure. The providers of this type of a cloud usually partition their physical servers and lease these portions to the cloud consumers. However, public clouds suffer from a lack of infrastructure transparency, which make them less attractive for large organizations [152].

Private cloud – is used to refer to services hosted in internal data centers of single organization, and not accessible to general users. They are not open to the public users in a sense that their infrastructure is controlled by private organizations. However, It may be managed by the organization or by a third party, and may exist on-premises or off- premises. Large organizations who wish to take advantage of scalability, availability and structural transparency with a strict enforcement of data security can use such a model.

Hybrid cloud – is a composition of two or more distinct cloud models (e.g. private, public) in order to combine the features of those models. This mixed environment is suitable for organizations that have software or hardware compatibility issues with the external cloud providers, but still want to take advantage of the vast storage space and other cloud resources provided by public clouds. Another reason to choose hybrid clouds is the flexibility in exposing IT assets for a limited time to the public users.

Community cloud – The cloud is shared by several organizations to support a specific community that has shared concerns. It may be managed by the organizations or by a third party and may exist on- premises or off-premises.

2.1.3 Cloud Service Models

Cloud computing distinguishes three levels of abstractions for providing services over the Internet [44]:

Software as a Service (SaaS) – delivers software that is remotely accessible by consumers through the Internet with a usage-based pricing model. SaaS has the ability to provide the end user with an interface, such as a web browser or a mobile app, to run applications on the cloud provider's premises. These applications are built and located on the clouds assets. Therefore, SaaS users can access the application from anywhere without concerns about the application's underlying installation and management issues. Google Docs[1] and Github[2] are key example of such a service model.

Platform as a Service (PaaS) – offers a high-level integrated environment to build, test, and deploy custom applications. This type of service can be beneficial to the developers for creating the needed software stacks or hardware structure to deploy their application to the cloud. Google App Engine[3] and AWS Elastic Beanstalk[4] are example of such model.

Infrastructure as a Service (IaaS) – is the provisioning of computer resources on-demand such as virtual machines (VMs), storage, networking and other resources to deliver software application environments with

[1] https://www.google.co.uk/docs/about/
[2] https://github.com/
[3] https://cloud.google.com/appengine/docs
[4] http://docs.aws.amazon.com/elasticbeanstalk/latest/dg/Welcome.html

a usage-based pricing model. Infrastructure can scale up and down dynamically based on application resource needs. Typical examples are Amazon EC2[5], Azure[6], OpenStack[7]. The major advantages of these types of cloud-based services are the low initial cost of application deployment, the flexibility in scaling the service based on users demand.

Figure. 2.1 illustrates these models.

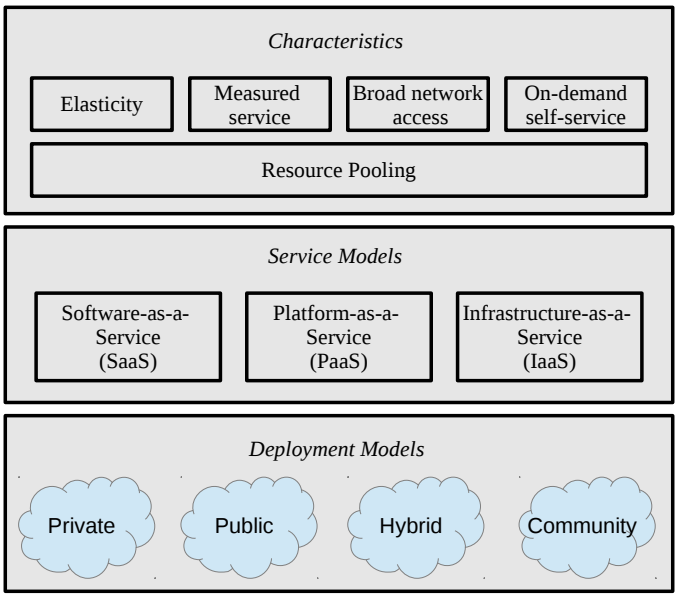


FIGURE 2.1: The cloud service models

2.1.4 Cloud Features

The cloud computing propose many features to orgaizations for cloudification of existing IT assets those are intended for agile development, self-managing workload, and economies of scale. The following are some of the features that are provided to the consumer based on their applicaiton domain.

Elasticity and scalability – cloud users have the ability to scale up and scale down the resources based on their application demands. They can also divide and distribute processing loads into multiple tasks operated by on-demand cloud resources.

High-availability – the cloud offers accessibility even with failure of some of its assets. Individual services may fail for one or more particular tenants, but the system continues to offer services to other tenants.

Utility-based service – cloud providers offer the pay-as-you-go model as their main billing system. Consumers pay for what they consume and

[5] https://aws.amazon.com/ec2/
[6] https://azure.microsoft.com/en-gb/
[7] https://www.openstack.org/

cloud providers can alert their consumers if they exceeded certain amount of resource usage or reach a limit of their assigned quota.

2.1.5 Cloud Computing Enablers

Virtualization is a key enabler of cloud computing which provides a high degree of flexibility in optimizing resource utilization. Virtual machine monitors (VMM) or hypervisors such as XEN, VMware, KVM can execute several virtual machine (VM) instances in parallel on a single physical machine, each VM is given a partition of the underlying resource capacity (CPU power, RAM size, etc.). Moreover, the live migration capability of hypervisors allows to migrate a VM from one physical host to another.

Virtual Machine Monitor (VMM) – is a software layer that takes complete control of the machine hardware and creates VMs, each of which behaves like a complete physical machine with its own operating system (OS). The VMM regains control anytime the VM tries to perform an operation that may affect the correct operation of other VMs or of the hardware. An OS that runs in a VM is a guest OS. Generally, there are two approaches to organizse VMs.

Type I, or hypervisor runs directly on the hardware without the need of a hosting OS. Examples include the VMware ESX and OpenStack. Figure. 2.2

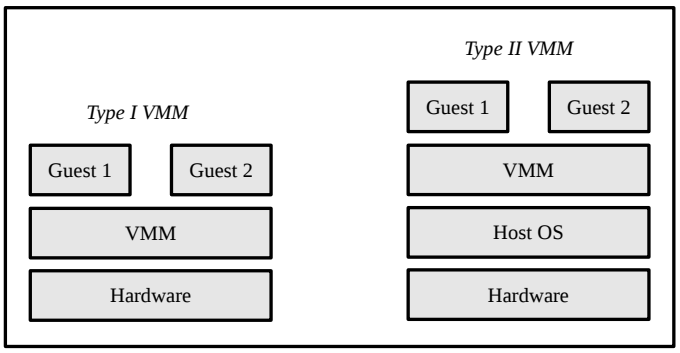


FIGURE 2.2: The VM organization appraoches

Type II – runs on top of a hosting operating system and then spawns higher level virtual machines. These VMMs monitor their VMs and redirect requests for resource to appropriate APIs in the hosting environment. Examples include the Hosted Xen and Virtualbox.

Virtual Machine Interoperability is a key challenge when it comes to migration of VMs from one cloud provider to another. There is format incompatibility for VM image and storage formats which is further compounded by lack of compatibilities in authentication, billing and resource allocation methods. Howerver, recently significant effort have been made with

respect to IaaS environments to standardised formats. The Open Virtualization Format (OVF) now provides a standard format for representing VMs and is the most likely candidate for allowing VM interoperability between IaaS providers. Furhter, the Open Grid Forum[8] and The Cloud Computing Interoperability Forum (CCIF)[9] is also working on this front.

2.1.6 Cloud Technologies

This setion provides brief description of cloud technology that is used in this thesis.

OpenStack OpenStack is an open source cloud computing platform which includes several APIs to provide cloud computing components on a cloud infrastructure provider level. The relevant stacks are `Nova` for compute resources, `Cinder` as block storage, `Swift` as a data storage, `Neutron` provides networking, `Keystone` offer identity servies, `Glance` for image services, `Heat` for orchestration and `Horizon` offers dashboard services to empower users to provision resources through a web interface. A big advantage of OpenStack is the open source code basis of the APIs therefore, developers could easily expand OpenStack architecture with functions they need. Similarly the API is adaptable so that mechanisms (such as *ADaaS*) that is developed in context of this work could be implemented.

2.2 Cloud Architectural Framework

The SECCRIT[10] consortium has developed an architectural framework for deploying infrastructure services in the cloud, which provides a basis for the development of research presented in this thesis (ADaaS - see Chapter 6). An architectural framework is needed in order to identify and locate the various mechanisms and potential new interfaces that are necessary to support infrastructure IT services.

The architectural framework was derived by SECCRIT consortium and has four abstraction levels. The key aspects of these levels are discussed using several viewpoints which include: multi-tenant, network access, monitoring, policy, legal and resilience. The author's contribution to the architectural framework is the resilience part which is highlighted in Section 2.3.1 and illustrated in Figure. 2.6. The resilience viewpoint is realized using tools and ideas developed as part of this thesis. These include the real time anomaly detection technique, the resilience management framework and anomaly detection as a service.

[8] https://www.ogf.org/ogf/doku.php
[9] https://www.cloudindustryforum.org/
[10] www.seccrit.eu

The proposed architectural as shown in Figure. 2.3 aims at a more precise role distinction that allows for better security analysis, separation of responsibilities and identification of separate administrative interfaces. It is an architectural framework, because it mainly serves descriptive and explanatory purposes, rather than specification purposes like a reference architecture definition in a standards organisation.

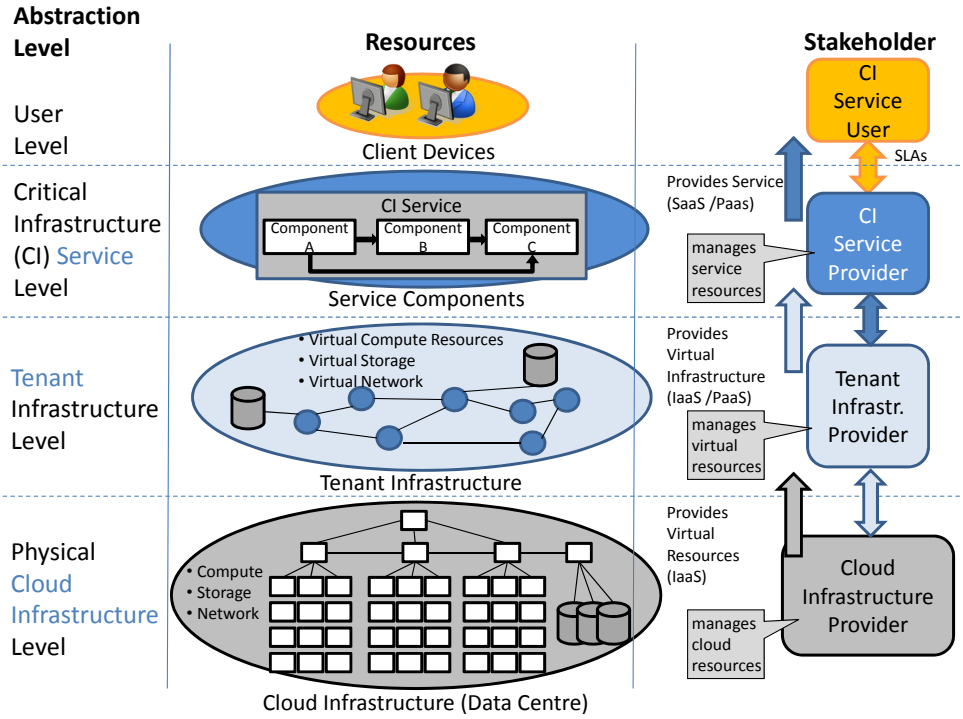


FIGURE 2.3: SECCRIT architectural framework [122]

Therefore, one should be able to map existing specific cloud architectures, deployments, and usage scenarios partially or fully to the SECCRIT architectural framework. An important aspect within the cloud services context is to consider that a single cloud infrastructure is typically simultaneously used by several tenants (i.e., customers, users of the cloud infrastructure provider), whose virtual resources are usually isolated from each other to some degree within the Cloud Infrastructure. The distinction between the virtual tenant infrastructure and the physical cloud infrastructure is important, since a clear separation between responsibilities is necessary. In case a service fails due to some anomaly, a root cause analysis should reveal the responsible party. Moreover, additional monitoring or logging mechanisms for anomaly detection can be located within the cloud infrastructure, as well as within the virtual infrastructure of the tenant. In this context, it may be helpful to identify additional interfaces that allow for increasing the trust level between the tenant and the cloud infrastructure provider by permitting some level of detection as a service. Furthermore, the architecture must clearly

distinguish the service provider from the virtual tenant infrastructure. This allows to separate two important aspects of any service hosted in cloud: functional and behavioural features. On the service layer, the functional features of the service are dealt with. Which components are required to compose the service and how these components need to be inter-connected. On the tenant infrastructure level the behavioural features of the service are dealt with: this includes elasticity features, component redundancy, and overload control. Failures on the physical infrastructure level can be made opaque to the service by self-healing mechanisms on the tenant infrastructure level, e.g., automatic fail-over to redundant components and auto-recovery by on-demand provisioning of virtual resources. In addition, geo-diversity can be realized on the tenant infrastructure level by requesting resources from multiple independent cloud infrastructure providers for increased dependability but also for higher security and resileince requirements [122].

SECCRIT architecture divides activities in a cloud environment into four different levels of abstraction. The different abstraction levels correspond to different stakeholders and their view on the managed resources.

User level – service user who remotely accesses the infrastructure service. For instance, an urban traffic management operator could observe and control the traffic flow within a city, using web-based interfaces as well as various distributed sensors that deliver measurement data as input to the next lower service level components.

Infrastructure service level – this level is controlled by the service provider who manages the resources at the service level. The service is composed of several components that interact with each other in order to provide the actual service. The service provider monitors the service operation and performance at this level. The service components usually either provide the application or the platform that are required at the user level. The service components are instantiated on the virtual infrastructure that is provided by the next lower level.

Tenant infrastructure level – provides a virtual infrastructure that consists of virtual compute resources, virtual storage, and virtual network resources. The virtual infrastructure is managed by the tenant infrastructure provider. This distinction of tenant from service provider is needed since they may be separate organisations. Several of such tenants are typically hosted within one cloud infrastructure that is the next lower level. The tenant infrastructure provider may provide either the pure virtual infrastructure (IaaS) or some basic services as a platform (PaaS) to the service provider. In the former case, the CI Service Provider may install complete VM images containing an operating system, middleware services, and application-oriented service components, including necessary configuration data. In the latter case, the tenant infrastructure provider may provide and operate some pre-installed

operating system images and middleware or supporting services.

Physical cloud infrastructure level – provides real physical compute, storage, and network resources, which are hosted in a data centre and administered by the cloud infrastructure provider. This level usually provides virtual resources (IaaS) to its upper level, i.e., the tenant. The virtualization solution usually provides (a certain degree of) isolation between the different tenants that are mapped onto the same physical infrastructure and thus permits the sharing of resources. For increasing the resource efficiency, the cloud infrastructure provider can usually transparently move virtual resources across its physical infrastructure.

2.3 Security and Resilience in Cloud

While moving from traditional computing paradigm to cloud computing paradigm new security and resilience challenges have emerged. The cloud architecture discussed in earlier section combines various levels of interdependent infrastrucutre, platform and applications; each level may suffer from certain vulernabilities which are introduced for example through misconfigurations either by user or service providers.

Further, the delivery models which are used within a particular deployment models of cloud can also exhibit certain characteristics (such as elasticity, scalability and multi tenancy and dynamic resource managemnt) which can expose cloud environments to threats related to the integrity, confidentiality, reliability and availability. These threats include but not limited to virtualization vulnerabilities, web application issues, privacy and control issues arising from third parties having physical control of data, issues related to identity management and issues related to data breach etc. Compared to non-cloud-based systems, understanding of the nature of challenges experienced by cloud providers is also opaque, as they are more prevalent than in the past, as highlighted in literature [99, 54]

In [19], the authors attempt to develop an understanding of the challenges faced by customers of an infrastructure-as-a-service (IaaS) cloud, along with their experience in resolving these problems. Their work is based on actual user problems and experiences as captured from the open support forum of a large IaaS cloud provider. They found that with the exception of problems related to application-level issues, the observed problems are closely related to the introduction of virtualization such as connectivity issues, virtual image management, performance, poor isolation between users, hardware degradation, etc. These findings are complemented by other literature which documented the virtualization-specific attacks where attackers can gain control over installed VMs. For example, DKSM [15] and "bluepill" [119] are some well known attacks on virtual layer of the cloud infrastructure. There are

also virtualization-specific zero-day vulnerabilities listed on NIST[11] that resulted an attacker gaining access to the hypervisor of a cloud node. One such vulnerability was exploited in the HyperVM application which resulted in destruction of many websites hosted on a HyperVM-based cloud [53]. One aspect of increasing security and resilience, and decreasing risk, is the detection of unusual activities (*anomalies*) in the use of resources by third parties. In order to combat those challenges, cloud providers usually install a first line of defence such as a firewall, which can help to prevent outside attacks, but not insider attacks [98]. In the same paper, the authors stressed that efficient intrusion-detection systems (IDSes) should be incorporated to mitigate challenges because various traditional non-cloud based attacks such as IP spoofing, routing-protocol attack, denial-of-service (DoS) and distributed DoS (DDoS), etc., are still valid for cloud environments. For example, a DoS attack on the underlying Amazon Cloud infrastructure caused bitbucket.org, a site hosted on Amazon Web Service (AWS) to remain unavailable for few hours [67]. Therefore, resilience and security are key concerns to be looked upon as highlighted by an International Data Corporation (IDC) survey [51].

A recent cloud-computing security white paper by Lockheed Martin cybersecurity division [91] shows that a major security concern after data security is anomaly detection which would allow understanding of normal patterns of cloud behaviour and, as a result, highlight abrupt deviations. However, anomaly detection potentially faces a challenge posed by cloud own characteristics (see Chapter 3) such as elasticity. In particular the dynamic invocation of new services and the migration of existing services, can lead to unpredictable normal behaviour that could render anomaly detection techniques unusable because of unacceptably high false-positive and -negative rates.

2.3.1 Resilience as a Cloud Need

Resilience is a wide-ranging concern, and can be defined as "the ability of a system to provide an acceptable level of service in light of various challenges" [132], [35]. Resilience is already supposed to be a fundamental property of the Cloud service provisioning platforms,e.g.,to be able to spawn new VMs in response to high load. However, a number of significant outages have occurred that demonstrate Cloud services are not as resilient as one would hope, particularly for providing critical infrastructure services [99, 101]. In a number of cases, these outages are caused by problems in the network infrastructure.

As a recent *ENISA*[12] report on the security and resilience of Governmental Clouds suggests: "the availability of a cloud service is often dependent on the network used to access it..." and measures should be taken to ensure

[11] http://nvd.nist.gov/
[12] http://www.enisa.europa.eu/

the resilience of access networks [30]. To the best of knowledge at time of writing this thesis, there is limited or no understanding about how to provide a resilient Cloud infrastructure that can collectively address challenges in a coordinated manner – these are treated as separate concerns.

Based on work by [132], the ResumeNet[13] project devised a framework whereby a number of resilience principles are defined, including the resilience strategy D^2R^2+DR: Defend, Detect, Remediate, Recover, Diagnose and Refine, which is outlined in Figure. 2.4.

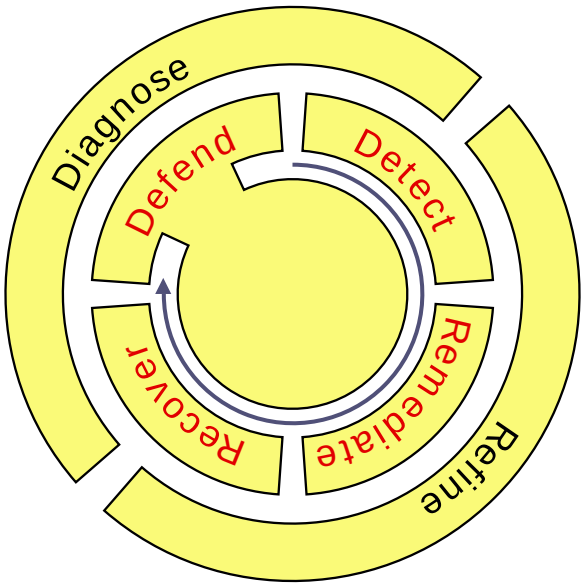


FIGURE 2.4: The D^2R^2+DR resilience strategy [132]

At its core is a control loop comprising a number of conceptual processes that realise the real time aspect of the D^2R^2+DR strategy and consequently implement resilience, and the thesis exploit this in implementing cloud resilience. Based on the resilience control loop, other necessary elements of the framework are derived, namely a deployment function, anomaly detection and policy engine that aim to build situational awareness, and multilevel information sharing and control mechanisms. Under the D^2R^2+DR framework, there must exist components capable of reconfiguring devices in response to challenges using policies (Figure. 2.5). Reconfiguration need not apply to the same components on which the detection was based. A policy engine is responsible for mapping detection events to reconfigurations, accepting a resilience strategy expressed as a collection of policies.

The SECCRIT architecture discussed in Section 2.2 provides a basis for deploying infrastructure services in the cloud. The D^2R^2 can be applied to the SECCRIT architectural framework to provide a resilience view (Figure. 2.6). At the physical layer, the cloud-infrastructure operator has access

[13] http://www.comp.lancs.ac.uk/resilience/

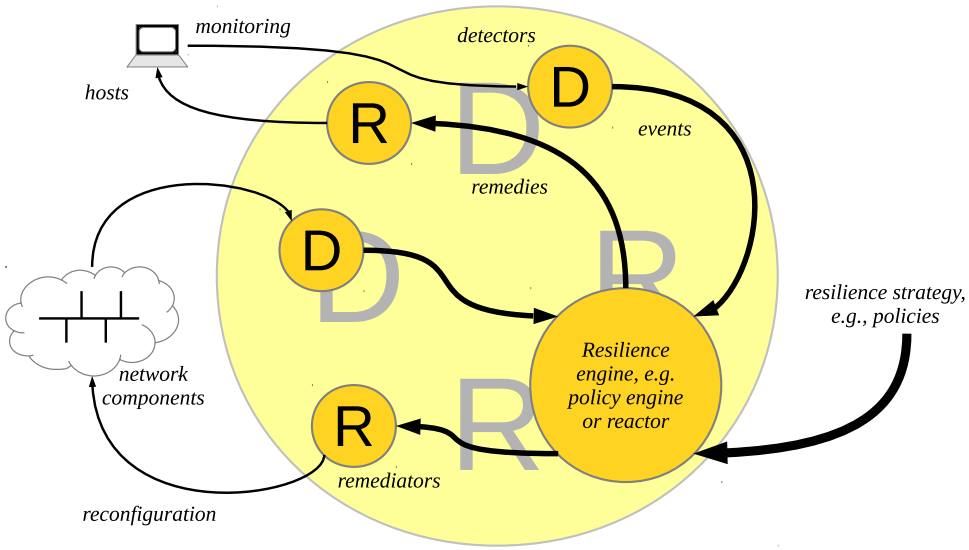


FIGURE 2.5: D^2R^2 Components

to physical nodes and the network, which can be monitored to inform the detection process. The operator can also reconfigure these devices, in response to detected challenges using policies. In a cloud infrastructure, D^2R^2 may exist as monitoring and reconfiguration points on physical hosts and networks, and on some virtual components. Resilience managers and detectors need not exist on any physical equipment used directly to provide virtual resources to the above layer. At the tenant-infrastructure layer, the tenant has access to VMs, and possibly virtual taps on VNs, which can inform detection. In response to challenges, the tenant may reconfigure the hosted machines, and some functionality of the virtual networks might also be exposed. Thus, tenant-infrastructure D^2R^2 is spread across components visible to this layer. Within the inner D^2R^2 loop, some interaction between these layers may exist in the form of events and reconfigurability exposed by the lower layer. For details on policy and resilience viewpoints, the thesis refer reader to the SECCRIT architecture white paper [7]. In order to ensure resilience the orchestration function includes a deployment function which translates high-level service descriptions and Service Level Agreements (SLAs) into automatically deployable descriptions such as *Heat*[14] templates in the *OpenStack*[15] environment. These deployment descriptions include the instantiation of anomaly detectors and their data collectors to observe network and system activity, and generate events to be handled by a the policy engine to remediate challenges. The activity can be performed at the physical (cloud-infrastructure provider) layer, which has physical networks and machines, and has an external view of system activity in VMs. Moreover,

[14] http://docs.openstack.org/developer/heat/template_guide/hot_guide.html
[15] http://www.openstack.org/

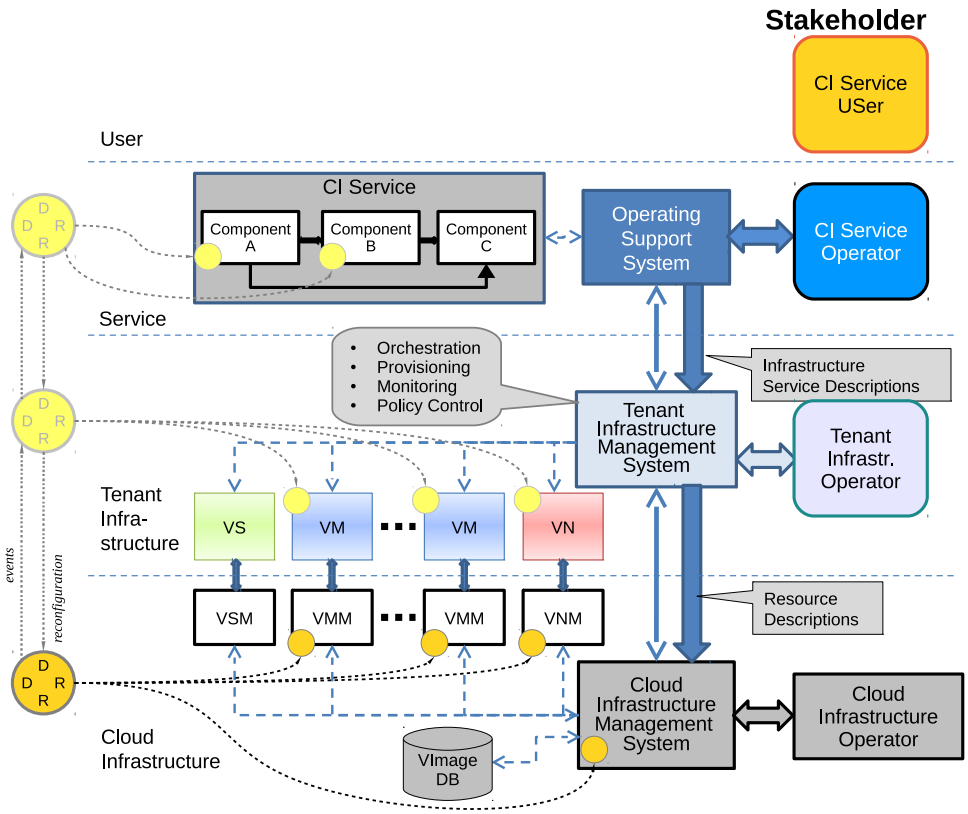


FIGURE 2.6: A resilience-oriented view [126]

network activity can be measured in the tenant-infrastructure layer by observing traffic on virtual networks, which could be performed by the tenant by running anomaly detectors on VMs that have access to these networks (Figure. 2.7). A mapping of components to architecture discussed in Chapter 4 (see Figure. 4.1).

2.3.2 Policy-based Resilience Management

Management and resilience of the cloud environments are closely linked. The extra layer of resource virtualization makes it difficult to plan effective management due to varying user demands, co-hosted VMs and the arbitrary deployment of multiple applications. Generally, management policies are used to govern the behaviour of a system. These management policies can be mostly looked upon as: "the constraints and preferences on the state or the state transition, of a system and is a guide on the way to achieving the overall objective which itself is also represented by a desire system state [52]". When using policy-based management, it is critical that rules being specified actually stem from the higher level requirements and that they are implementable.

Challenges to the operation of a Cloud Infrastructure can occur rapidly

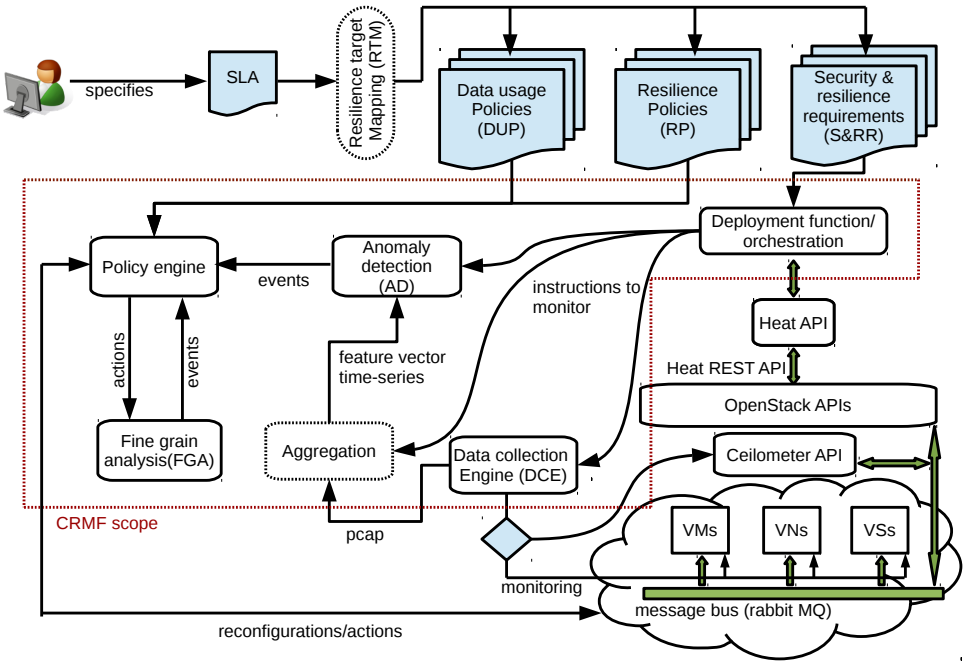


FIGURE 2.7: CRMF system architecture [126]

and with little warning, requiring a fast response in order to maintain acceptable service levels. In order to mitigate a challenge, complex multi-phase strategies are required, which combine various monitoring and detection mechanisms that influence the behaviour of remediation mechanisms. To address these issues, SECCRIT advocates the use of policy-based network management techniques for the configuration of resilience strategies.

These techniques allow descriptions of real-time adaptation strategies, which are separate from the implementation of the mechanisms that realise the strategy. This separation allows changes to be made to strategies without the need to take resilience mechanisms off-line. In short, two forms of policy are supported: authorisation (or access control) and obligation policies. Mainly policies are categorised as obligation and authorisation polices which are also supported by IND^{2}UCE[16], which is the policy environment used by the resilience framework.

Obligation policies specify management operation that must be performed when a particular event occurs given some supplementary conditions being true. These policies follow the Event-Condition-Action (ECA) paradigm and are of the form:

```
on <event>
        if  <conditions>
                do <target> <action>;
```

[16] http://www.iese.fraunhofer.de/en/competencies/security/usage_control/philosophy_uc.html

Therefore, the occurrence of the specific event is a necessary condition for the mandated operation to be performed. The event is a term of the form $e(a_1, \ldots a_n)$, where e is the name of the event and $a_1, \ldots, a_n$ are the names of its attributes. The condition is a boolean expression that may check local properties of the nodes and the attributes of the event. The target is the name of a role (i.e., a placeholder) where the action will be executed and so the service or resource assigned to the target role must support an implementation of the action. The action is a term of the form $a(a_1, \ldots, a_m)$, where a is the name of the action and $a_1, \ldots, a_m$ are the names of its attributes. To simplify notation an obligation policy can have a list of target-action pairs, all evaluated when the event is true and the condition holds. The attributes of an event may be used for evaluating the condition (to decide whether to invoke the action or not), or they may be passed as arguments to the action itself. Implicitly the role to which the obligation policy belongs is the subject of the obligation i.e. the entity enforcing the policy, and the action is invoked on a target role. Note the target may be the same as the subject i.e. a role may perform actions on itself. Obligations can also be used to load other policies (obligations or authorisations) into the system or existing policies may be enabled/disabled to change the management strategy at run-time.

Authorisation policies specify what actions a subject is allowed (positive authorisation) or forbidden (negative authorisation) to invoke on a target. The subject and the target are role names. The action and the condition are defined like in obligations. Authorisation decisions could be made by one or more specific roles in the network, but commonly implementations are based on the target making decisions and enforcing the policy as it is assumed that target roles wish to protect the resources they provide to the network.

```
auth[+/-] <subject>  if <condition>
                              then <target><action>;
```

2.3.3 Related Work on Resilience

The need for real-time and more dynamic nature of cloud infrastructure has made the task of defining a resilience framework very challenging. The policy based management has proven to be very effective for complex system management as evident in previous literature. The policy based approach to network and system management proposed in [95, 59] defines a framework for management for polices, policy hierarchies and policy transformation. In [117], the authors propose to enrich managed objects with policy goals as required by the management policy. It describes policies in two parts: an active part, containing application specific functionality, and a passive

part which can be re-used without any change. In [69], the authors used an approach to enforce policies by means of rules, but the understanding of a rule is more restrictive.

The works mentioned above are in the context of specification and implementation of policies and recently SLA, while little focus has been given to the refinement of the high-level requirements into low level polices. In [140], the authors presented an approach to policy translation that is based on a set of tables. The tables identify the relationships between users, applications, servers, routers and classes of service supported by network. Whilst this technique offers the advantage of being fully automated, it is an inflexible approach, only supporting a very specific type of high-level SLA policy and low level device configuration policy.

The work in [29] outlines a policy authoring environment that provides a policy tool, called POWER, for refining policy. A domain expert first develops a set of policy templates, expressed as Prolog Programs. The policy authoring tools have an integrated inference engine that interprets these programs to guide the user in selecting the appropriated elements from the management information model to be included in the final policy. The main limitation of this approach is the absence of any analysis capabilities to evaluate the consistency of the refined policies. Similarly, work presented in [18] allows for the translation of service-level objectives into configuration parameters of a managed system. The transformation engine takes the service requirements of the user as input, and search the database to determine the optimal parameters values that provide level of service limitation of this technique include its dependence on a rich enough database which is only possible by observing the system for some period of time; and the inability to deal with situations where a given requirement specification results in different configurations. There are several relevant projects which are highlighted below.

1. *ResumeNet* [113] defines a multi-level systematic framework to network resilience. ResumeNet provides blueprints and design guidelines for the cloud resilience management framework. The proposed resilience strategy in the *ResumeNet* is validated by detailing the guidelines which can be applied to the problem of channel interference in wireless mesh network and to explore the implications of multi-staged and collaborative detection.

2. *TClouds* [135] was an EU FP7 project aimed at developing a cloud infrastructure that achieves security, privacy and resilience. It objectives include to identify and address legal and business issues, define a security architecture for the cloud, and provide resilient middle-ware for

adaptive security on the cloud-of-clouds. The TClouds project targets cloud computing security and minimization of the widespread concerns about the security of personal data by putting its focus on privacy protection in cross-border infrastructures and on ensuring resilience against failures and attacks. They published work about an advanced cloud infrastructure that can deliver computing and storage which achieves a new level of security, privacy, and resilience.

3. *PRECYSE* [109] is an EU FP7 project. The strategic goal of PRECYSE is to define, develop and validate a methodology, an architecture and a set of technologies and tools to improve – by design – the security, reliability, and resilience of the information and communication technology (ICT) systems that support critical infrastructures (CIs).

4. *Cloud Controls Matrix (CCM)* [136] is specifically designed to provide fundamental security principles to guide cloud vendors and to assist prospective cloud customers in assessing the overall security risk of a cloud provider. The CSA (Cloud Security Alliance) CCM (Cloud Control Matrix) provides a controls framework that gives detailed understanding of security concepts and principles that are aligned to the Cloud Security Alliance guidance in 13 domains. The foundations of the Cloud Security Alliance Control Matrix rest on its customized relationship to other industry accepted security standards, regulations, and controls framework such as the **ISO 27001/27002**, **ISACA CoBIT**, **PCI** and **NIST** and will evolve to provide internal control directions for **SAS 70** attestations provided by cloud providers. This control framework can possibly serve as as the backbone for evaluation of the security levels of the CRMF.

5. *OrBAC* [70] was developed inside the RNRT MP6 project (communication and information system models and security policies of healthcare and social matters). The purpose of this project is to define a conceptual and industrial framework to meet the needs of information security and sensitive healthcare communications. OrBAC provides a well-defined access control policy model, which can be integrated into the CRMF framework, and shall enable fine-grained access control of the resources. The OrBAC API has been created to help software developers introduce security mechanisms into their software. This API implements the OrBAC model, which is used to specify security policies and also implements the AdOrBAC model [40], which is used to manage the administration of the security policies. The MotOrBAC [39] tool has been developed using this API to edit and manage OrBAC security policies. OrBAC has only been realized on homogeneous systems (such as firewall) or at software level.

2.4 Anomaly Detection

Anomaly detection systems are one more branch of intrusion detection systems that deal with identifying events that appear to be anomalous with respect to normal system behaviour. They get a model of the normal system behaviour and issue alerts whenever the behaviour changes, making a suitable assumption that such changes are frequently caused by malicious or disrupting events. Anomaly detection has been studied within diverse research areas and application domains. However, for cloud environments, anomaly detection techniques are still evolving due to the fact that it presents several challenging problems.

In IaaS the customer is responsible for the correct operation of their own software, the cloud provider is only responsible for the underlying infrastructure resources. This exacerbates the importance of anomaly detection and remediation mechanisms. However, the detection system is only effective if the alert it generates are timely, accurate and provide actionable information to the administrators to respond to potential threats.

2.4.1 Types of Anomalies

In [31], authors describe different type of anomalies:

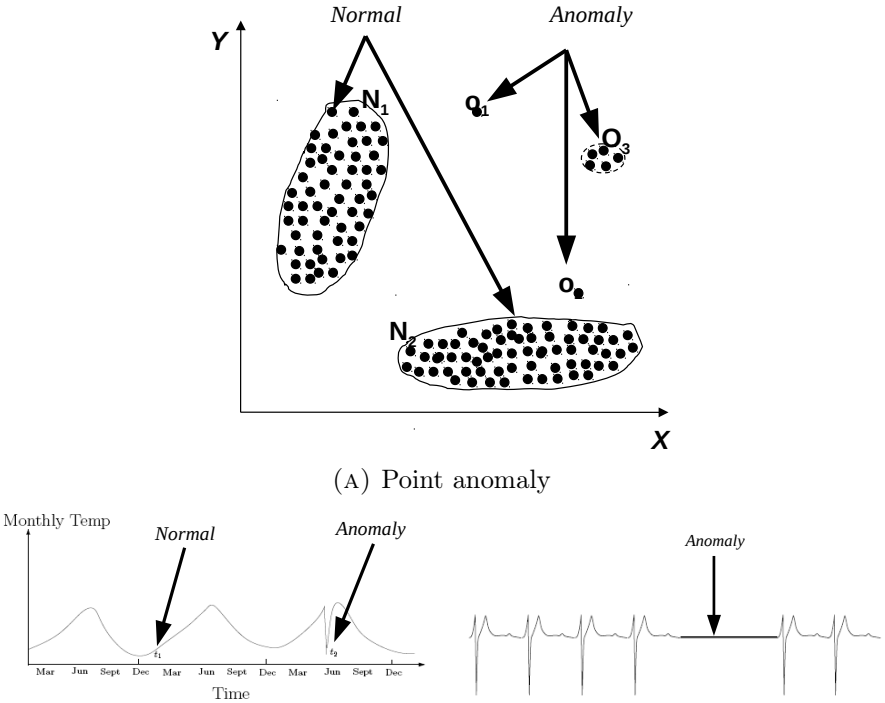
Point anomalies If an individual data instance can be considered as anomalous with respect to rest of the data (see Figure. 2.8a; O_1, O_2 and O_3 are outside the normal dense regions of N_1 and N_2).

Contextual anomalies If an information occurrence is anomalous in a precise context, but not or else, then it is characterizing a related anomaly (see Figure. 2.8b; a drop in `June` is not indicative of normal value found during this time).

Collective anomalies If collection of data instance are anomalous with respect to the entire data set, it is termed a collective anomaly (see Figure. 2.8c; horizontal line indicated anomalous subsequence when considered against the normal pattern of the rest of the data).

2.4.2 Anomaly Detection in Cloud Environments

In the cloud, the network traffic comes from multiple heterogeneous domains. Moreover, it changes rapidly with respect to its behaviour patterns due to heterogeneity of the tenants using the cloud and the elasticity of the exposed services. Under such circumstances of cloud computing, there are many challenges faced by underlying anomaly detection techniques such as mis-configurations, or simply by high volumes of legitimate traffic. The importance of anomaly detection in cloud is due to the fact that anomalies in



(A) Point anomaly

(B) Contextual anomaly

(C) Collective anomaly

FIGURE 2.8: Type of anomalies [31]

data translate to important actionable information. For example, in case of cloud, the anomalous traffic pattern could mean the request for VMs suddenly going high, because these requests are mostly accessible via Internet, therefore, causing some denial-of-service attack which could eventually hamper service availability to authorized users. Generally, commercial off-the-shelf systems (COTS) to detect intrusions are based on signatures also known as rules [123], [36]. In cloud environment signature based IDS can be used to detect known attacks. It can be used either at front end of cloud to detect external attacks or at back end of cloud to detect external/internal attacks. Like in non cloud based system it can not be used to detect unknown attacks in Cloud. In [16, 93], the authors have presented signature based techniques for detecting intrusions in cloud environments.

Contrary to signature based techniques there are many anomaly based detection techniques which have recently been applied at various levels of cloud computing, due to their native strength in finding unknown attacks. In [57, 49], the authors have highlighted that applying such techniques to cloud has been very challenging due to the fact that there exist large number of events at network and system levels in a cloud node.

Some of these approaches have been shown to be sensitive to parameters of the data sets, such as the level of traffic aggregation, and suffer from other robustness problems [115]. In general, anomaly detection techniques

TABLE 2.1: Summary of the reviewed work

Reference	**Scalability**	**Online**	**Memory-less**	**Multilevel**
Dapper [129]	✓	✗	✗	✓
EbAT/Wang et al. [141]	✓	✗	✗	✓
Guan et al. [58]	✗	✗	✗	✓
Garfinkel et al. [50]	✗	✗	✗	✓
Lee et al. [81]	✓	✗	✗	✗
Dastjerdi et al. [41]	✗	✗	✗	✓
Pannu et al. [105]	✗	✓	✗	✓
Cohen et al [38]	✗	✗	✗	✓
Pinpoint [34]	✗	✗	✗	✓
E2EProf [3]	✗	✗	✗	✓
SysProf [2]	✗	✗	✗	✓
PREPARE [134]	✓	✓	✗	✗
DAPA [73]	-	✗	✗	✗
Data Density approach/*ADaaS* [125]	✓	✓	✓	✓

are based on specific assumptions about the data, for example the statistical distributions of events. If these assumptions do not match reality, the outcomes can be unacceptable rates of false negatives and false positives [8].

The thesis would like the reader to refer to the detailed survey of anomaly detection for non-cloud based systems which has been produced by Chandola et al. [32].

2.4.3 State-of-the-art of Anomaly Detection in Cloud Environments

Previous research has created scalable methods for real time data collection [92, 148], to support online detection based on data mining and machine learning approaches [4, 14, 74, 143, 56]. However, while monitoring has been feasible at scale, detection is typically performed after volume of monitoring data has been stored in disk, which impedes the scalability of real time detection. Dapper [129] uses a static sampling strategy which is homogeneous across all nodes in the network which makes it inflexible for multi tier applications.

Wang et al. [141] proposed the EbAT system to allow the online analysis of multiple metrics obtained from system-level components (e.g., CPU utilisation, memory utilisation, read/write counts of the OS, etc.). The system showed potential in detection accuracy and monitoring scalability, but it was not evaluated in the context of adequately pragmatic cloud scenarios. Guan et al. [58] and Garfinkel et al. [50] proposed multi-level anomaly detection techniques to detect intrusions at different levels of a cloud system. The techniques appear to be rather inflexible and the application of those techniques in an operational context requires better clarification. Lee et al. [81] proposed a multi-level approach, which provided fast detection of anomalies discovered in the system logs of each guest OS. One of its disadvantages is the apparent lack of scalability, since it required increasingly more resources under high system workload. Also, it is only specific to detection in logging data.

Similarly, Dastjerdi et al. [41] proposed an approach based on mobile agents for an intrusion detection system for cloud systems. However, scalability appears to be an issue due to the high number of virtual machines that are required to be attached to the agent. The authors in [105] instrumented a real time adaptive anomaly detection framework that was able to detect anomalies through the analysis of run time metrics using the traditional two-class Support Vector Machine (SVM) algorithm. However, the main issue raised by this study was that the formulation of the two-class SVM algorithm suffered from the data imbalance problem, which affected the training phase, and consequently led to several misclassifications of newly tested anomalies.

There also exist threshold based techniques which are mostly being used in industry monitoring products. They use upper/lower bounds of each metric based on threshold values which come from predefined performance knowledge or from historical data analysis. However, they can not deal with the scalability needs of the future cloud applications because most of them underpin statistical algorithms with high computing overheads. In addition they often require historic data and/or knowledge in regards to normal or anomalous behaviour of a system. Specifically, Cohen et al [38] developed an approach that statically clusters metrics with respect to service level objectives to create system signatures. Pinpoint is proposed in [34], which uses clustering/correlation analysis for problem determination. E2EProf and SysProf are profiling tools proposed in [3, 2] respectively, that can capture monitoring information at different levels of granularity.

PREPARE [134] and DAPA [73] are two recently proposed frameworks for performance evaluation based on anomaly detection for virtualised environments. Although, their main focus is to identify SLA violations. These frameworks only address application-related issues which are manifested into

performance anomalies. However, none of these approaches focus on the impact of elasticity of the cloud such as VM live migration and high volume of data due to heterogeneity of the cloud. The thesis make an attempt to propose a model that assimilate density based technique which can work under these challenges. Table 2.1 provides a summary of the reviewed work for anomaly detection in cloud along with comparison of density based approach offered by *ADaaS*, discussed in Chapter 6.

2.5 Summary

This chapter introduces the main concepts underlying the thesis namely the definition of cloud computing systems from the unique perspectives of security and resilience. It start by providing overview of cloud computing: its features, its technologies, concepts. This serves to frame the context of this thesis and serve as a preamble for anomaly detection in cloud environments. The following section discusses the resilience as a cloud need. Furthermore, the resilience management is discussed from the perspective of technical challenges and open issues when it comes to operationally apply resilience mechanisms such as anomaly detector in cloud environments. Finally, the chapter considers the positioning of anomaly detection in an overall cloud architecture and summarizes currently available anomaly detection techniques.

Chapter 3

Impact of Live VM Migration on Anomaly Detection

Cloud environments have evolved as the critical backbone for many ICT infrastructures due to their elasticity and resource transparency. As reflected in a recent report by the European Network and Information Security Agency (ENISA), cloud environments are becoming increasingly mission-critical [85]. Since they provide always-on services for many everyday applications (e.g. IPTV), safety critical operations (e.g., Air Traffic Control networks), critical manufacturing services (e.g., utility networks and industrial control systems), and critical real-time services (e.g., transportation and surveillance systems) [20]. Therefore, the ability of such cloud environments to remain operational in the face of anomalous activities becomes paramount.

Modern virtualised cloud environments support migration of services and virtual machines (VMs) to different physical nodes, and exploit the consequent elasticity and resource transparency for dynamic resource management. In particular, in contrast to cold migration, live migration allows a service or VM to move while retaining its network identity and connections, and without having to be powered off, by transferring its active memory and execution state. This makes live migration essential functionality for effective on-line resource management, allowing the workload to be balanced across physical nodes without major disruption to users, and is performed by the majority of cloud operators (such as VMware VSphere [96]).

While migration is a key feature of cloud environments, it introduces novel security and resilience challenges. For instance, an anomaly detector applied to network traffic visible at the cloud-infrastructure level[1] could be misled by the effects of migration on that traffic in two ways. First, legitimate migration could be misidentified as an anomaly (a false positive indication). Second, migration could occur simultaneously with a genuine challenge, and thus mask its detection (a false negative). Overall, despite the plethora of signature-based and anomaly-based detection solutions for a number of

[1]The thesis refer the reader to a SECCRIT whitepaper [7] which presents an architectural model as a basis for the bringing services into cloud environments: https://www.seccrit.eu/whitepaper

computer networks (e.g., [16, 13]), there has not yet been a thorough analysis on the impact of VM migration on state-of-the-art AD solutions.

In this chapter, the objective is to understand how migration is represented or visible at hypervisor level, and then to what extent VM migration, which generally is representative for normal traffic, can be incorrectly classified as an anomaly. Additionally, certain anomalies may be missed because of VM migration. Considering this as a significant problem, the aim is to test whether state-of-the-art anomaly detection techniques remain reliable in cloud environments and, therefore, whether they meet security and resilience requirements for underlying infrastructures. Using a testbed along with custom scripts and various characterisation (see Section 3.2.13), the detection performance of PCA [79, 106], clustering-based, i.e., K-means [106, 147], Naïve Bayesian [150] and Expectation-Maximization Gaussian Mixture Model-EMGM [21] anomaly-detection techniques have been examined. As these, AD techniques have been shown to give acceptable detection performance results in non-cloud based systems [116, 46, 104, 111]. Under different attack intensity and VM migration scenarios, the ability of these detection techniques to detect anomalous behaviour in the cloud is measured. The obtained results suggest that, in some configurations, anomalies are missed and some configuration anomalies are wrongly classified. These outcomes, empower the thesis argument that under certain attack-type and migration conditions, the number of attacks that are missed and false alarms generated by these techniques could render them unreliable and unusable respectively.

Consequently, the widely used AD techniques are directly affected by the live-migration aspect and, therefore, future designs of cloud-oriented anomaly detection components should consider this factor. The rest of the chapter is organised as follows: Section 3.1 describes various types of migration in cloud environments. Section 3.2 reports on experiments carried out to determine the impact of migration on anomaly detection, including methodology and description of experimental setup. Finally, Section 3.3 highlights key findings of the results, and suggests experimental and analytical improvements. Section 3.4 offers summary of the chapter.

3.1 Migration in Cloud Environments

Migration is a capability of cloud-computing implementation models that allows an individual or organisation to easily shift between different cloud vendors without any implementation, compatibility, interoperability and integration issues. Different types of migration are possible, such as local- vs. wide-area migration, and service vs. VM migration, and these involve varying processes, and have differing security and resilience impacts.

3.1.1 Significance of VM Migration with respect to Anomaly Detection

Virtual machine migration refers to a process to move virtual machines between different physical hosts without causing disruptions to users and applications. The technique is widely used in today's data centre for efficient workload consolidation, elastic scaling, maintenance and fault tolerance [64]. Further, the migration process is designed in such a way that the move of virtual machines remain transparent to the users i.e., no disconnection or impact on the availability of the resources or any underlying security mechanisms. However, a recent work suggested [151] that VM live migration adds difficulties to anomaly detection since it is based on the large numbers of memory copy operations which can mask anomalies and attacks. In addition, the problem is compounded due to applications behaving differently on newly migrated host due to different workload, memory size and other network settings from original host. Considering this an interesting problem the thesis aims to quantify the impact of elasticity within an intra-cloud scenario where migration occurs between physical nodes in the same cloud environment. The focus of the experimental study is on VM migration as the main elastic scenario influencing anomaly detection due to the following reasoning:

- VM migration is chosen as it is a readily available function of the virtualization software installed on the testbed. Services running on top of a VM are therefore at a distinct layer, meaning that their configuration is no more complex than if they were deployed in a conventional non-cloud environment. To reproduce service migration, the additional management software is needed to be installed on the testbed, so that it can migrate services as required.
- VM migration is more clearly defined than service migration for the purpose of reproducing experiments.
- The traces obtained from the testbed will reflect genuine background traffic. TCP streams, for example, will compete with each other for bandwidth, vary their window sizes, and perform retransmissions.
- The effect of migration will also be reflected accurately in the traces. As a VM migrates away from a physical node, traffic to it will cease, and other traffic to the node will be able to consume it.
- The effect of anomalies will also be reflected accurately in the traces. For example, volume-based attacks will consume bandwidth from legitimate traffic.

The sub sections below highlight different aspects of migration.

3.1.2 VM Migration and Service Migration

The use of virtual-machine migration has attracted significant attention in the recent years [138]. VM migration is the transfer of a complete virtual machine from one physical host to another, and is of benefit in cloud environments by providing dynamic options for load balancing, maintenance and fault management. A virtual machine exists as a memory and execution state on its physical host, under control of the local hypervisor, and this state must be copied exactly to the new host's hypervisor to perform the migration. As the state is dynamic, it helps to suspend the VM just before copying, but this "cold" form of migration can lead to several seconds of downtime for services supported by the VM. Under "live" VM migration, this downtime is minimised by exploiting the lack of activity in most of the state most of the time.

Service migration is the transfer of the implementation of a service between machines (whether physical or virtual), and is possible because services are not bound to specific machines. However, as services are implemented by running processes, and these are attached to open sockets, file descriptors, shared memory and other resources, live migration of processes is non-trivial, and requires hardware abstraction [43] and *a priori* partitioning of processes [76].

3.1.3 Local-area and Wide-area VM Migration

The scope of a migration can vary. During a *local-area migration*, a VM running on a physical host is migrated to another physical host in the same subnet. Since the migrated virtual machine retains the same network address as before, any ongoing network level interactions are not disrupted. Similarly, bulk storage is often not part of the VM (e.g. it is network-attached storage), does not need to move with it, and remains reachable from the VM's new location.

A *wide-area migration* involves a VM moving to a location outside the local-area network, possibly to a different physical datacentre. This is more complicated than local-area migration, first because of the volume of state to be transferred (which might include bulk storage), potentially over networks with smaller capacity than available in the LAN. Second, if the VM is to be migrated live, it will have to retain the IP address from its old location, and redirection, masquerading or tunnelling will be required to ensure that current and future external connections continue to reach the VM in its new location. Alternatively, for reduced complexity, a VM could be shutdown for a cold migration (with the corresponding disadvantages of downtime and lost connections), and could acquire a local IP at the new location with DHCP,

but this would still require additional indirection mechanisms (e.g., DNS) to be updated.

3.1.4 Security Issues with Migration

During the procedure, protecting the contents of the VM state is an important consideration as the volatile state being transferred may contain highly sensitive information like passwords and encryption keys. A secure channel is at times not enough for protection. Mutual validation among the hosts involved in the migration might even be a more important issue to be considered [62, 102].

Live VM Migration, like any other network-bound process, is susceptible to network attacks such as ARP spoofing, DNS poisoning. If an attacker somehow manages to place himself between the source and the destination host, he can then launch active (man-in-the-middle) attacks. The fact that the live migration procedure is usually carried out inside a LAN makes it even more likely for a network attack to be successful, especially in situations where different third-parties run their VMs inside the same network subnet, which is the case in cloud environments.

3.2 Experiments

The main objective of this chapter is to study the effect of a cloud's elastic properties on anomaly detection techniques. The focus is on *migration* as the main elastic scenario influencing anomaly detection, and on *LAN VM* migration in particular, as its study can form the basis of later studies into WAN VM migration, and because it is more clearly defined than service migration for the purpose of reproducing experiments. Service and WAN migration also present greater complexity in their set-up within a simulated or testbed environment, whereas LAN VM migration is a standard feature of most cloud virtualization systems.

The experiments are designed to test various AD techniques against various scenarios that include both attack-related anomalies and VM migration, and compare their outputs against the corresponding ground truth for those scenarios. By discriminating those comparisons according to the presence or absence of migration, the detection performance of each technique along the same distinction can be quantified, and compare them to measure the "effect of migration" on any given technique.

3.2.1 Selection of Anomaly Detection Techniques

In order to investigate effect of migration on anomaly detection techniques, The choice had to be made on anomaly detection technique to be used. This

choice was based on what traffic it analyses and where in the cloud environment that traffic is collected. The data for anomaly detection can come from continuous monitoring at various levels in cloud such as (hypervisor and system level) and recording of measurements data. Running services in virtualised environments open a new issue to acquire data from running systems because of it's dependency on the infrastructure itself such as OS. Customised scripts and tools such as volatility can be added dynamically on hypervisor level to access, for example the file systems or dynamic memory and traffic feature extraction. This approach has two main benefits:

(a) Examiner can use new or updated versions of his tools without fearing to taint the service environment in an unpredictable way.

(b) Running these tools outside the service environments prevents malicious code from hiding itself by sophisticated obfuscation techniques.

Another challenge in this regard is that services expand and shrink based on user demand spanning multiple physical systems if required. System virtualization is the technology that enables this partitioning and pooling physical resources. The same technology is used by the cloud provider to relocate running services to other physical resources during e.g. maintenance periods. Thus, network analysis tools need to be extended to cope with this increased dynamicity of cloud services.

One other reason to combat a challenge on cloud computing requires investigating different levels such as traffic at network level, user actions to identify typical attack behaviour. A specific attack might not be visible at network level or system level because cloud specific attacks don't necessarily leave traces in a node's operating system where the system based monitoring resides so increased knowledge coverage is required for better detection of anomalies. Moreover, the choice for AD technique selection was also based on ease of implementation, proven ability to detect anomalies. In total four different techniques were implemented. Below section discuss how individual techniques are realised for investigating their performance in face of migration.

3.2.2 Principal Component Analysis

The spirit of *PCA* as described by [79], is to separate the normal data from anomalous by mapping a set of data points onto new axes. PCA employees specific technique called Singular Value Decomposition *SVD* to separate the normal data from anomalous. These axes are called the principal components. Applying PCA to the dataset X yields a set of m principal components $\{pc_i\}_{i=1}^{m}$. Assuming that data in X is zero-mean, the first principal component pc_1 is the vector that point in the direction of maximum variance in X and computed as follows:

$$pc_1 = arg \max_{\|pc\|=1} \| Xpc \| \tag{3.1}$$

The basic idea of applying PCA for anomaly detection is that: the $k - subspace$ obtained through PCA corresponds to the normal behaviour of the traffic, and is spanned by pc_1, through pc_k, whereas the remaining subspace i.e, pc_{k+1} through pc_m corresponds to anomalies. Afterwards, the magnitude of the projection of data point x_i can be computed into the anomalous subspace to quantify its abnormality. This unit of abnormality is used to produce an anomaly score graph, under the assumption of normality of the original data which will have an exponential distribution. Finally different thresholds can be set to classify the traffic measurement as in dataset X as normal or anomalous.

3.2.3 K-means

Another approach to detect anomalous behaviour is based on clustering [66], which functions by assigning data points that have similar features to cluster structures. There are a variety of clustering based approaches for anomaly detection, which are based on certain assumption. In this work the *K-means* clustering has been used with an assumption that normal data instances lie close to their closet cluster centroid, while anomalies are far away from their closest cluster centroid [32]. In the first step, the selection has to be made for the number of clusters k and then choose training data using K-means which have random samples of migration, anomaly and background traffic. These random samples give centroid. In next step calculate the distance of test data instances from centroids of cluster, using Euclidean method as per equation 3.2 and then for each test data instance its distance to its closest cluster centroid is calculated as its anomaly score [147].

$$d(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \tag{3.2}$$

where $x = (x_1, \ldots, x_n)$ and $y = (y_1, \ldots, y_n)$ are two input vectors with n quantitative features.

Obviously, a different number of clusters may result in better clusters, e.g. if the migration already shows distinct periods of very low and very high traffic volume under normal conditions. However, the determination of an optimum number of clusters based on a cluster evaluation criterion is beyond the scope of this work.

3.2.4 Naïve Bayesian

Naïve Bayesian is a well established Bayesian method primarily formulated for performing classification tasks. This is a simple yet very powerful technique for anomaly detection. It works best in distinguishing those anomalous instances in data which are similar to normal data instances due to strong assumption that the independent variables are statistically independent. Therefore, under this assumption it works by analysing the relationship between independent variables and then dependent variable to derive a conditional probability for each relationship.

In [100] author used this technique for intrusion detection where the Bayes theorem is described as:

$$P(H|D) = \frac{P(D|H).P(H)}{P(D)} \tag{3.3}$$

where D be the data and H be some hypothesis representing data instance D, which belongs to a specified class C. For detection, the first step is to determine P(H|D),which is the probability that the hypothesis H holds, given an observe data instance D. P(H|D) is the posterior probability P(H|D), is based on more information such as background knowledge than the prior probability P(H), which is independent of D.

Similarly, P(D|H) is posterior probability of D conditioned on H. Bayes theorem is useful because it provides ways to calculate the posterior probabilities P(H|D) from P(H), P(D), and P(D|H). Two classes(C1 = normal, C2 = anomalous) are considered, given D, predict C1, C2. The Bayes rule is shown in equation 3.4:

$$P(C_i|D) = \frac{P(d_i|C_i).P(C_i)}{P(D)} \tag{3.4}$$

where C_i represents the category of classes and $d_1, d_2, \ldots, d_n$ are instances of data D. These probabilities are used to compute the anomaly score.

3.2.5 Maximization (EM) for Gaussian Mixture Model (GMM) - EMGM

A Gaussian Mixture Model (GMM) [114] can be used as the basis for anomaly detection and is the most ubiquitous parametric model used for the anomaly detection [87]. This method learns patterns of normal and anomalous behaviour to classify that use a set-of Gaussian probabilities distribution functions. The use of maximum likelihood is then used as the deviation between normal and anomalous behaviour. These mixture models are a type of density models that comprise of number of a component, usually Gaussian. The GMM probability density function is a weight sum of N Gaussian distribution components and can be defined as:

$$p(x|\lambda) = \sum_{i=1}^{N} w_i g(x|\mu_i, \Sigma i),) \tag{3.5}$$

Where x is a D -dimensional random vector (i.e, features),w_i, $i = 1, \ldots, N$ are the mixture weights, and $g(x|\mu_i, \Sigma i),)$, $i = 1, \ldots, N$, are the component Gaussian densities. Each component density is a D-variate Gaussian function of the form,

$$g(x|\mu_i, \Sigma i),) = \frac{1}{(2\pi)^{D/2}|\Sigma i|^{1/2}} \exp\left\{ -\frac{1}{2}(x-\mu_i)'\Sigma_i^{-1}(x-\mu_i)\right\}, \tag{3.6}$$

with mean vector μ_i and covariance matrix Σ_i. The complete Gaussian mixture model is parametrised by the mean vectors, covariance matrices and mixture weight from all the component densities.

The distribution of feature vector was extracted from time series of each dataset and parameters of mixture model are used by the Expectation Maximization (EM) algorithm. The basic idea of the EM algorithm is beginning with an initial model *lambda*, to estimate a new model $\bar{\lambda}$ if the probability of $\langle x|\bar{\lambda}\rangle$ is greater than or equal to $\langle x|\lambda\rangle$. The new model then becomes the initial model for the next iteration and process is repeated until some convergence threshold is reached.

Finally, calculate the log-likelihood which is simply the log of the probability density function for the model to calculate anomaly score.

3.2.6 Selection of Features

All tested anomaly-detection techniques take a time series of feature vectors as input. For experiments, a series is generated from a packet trace by computing a feature vector for each 1-second bin of the trace. These vectors include both volume-based features ("number of…") and distribution-based features ("entropy of…"), and are chosen to capture the dynamics of varying anomaly types. The complete list of features follows:

- Number of packets
- Number of bytes
- Number of active flows in each bin
- Entropy of source IP address
- Entropy of destination IP address
- Entropy of source port
- Entropy of destination port
- Entropy of packet size

3.2.7 Evaluation Metrics

Each AD technique is evaluated with several metrics. First, all applications of an AD technique to a dataset yield the same type of structured data, namely, a time-series anomaly-score table. Each input entry submitted to the detector describes the features of traffic during a given time period (bin), and the detector yields a corresponding entry in the anomaly score table, a value in $[0, 1]$ representing the 'distance' of the bin's feature vector from normal traffic. Each output entry is augmented with ground truth (GT; a boolean value indicating the presence of an attack during the bin), and 'instantaneous migration intensity' (IMI; a value in $0, 1$ indicating whether migration occurred during the bin). This transformation is depicted in Figure. 3.1. (Further details are to be found in Section 3.2.10.) The first form

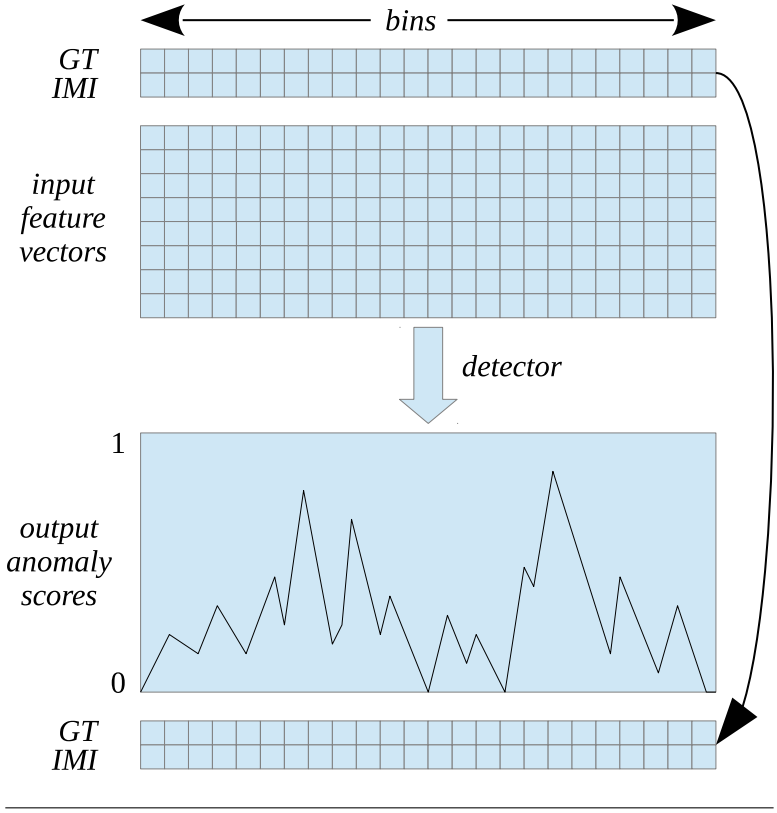


FIGURE 3.1: Conversion by detector of features to anomaly scores, with copied labels

of evaluation is the *Anomaly Score Graph* (ASG), which is a graphical representation of the anomaly-score table, allowing visualization of the detector's response to migrations and anomalies known to take place during the period covered by the graph.

When any given threshold is applied to the anomaly-score table, an extra boolean attribute is generated for each bin (true if the score is above the threshold; false otherwise). This can be compared with GT, yielding a bin which is counted as a true positive (TP; if GT is true, and the score is

above the threshold), a true negative (TN; if GT is false, and the score is below), a false positive (FP; if GT is negative, but the score is above), or a false negative (FN; if GT is positive, but the score is below). For a given table and threshold, this then allows computation of the *true-positive rate* (TPR, sensitivity or recall; $TP/(TP + FN)$), the *false-positive rate* (FPR; $FP/(FP + FN)$), the *precision* ($TP/(TP + FP)$), and *detection rate* ($(TP + FN)/(TP + FP + TN + FN)$).

These rates depend on a potentially very arbitrary choice of threshold. In contrast, Receiver Operating Characteristics (ROC) curve and a Precision Recall Curve (PRC) represent all choices of threshold graphically. To generate either from the anomaly-score table labelled with ground truth, the applied threshold is gradually increased from 0 to 1. For each threshold, a pair of rates is computed (TPR and FPR in the case of ROC; precision and recall (TPR) in the case of PRC), and these form the next pair of co-ordinates in a graph (both of whose axes are in the range $[0, 1]$).

For a ROC, as the threshold is increased, these co-ordinate pairs cannot decrease, and the curve gradually approaches $(1.0, 1.0)$, ultimately reaching it when the threshold is set to 1.0. As true positives are good, and false positives are bad, a ROC indicating good performance shows a curve leaning towards $(0.0, 1.0)$, while bad performance is expressed by a curve leaning towards $(1.0, 0.1)$. For example, if an AD technique such as PCA has a with-migration curve which is chiefly lower than its without-migration curve, this would show that migration has worsened the performance of the detector.

In order to compare the detection of the individual detectors by their sensitivities, the precision recall curve (PRC) is used. The recall or true positive rate (TPR) is the proportion of correctly predicted anomalies to the actual size of the anomaly vector. Precision is then the proportion of anomalies that were correctly detected by detector relative to the predicted size of the anomaly vector. These measures are defined as:

$$Recall = \frac{True\ positive}{True\ positive + False\ negative} \tag{3.7}$$

$$Precision = \frac{True\ positive}{True\ positive + False\ positive} \tag{3.8}$$

The PRC is important for evaluation since there could be possibility that some of the data instances are few in numbers relative to other. For example if the portscan attack represented 95% of total traffic in dataset and the normal data instances were 5%. If all the predicted instances were portscan then the overall accuracy will be 5% in spite of the lost predicted normal class of data.

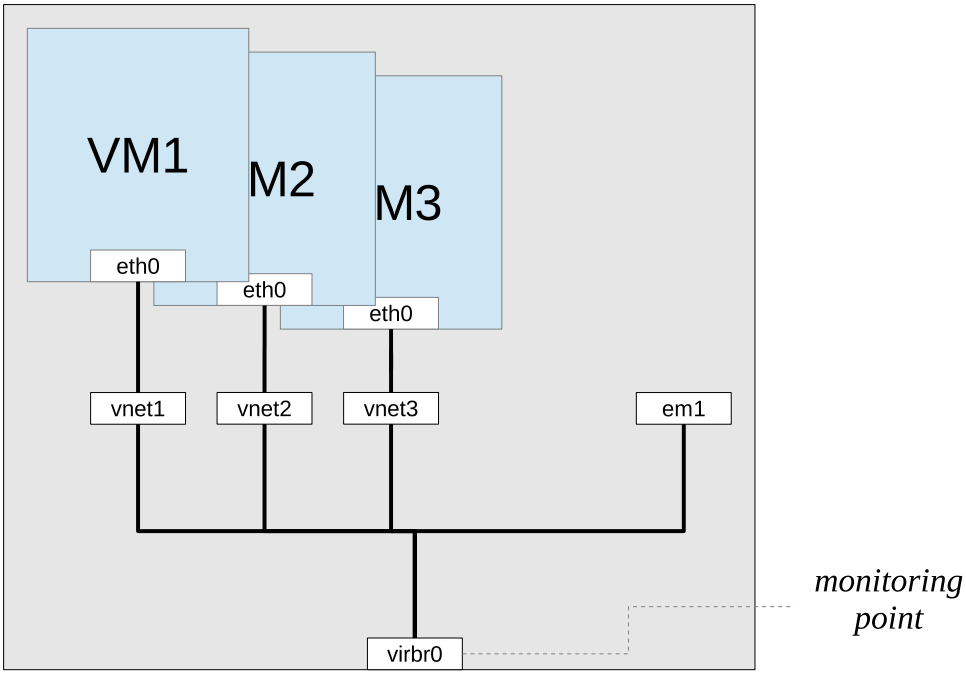


FIGURE 3.2: Virtual network architecture in a single node

3.2.8 Testbed

The testbed is established in which live Local Area Network (LAN) VM migration can take place while experiencing both normal and abnormal traffic conditions. Two hosts serve as nodes for running multiple VMs. Another host acts as a controller to initiate migrations, and also to generate background traffic. A fourth host generates attack traffic. All are connected to a LAN, Figure. 3.2 shows testbed architecture in a single node. Each physical node runs Kernel-based Virtual Machine (KVM)[2] as virtualization infrastructure, and the Quick EMUlator (QEMU[3]) provides hardware emulation. Migration is achieved with *libvirt*[4]. All VMs on a node are connected to a virtual bridge interface `virbr0`, so their own interfaces appear to be part of the LAN.

For the experiments presented here, each VM runs Apache HTTPd. The client host runs custom scripts to initiate random HTTP requests from the VMs. The 'challenger' host runs custom attack scripts to generate attack traffic directed towards the VMs' address range for a selected attack type and intensity (i.e., the volume of traffic it generates). Tcpdump[5] is used to simultaneously collect packet traces from the two virtual bridge interfaces, one in each physical node, and so these traces represent aggregate traffic to/from all VMs on a node.

[2] http://www.linux-kvm.org/
[3] http://www.qemu.org/
[4] http://libvirt.org/libvirt2
[5] http://www.tcpdump.org/

This set-up allows us to run experiments in which the legitimate traffic of several web servers is continuously emulated, while anomalous traffic is emulated by overlaying the legitimate traffic with attack traffic from the attack scripts that run during part of the experiment. Independently, one of the VMs running a webserver can be migrated live between the nodes during a period of either normal or anomalous traffic. Subsequently, traces obtained at the virtual bridges can be fed into anomaly detectors to observe their reactions to normal/anomalous traffic with and without migration. Because to have control over when migrations and anomalies occur, the obtained traces can confidently be labelled with ground truth about both conditions, and therefore assess the impact of migration on anomaly detection that takes such traces as input. Further, every trace described by the type of background traffic it captures, the anomaly/attack type, the attack intensity, migration overlap (whether migration happens during the normal or anomalous period), and migration direction (whether a VM is leaving or arriving at the node generating the trace). Section 3.2.13 explains characterization in more detail.

3.2.9 Framework

The AD evaluation framework compose of various pre and post processing modules, which comprises of several scripts and libraries written in perl, python, C and Matlab.

Attack scripts For network and portscan, various python scripts are written which exploit rate limiting features of nmap[6] to control attack intensity and build attack scenarios. These scripts take a configuration file which specifies the timing (such as start and end time of the attack) and networking (such as IP addresses of the VM). These scripts are further reconfigurable using different scan, delay and timing modes of the *Nmap* scanner.

Denial of Service attacks are realised using, LOIC[7]. LOIC is an open source network stress testing tool, written in C#. It performs denial of service attacks on a target HTTP server by flooding the server with TCP or UDP packets. Intensity of attack was controlled by number of attacking machines and threads involved in generating attack traffic.

Monitoring scirpts are based on *tcpdump* format which collected traffic at each VM network bridge interface.

[6] http://nmap.org
[7] http://sourceforge.net/projects/loic/

Background traffic that is mainly composed by a variety of random client requests to the VM's HTTP servers at random intervals, which were achieved with a custom console application written in perl.

Summary extraction script (*traceStat.c*) are written in C based on *libpcap5* to convert the input into various normalised statistical properties on a per packet basis, which also has an interface with MATLAB for detectors. The output is a time-series which is converted into *STDIN* input into various features for specified bin size using aggregations script written in perl.

Detector scripts are written in Matlab to perform the anomaly detection. These scripts basically translate the combined scenario trace (background traffic + migration + anomalous) into detection percentage and then visualised the results by plotting PRC and ROC cure. These detectors are reconfigurable as per the parameters (such as components/dimensions, thresholds, normalization scheme) for individual detector types. Visualization is provided by a script which is based on comparing anomaly score to threshold and plotting ROC and PRC curve.

3.2.10 Method

To evaluate an anomaly detection technique in the face of migration, firstly several experimental runs were performed, where each yields a pair of packet traces which are labelled with the ground truth regarding the presence of attack traffic or migration in the trace. In each 10 minute run, background traffic occurs continuously at a fixed rate, and hence appears throughout a trace. At the first 5 minutes in the first run, an attack script starts, hence its traffic appears in each trace from the midpoint. At either 2.5 minutes or 7.5 minutes, a migration of one of the VMs is initiated. A run can therefore be characterized by the attack type and intensity, and whether the migration occurs during the attack or during the normal period (i.e. 'migration overlap') (see Section 3.2.13). One run for each of the 8 possible combinations were performed of these characterizations, yielding 16 traces. Each packet trace is filtered to eliminate the related management traffic between the VM host nodes. It is subsequently divided into 1-second bins, and each bin is converted into a feature vector, also labelled with ground truth. Tables of labelled feature vectors from related traces, i.e., the four in which the same attack type and intensity was applied (with NM/AM and MDin/MDout varying), are combined to form a dataset representing 40min of experimentation. For illustration purposes the processing from experiments to combine data sets for NS and DoS is shown in Figure. 3.3.

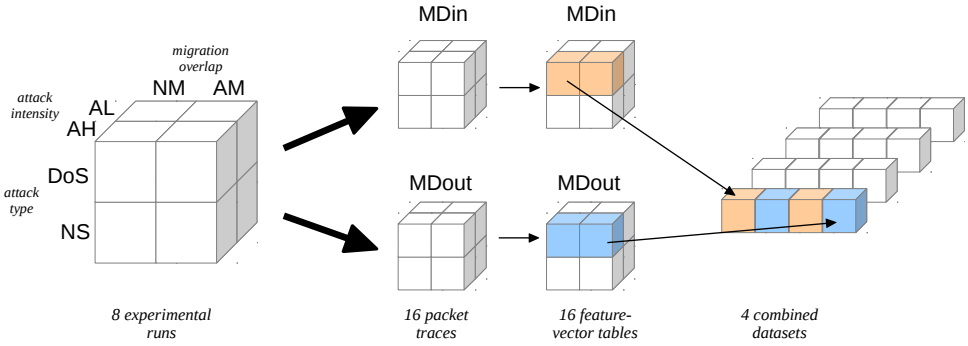


FIGURE 3.3: Processing of data

Each examined detection technique is then applied to each dataset, by submitting them together to an evaluation process, as depicted by Figure. 3.4.

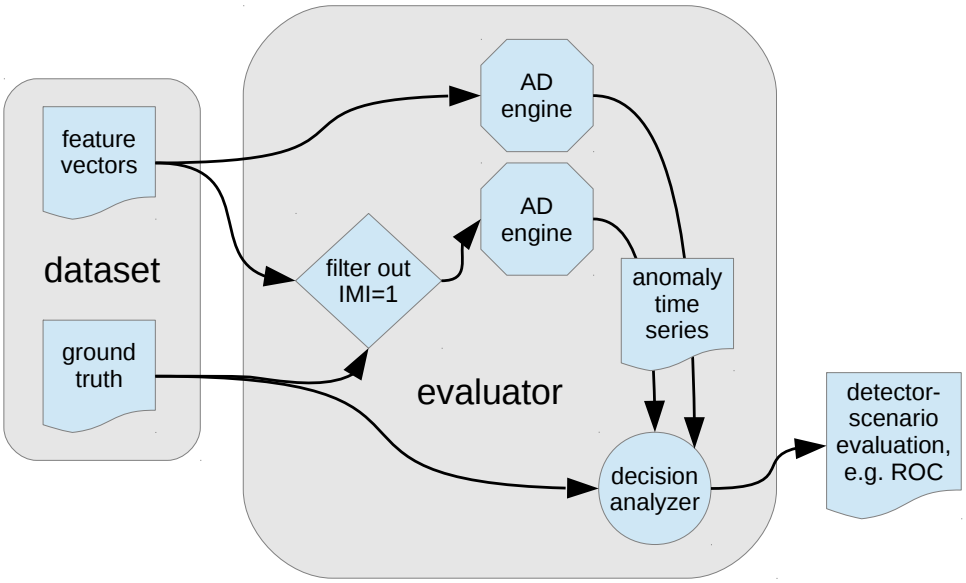


FIGURE 3.4: Evaluation process

The feature vectors from the dataset are submitted through a configured anomaly-detection engine, yielding an anomaly time series, a series of anomaly scores, one for each vector, indicating how anomalous the detector finds that vector with respect to the others. The anomaly scores are recombined with the corresponding ground truth, to yield a table of 3-tuples including anomaly score (AS), boolean anomaly ground truth (GT) and boolean migration ground truth (i.e. 'instantaneous migration intensity', IMI). The same dataset is also stripped of rows with IMI set, and submitted through the AD engine to yield a second anomaly-score table, which is similarly recombined with ground truth columns. Each labelled anomaly-score table then

generates a Receiver Operating Characteristics (ROC) curve[8], and the two can be compared to derive an evaluation of the detection technique against the scenario represented by the dataset. In order to aid the visualization process of the dataset, Some of the anomaly-score tables visually are presented as anomaly-score graphs (ASGs)[9].

Datasets could be obtained from public traces, simulation, or testbeds. The options for acquisition datasets are discussed in Section 3.2.11, and details of the actual datasets used appear in Section 3.2.12.

3.2.11 Potential Dataset Sources

Virtualized environments leverage the functionality of virtualization platforms, i.e., of the underlying Virtual Machine Monitors (VMM), and are thus able to monitor multiple VMs at the same time, which can provide different data for anomaly detection. From perspective of anomaly detection, data in cloud environments can be obtained at various layers. Several options exist for studying the behaviour of anomaly detection in cloud computing, but the essential core of any study is to run a packet/flow trace through an AD engine, and compare its output against ground truth.

VMM can be used to extract various system level data and there is flexibility to configure monitoring script at hypervisor level of the individual node for network data. Traces must contain varying types and intensities of anomalies, and varying intensity of migration. Each trace must reproduce several activities:

Background traffic A trace must include background or normal traffic, as the monitoring point in a real physical infrastructure will observe such traffic. This includes the traffic between the customers' VMs/services and their clients, but can optionally exclude internal management traffic, such as the transfer of a VM or the control protocols that establish new services, as these are likely to be handled by separate physical or virtual networks.

Anomalies A trace must include anomalous traffic, which is also observed at the monitoring point. It must be possible to know the type of challenge that is creating the anomaly, and its intensity, as well as when the anomaly is actually occurring, in order to determine ground

[8] A ROC curve is a plot of true-positive rate (TPR) against false-positive rate (FPR) for a range of thresholds. Points towards the bottom left correspond to high thresholds (or low sensitivity), and the top right to low thresholds (or high sensitivity). Better performance is indicated by curves that tend to occupy the top left, as these imply that sensitivity can be decreased to eliminate more FPs without degrading the TPR.

[9] An ASG displays a time series of outputs (anomaly scores) from a detector. The particular ASGs are annotated with the periods during which attacks (GT = 1 in pink) and migration (IMI = 1 in pale green) occur. An ideal detector would show high scores only while GT = 1, independently of IMI.

truth. How these are known depends on how the anomaly finds its way into the trace.

Migration A trace must include evidence of migration, including both arrival and departure of VMs or services. The presence of a migration does not affect ground truth about anomalies, i.e., the ground truth of an anomalous bin is the same regardless of whether a migration has just happened or is about to happen, or is even happening at the same time. However, it is necessary to know the intensity of the migration, e.g., the number of VMs or services migrating at the same time, in order to determine the effect of this particular elastic scenario (migration) on anomaly detection.

The main options for these aspects of reproduction are real traces, testbeds, trace injection and simulation.

Real traces Traffic data is significant part of any cloud system and the pattern of traffic will vary depending the assignment of virtual machines between physical servers. Due to the complexity of the patterns of such traffic, it is very hard to model it. Ideally, one would use a trace taken inside a datacentre at a location equivalent to the monitoring point, but no public datasets containing cloud behaviour are available. (Datacentres could be unwilling to provide useful traces that risk exposing sensitive technical and personal information.)

Non-cloud network traces are available CAIDA[10], which could be filtered, and then injected with migration to give the appearance of being taken inside a datacentre, but their use as a true representation of cloud traffic is questionable.

Real traces must be analysed to identify both anomalies and migration, either to label the traces to enable a detector to be evaluated, or so they can be removed to provide a clean trace, into which anomalies and migration must be injected artificially. In the former case, there might not exist a sufficient spectrum of anomaly types and intensities, nor a sufficient spectrum of migrations occurring at the right times with respect to the anomalies, to achieve the goals of the experiment.

Testbed traces Testbed (Section 6.3.1) can be configured to resemble a datacentre with two physical nodes running various VMs, each supporting various services. Background traffic can be created by running multiple clients outside the VM host machines. Although a testbed's traces can be regarded as unrealistically dry, the testbed provides a very controlled environment in which the ground truth is absolutely certain,

[10] http://www.caida.org/data/overview/

i.e., the type, timing and intensity of both anomalies and migration are under experimental control.

Trace injection Given a clean trace (e.g. from the testbed), one could modify it to make it appear to contain anomalies using an anomaly injection framework such as FLAME[11]. However, it is unclear how such tools model attack type, so there is a risk of unrealistic anomalous traffic.

Further, the thesis also studied the effect of LAN VM migration on packet traces of all VM traffic obtained by the hypervisors at the source and destination of the migration. The migrating VM remains on the same network segment, and so does not need to re-negotiate an IP address. Consequently, clients connecting to the VM remain connected, and the migration is functionally transparent to them. The source hypervisor observes the sudden absence of all traffic to and from the migrating VM. Similarly, the destination hypervisor observes the sudden appearance of the same traffic. Also, a time delay of about 0.8sec is observed between the end of traffic at the source and the start of traffic at the destination. From this, it may seem that the migration can be injected into an existing trace simply by removing all evidence of the migrated host before or after a certain point. This should be functionally accurate, but there may be non-functional affects that it would not represent. For example, client traffic to VMs that do not migrate might fill the bandwidth freed up by the departing VM. A feature such as bytes per second might be unaffected by this, while bytes per packet might increase as TCP window sizes increase.

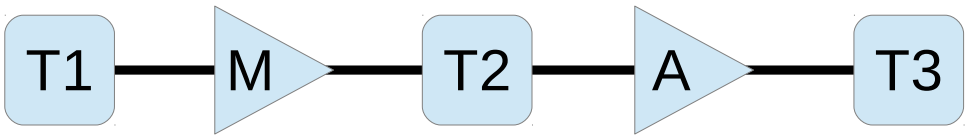


FIGURE 3.5: Simulating migration before anomalies

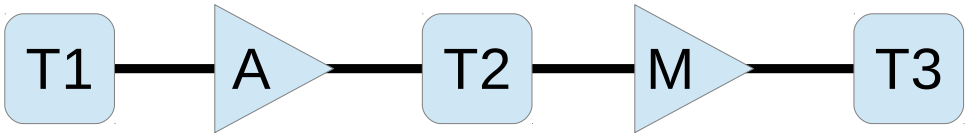


FIGURE 3.6: Simulating anomalies before migration

Injection of both migrations and anomalies into a trace is problematic. When a migration is injected first (Figure. 3.5), care must be taken to

[11] http://flame.ee.ethz.ch/

ensure that a subsequently injected anomaly does not contradict evidence of the migration. For example, after a VM has been migrated away from a monitoring point, no anomalous packet or flow injected into the trace from that point may appear to interact with that VM's address. This could be significant for some anomaly types that happen to be targeted at the migrating VM, while irrelevant for others, increasing the complexity of the simulation. It especially makes ground truth more difficult to establish, as that now depends on whether the anomalous traffic.

Injecting migration after the anomaly (Figure. 3.6) is also potentially problematic. If the anomaly is targeted at the migrating VM, evidence of the anomaly must move with the VM. Also, non-functional effects of migration might have an impact on the anomaly traffic, which can no longer be modified because it has already been injected.

In summary, it is non-trivial to model the non-functional effects of anomalies and migration by injecting them into an existing trace, especially when anomalies and migration occur together.

Simulation A simulation of all experimental aspects has the benefits of complete isolation and control over anomalies and migration. However, at this time, it is unknown whether existing simulators include models of VM/service migration, whereas one can immediately reproduce VM migration accurately on the testbed. Cloud environments have varying traffic patterns, which depend on the actual services offered by the supported VMs. By simulating traffic trace, it is hard to model such internal traffic pattern. Without exact model of internal traffic, local-area migration can not be observed or simulated because it only involves internal traffic.

3.2.12 Acquisition of Datasets

For these experiments, packet traces are opted to be obtained from the testbed. Background traffic is created by running several HTTP servers, and several clients repeatedly requesting dynamically created documents of varying size. Entropy in the background traffic's external IP addresses is created by restricting each client to an allocated port range, and then later mapping anything in this range to a new IP address. The monitoring scripts are configured based on *tcpdump* format which collected traffic at each VM network bridge interface. VM migration is applied during the acquisition of traces using *libvrt*, and anomalies (portscan and netscan) are orchestrated using custom scripts by exploiting rate limiting features of Nmap for intensity

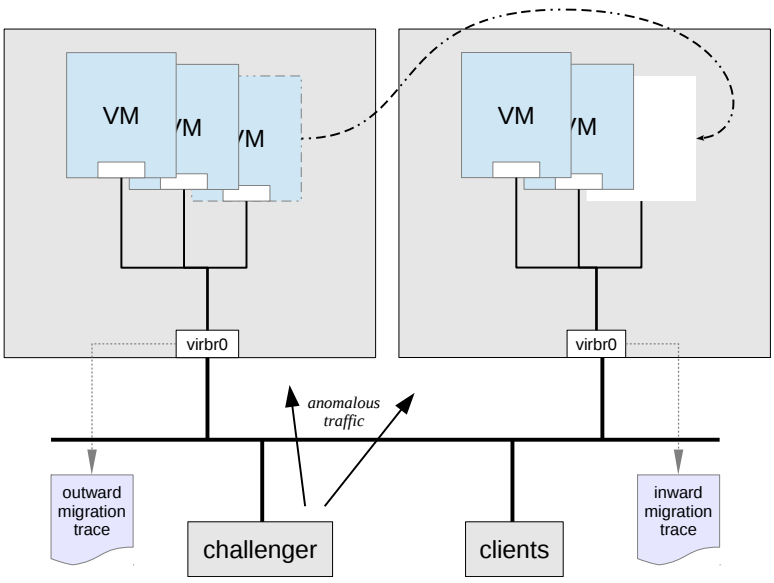


FIGURE 3.7: Experimental test-bed set-up

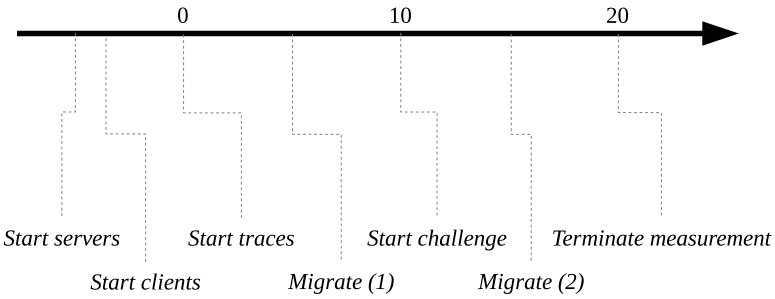


FIGURE 3.8: Measurement procedure timeline

of attacks,whereas, DoS was realised using LOIC, where intensity was controlled by number of attacking machines and threads involved in generating attack traffic.

Figure. 3.7 shows the set-up of the testbed components for performing a measurement run. Figure. 3.8 shows the procedure for taking a pair of measurements. Several VMs are established on each node, and an HTTP server runs on each one. Multiple HTTP clients run on a separate machine, and generate background traffic by interacting with the servers. Packet traces are started on both nodes, so two complementary traces are obtained per run. About ten minutes later, anomalous traffic is generated by a suitable tool, and directed towards the VMs' address range shared by both nodes. The precise time is recorded with respect to the clock on each node, which is used to time-stamp the trace. Migration either happens at about 5 minutes into the tracing, or about 5 minutes into the anomaly.

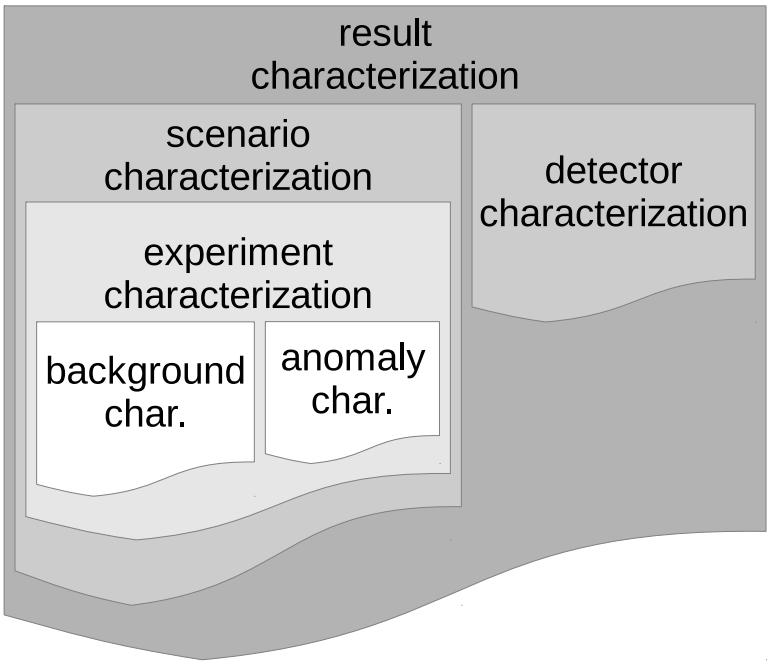


FIGURE 3.9: Composition of characterizations

3.2.13 Characterization of Experimental Runs, Scenarios, Detectors, and Results

Each dataset is characterized by the parameters of its creation, and this characterization serves to identify the dataset. Similarly, each AD engine (detector) that is to be evaluated has certain characteristics, such as its type and thresholds. When a detector is evaluated against a dataset, the obtained results are characterized, and effectively identified, by a combination of the dataset's and detector's characterizations. Correlating a result with its own characteristics will help us to learn about the relationship between migration and anomaly detection in cloud computing.

Several terms for characterizations are defined here of not only datasets, detectors and results, but also anomalies, background traffic, and experiments, as these contribute to the definitions of the other characterizations. This relationship is expressed in Figure. 3.9, which shows how each characterization is composed of others. In total, seven types of characterization are defined:

- Background characterization describes background traffic used in an experiment.
- Anomaly characterization describes how anomalous traffic is generated.
- Experiment characterization describes how an experiment is run.

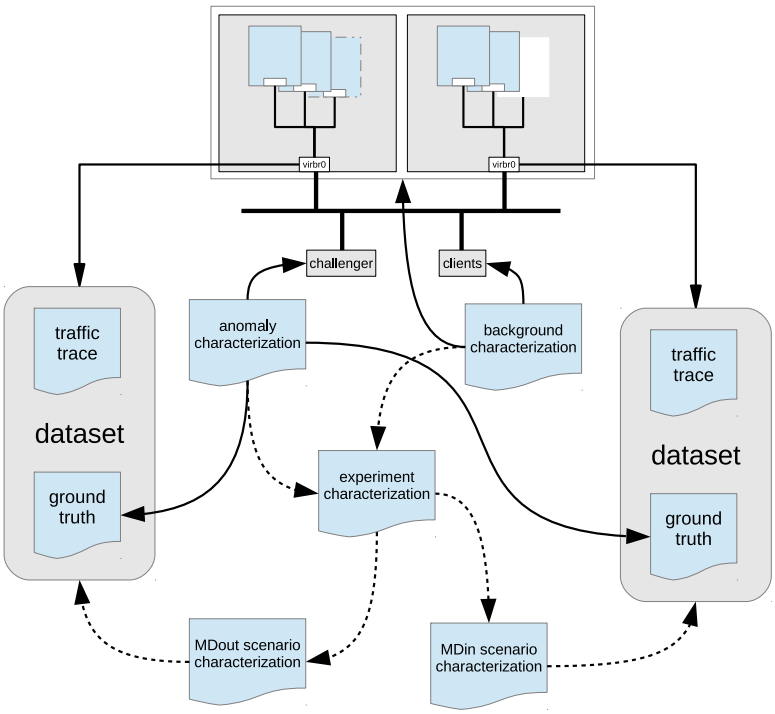


FIGURE 3.10: Influence of characterizations on dataset acquisition

- Scenario characterization describes a dataset obtained from an experiment. (More than one dataset may be obtained from a single experimental run.)
- Detector characterization describes the type and configuration of an AD engine.
- Result characterization identifies the evaluation of an AD engine against a dataset.

Dataset acquisition is influenced by most of the characterizations, as shown in Figure. 3.10. Each experiment generates two datasets, so the experiment's characterization is incorporated into the datasets' distinct characterizations.

AD evaluation is influenced directly by detector and scenario characterizations, as shown in Figure. 3.11. Scenario characterization selects the dataset, and detector characterization selects the AD type and its configuration. Together, they characterize the result.

3.2.14 Background Characterization

A *background characterization* defines how background traffic is generated during an experimental run, including the initial deployment of VMs and services in the testbed. Following are the background characterizations:

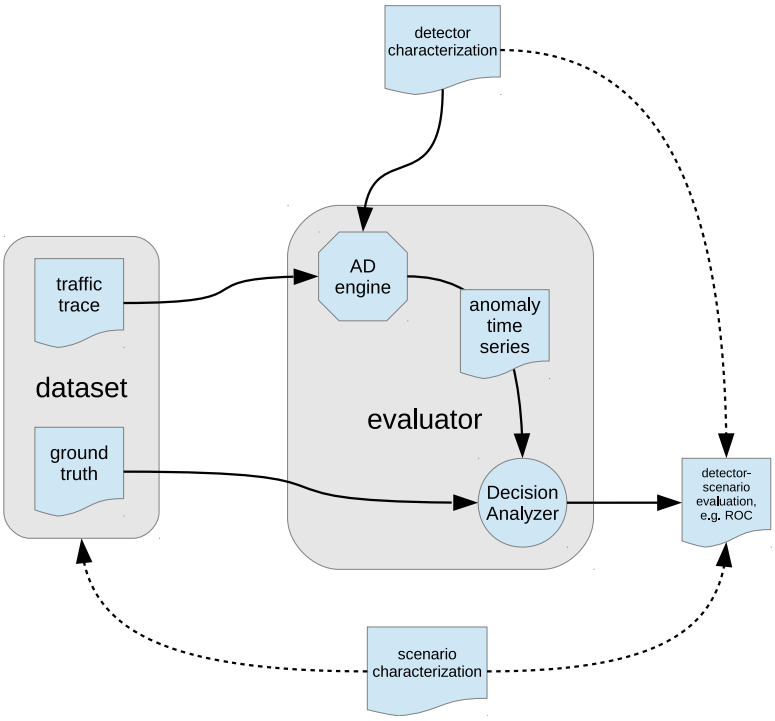


FIGURE 3.11: Influence of characterizations on evaluation

BC0 Five VMs run identical HTTP servers. Three run on one physical host, and two on the other. A host external to the VM infrastructure runs HTTP clients repeatedly connecting to each VM, two per VM. These clients and servers run for at least the duration of the experimental run.

3.2.15 Anomaly Characterization

An *anomaly characterization* characterizes the configuration of the software used to generate anomalous traffic during an experimental run, and consists of the following characteristics:

Anomaly type Possible types are no anomaly (NA), portscan (PS), network scan (NS) and Denial-of-Service (DoS). There are many other types of anomalies, but for this work it is limited to these three. These anomalies were picked as they are highly related with modern challenges face by critical services which are worms and attacks. Portscan is often used to probe a target VM and discover which ports are open. If attacker find a vulnerable service running on an open port, He can attempt to compromise the VM by exploiting some vulnerability. Network scans are typically used as propagation mechanisms by worms by using some exploit to gain access to new VM and then infect them.

For portscan and netscan, the nmap timing features are used to control the intensity of the attacks. Nmap's *–max-rate* and *–min-rate* options allow us to specify the rate. For high intensity, nmap dynamic timing options does a pretty good job to find appropriate rate to scan, e.g., when *–min–rate* option is given Nmap will do its best to send packets as fast as it can. For low intensity, the limit of *–max–rate* is set to 0.3 which means sending not more than 3 packets every 10 seconds.

For DoS, the attack intensity is controlled by setting thread limit in LOIC and number of machines running attack instance. For low intensity, attack was carried out using single machine with 54 threads, whereas for high attack intensity, 2 machines were used with 84 threads each.

Anomaly intensity Intensity varies with anomaly type, but for each type, it is defined as low (AL), and high (AH), as shown in Table 3.1.

Anomaly type	Low intensity	High intensity
Portscan (PS)	*nmap* with a rate of 0.3	*nmap* with a rate of *default/maximum*
Networkscan (NS)	*nmap* with a rate of 0.3	*nmap* with a rate of *default/maximum*
Denial-of-service (DoS)	single client with 54 threads	2 clients with 84 threads each

TABLE 3.1: Intensities of tested anomalies

3.2.16 Experiment Characterization

An *experiment characterization* characterizes an experimental run, and consists of the following characteristics:

Background characterization An experimental run involves background traffic.

Anomaly characterization An experimental run usually involves anomalous traffic being added the background traffic.

Migration intensity M0 is defined as absence of migration, M1 as migration of one host, and M2 as migration of two hosts.

Migration-anomaly overlap The migration happens either during the normal period (NM; Figure. 3.13a) or during the anomalous period (AM; Figure. 3.13b).

Migrant-targettedness MT0 indicates that the anomaly is targeted only at VMs that do not migrate during the experimental run. Figure. 3.14a

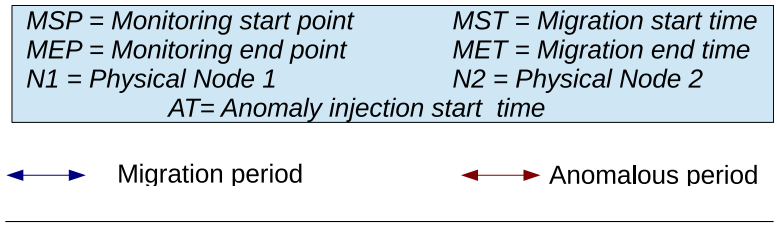


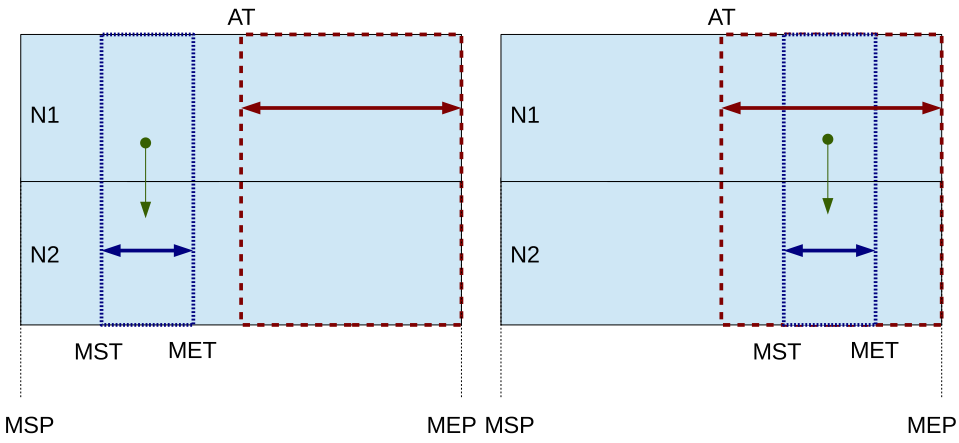
FIGURE 3.12: Legend for depictions of anomaly/migration interaction

and Figure. 3.14b show the interaction of MT0 with NM and AM respectively.

MT1 indicates that the anomaly targets at least one VM that migrates. Figure. 3.15a and Figure. 3.15b show the interaction of MT1 with NM and AM respectively.

This option is absent when the anomaly type targets all VMs.

(The legend for the aforementioned figures is given in Figure. 3.12.)



(A) Depiction of migration during normal period, with anomaly directed at network (NM)

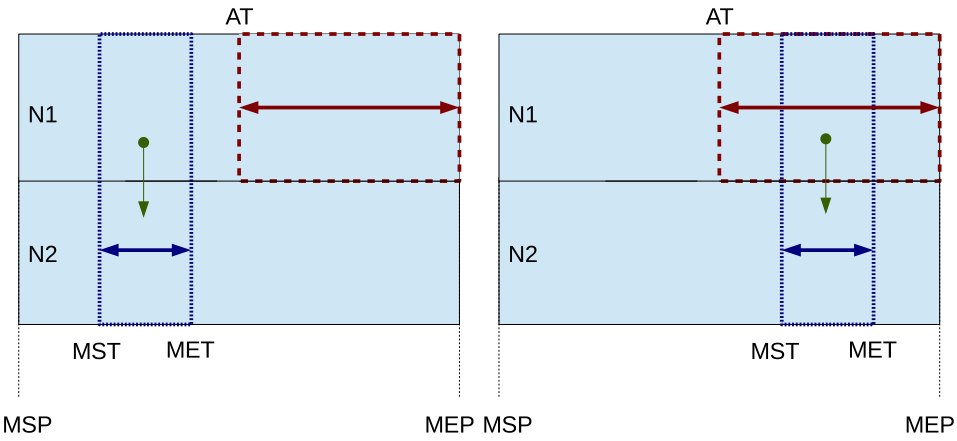
(B) Depiction of migration during anomalous period, with anomaly directed at network (AM)

FIGURE 3.13: Illustration of migration-anomaly overlap.

For example, BC0-PS-AL-M1-AM identifies an experimental run involving several HTTP servers and clients generating background traffic (BC0), a low-intensity (AL) portscan (PS), and a migration of one VM (M1) during the scan (AM).

3.2.17 Scenario Characterization

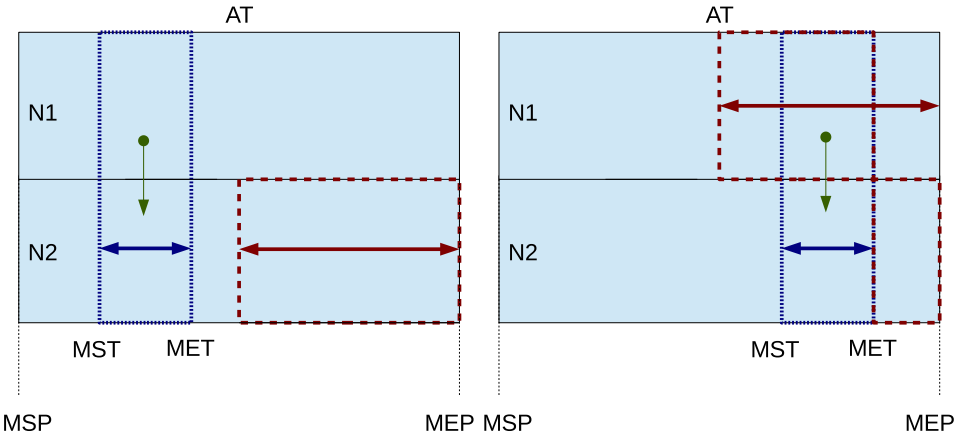
A *scenario characterization* characterizes a dataset, including one of the traces obtained from an experimental run, plus its ground truth. It consists of the following characteristics:



(A) Depiction of migration during normal period, with anomaly directed at static host (NM-MT0)

(B) Depiction of migration during anomalous period, with anomaly directed at static host (AM-MT0)

FIGURE 3.14: Illustration of migration-targettedness (MT0).



(A) Depiction of migration during normal period, with anomaly directed at migrating host (NM-MT1)

(B) Depiction of migration during anomalous period, with anomaly directed at migrating host (AM-MT1)

FIGURE 3.15: Illustration of migration-targettedness (MT1).

Experiment characterization Each dataset (trace plus ground truth) is identified by the experiment which generated it.

Migration direction The trace is obtained from either the physical host from which VMs migrated (outward migration, denoted by MDout), or the other host which received the VMs (inward migration, MDin).

For example, BC0-PS-AL-M1-AM-MDin identifies the trace taken on the interface of the physical host receiving the migrating VM (MDin), during the experimental run involving several HTTP servers and clients generating

background traffic (BC0), a low-intensity (AL) portscan (PS), and a migration of one VM (M1) during the scan (AM). When related datasets are combined, some elements may be missing, e.g., BC0-PS-AL-M1 combines AM/NM and MDin/MDout datasets.

Migrant-targettedness (part of experiment characterization) may affect ground truth. If the anomaly is targeted towards a VM that migrates during the anomaly, the ground truth on the source host (MDout) goes from true to false after the migration, while it goes from false to true on the destination host (MDin). In other words, ground truth could be different in MDout and MDin if MT1 applies.

3.2.18 Detector Characterization

A *detector characterization* specifies the selection and configuration of an AD technique. It includes the technique type, and (for example) any pre-selected thresholds used to configure it. The following detector characterizations have been defined (based on techniques described in Section 3.2.1):

PCA Principal Component Analysis (PCA) using Singular Value Decomposition (SVD)

KM K-means with $k = 2$, in supervised mode

NB Naïve-Bayesian in supervised mode, where random data was used for training.

EMGM Unsupervised mode

3.2.19 Result Characterization

A *result characterization* characterizes an output from the evaluator. It consists of the following characteristics:

Detector characterization This is one of the inputs to the evaluator, and so characterizes the results.

Scenario characterization This identifies the other input to the evaluator, so it also characterizes the results.

A result characterization is expressed as the concatenation of its components. For example, BC0-PS-AL-M1-AM-MDin-PCA identifies the results obtained by passing dataset BC0-PS-AL-M1-AM-MDin through PCA. When related datasets are combined, some elements may be missing, e.g., BC0-PS-AL-M1-PCA combines AM/NM and MDin/MDout datasets, before submission to PCA.

3.3 Results

For groups of related experiments, a combined dataset is prepared, starting with four traces corresponding to a particular anomaly characterization, e.g. high-intensity host port scan (PS-AH) over regular traffic (BC0). (For some anomaly types, there is also distinction between an anomaly targeted only at migrating VMs (MT1), and an anomaly targeted only at static VMs (MT0).) In two traces, migration of one VM (M1) occurs during the anomaly (AM), and during the normal period (NM). Orthogonally, two traces show inward migration (MDin) and two outward (MDout).

Each trace is converted into an 8-dimensional feature-vector table with 1-second bins, and each bin is labelled with ground truth (GT) and instantaneous migration intensity (IMI). The tables are then concatenated in the order AM-MDin, AM-MDout, NM-MDin, NM-MDout, to yield a single 8D table with additional GT/IMI labels. For each experiment, such a combined dataset is chosen, and submitted to a particular AD engine, e.g. PCA.

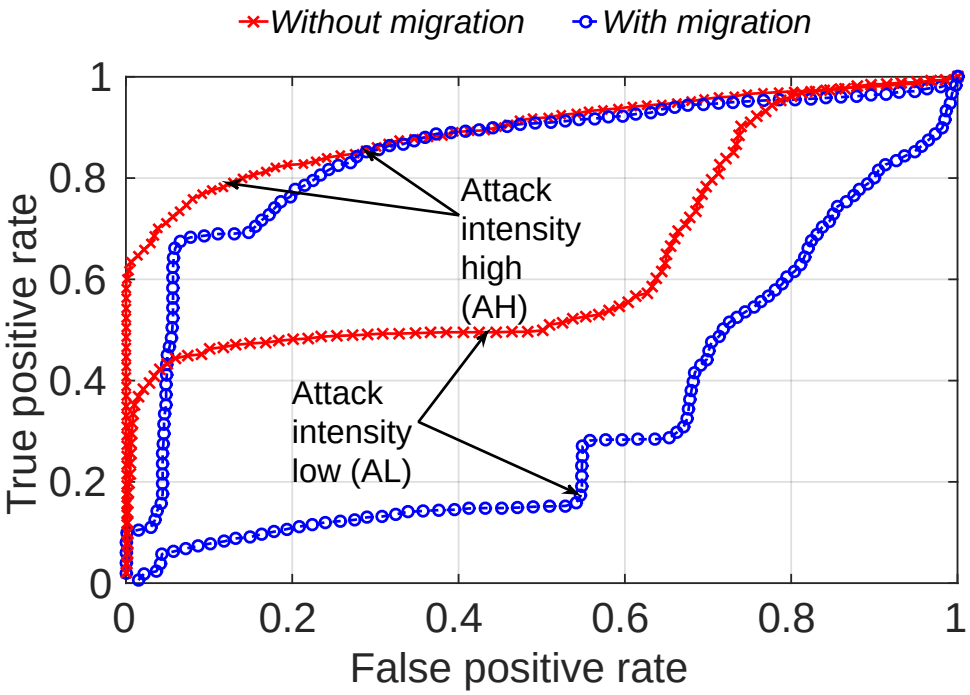
This section only presents and discusses the performance of two AD techniques PCA and K-means for attack scenarios DoS and NS under migration. Number of experiments were carried out on various state-of-the-art detection techniques by varying the type of and intensity of anomalies, and examining the live VM migration impact. These results are presented in Appendix A.

Submitting each of the four 40 minutes datasets through the evaluation process produces one ASG and two ROCs, one including the effects of migration, and one without. By comparing these ROCs, it was possible to assess the effect of migration on the AD technique that produced the table in the context of detecting the anomaly used to produce the traces that were submitted to the detector.

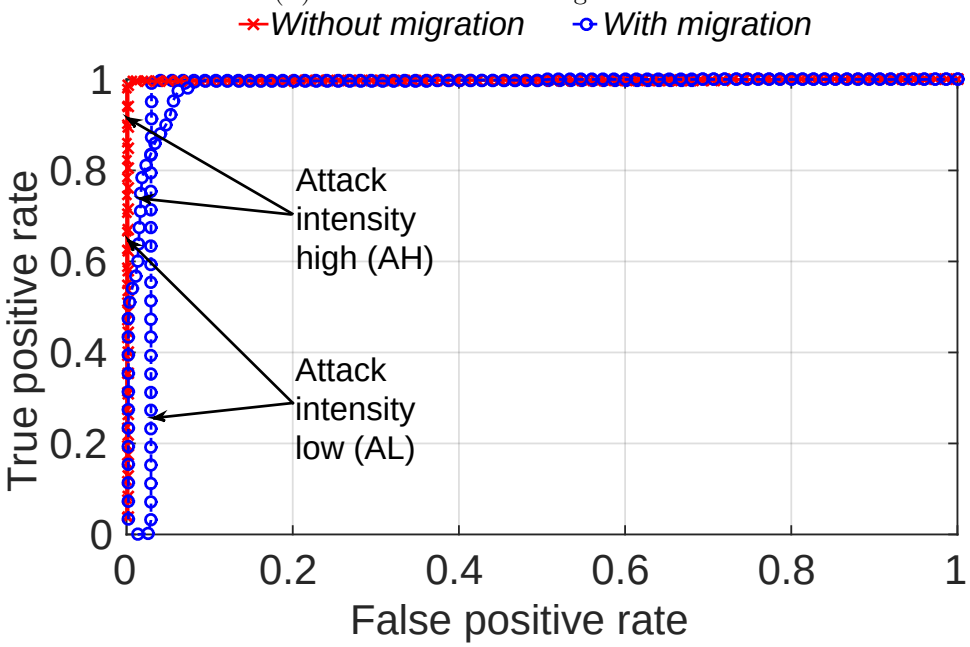
For illustration, two Anomaly score graphs (ASG) are presented here. Figure. 3.18a and Figure. 3.18b show some typical ASG obtained by using K-means for netscan and PCA for DoS respectively. High scores occurring within pink regions in each ASG plot indicate that each type of attack was successfully detected, i.e., these would be regarded as true positives. However, there are also high-scoring bins when only migration occurred (the green zones with no surrounding pink, at around bins 1350 and 1950), which would be regarded as false positives.

From the experimental runs it was observed that the presence of migration degrades the detection performance of both detectors for both the volume-based (DoS) and non-volume-based (NS) attack types.

Given the ROC curve outputs for K-means under netscan and DoS attack scenarios depicted at Figure. 3.16a and Figure. 3.16b respectively, it was noticed that the K-means method is sensitive to the chosen anomaly intensity and type. Also, in some instances its performance degrades even



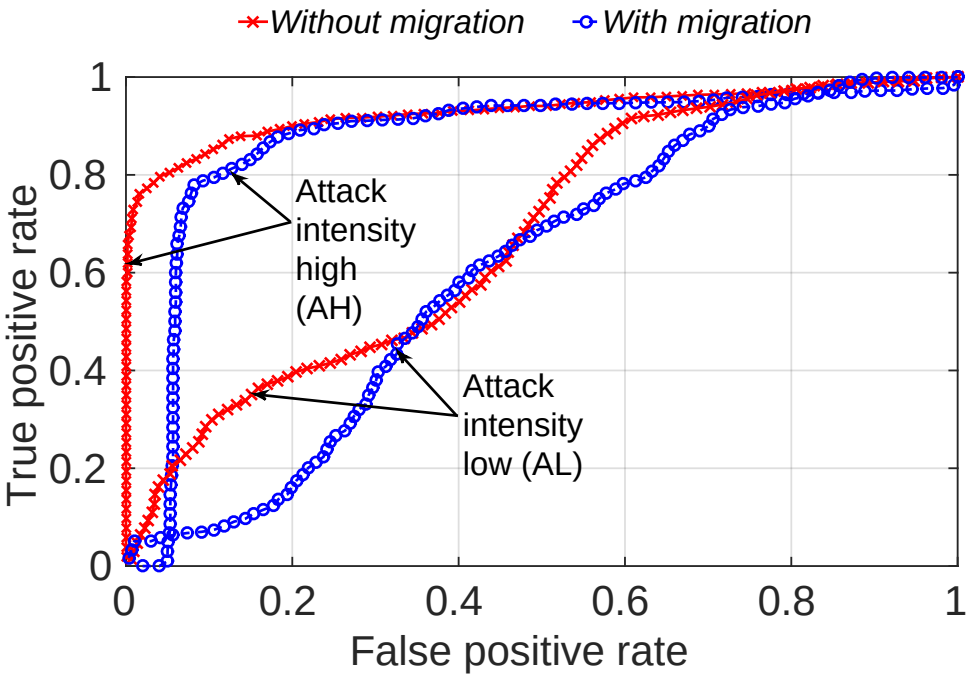
(A) ROC for netscan using K-means



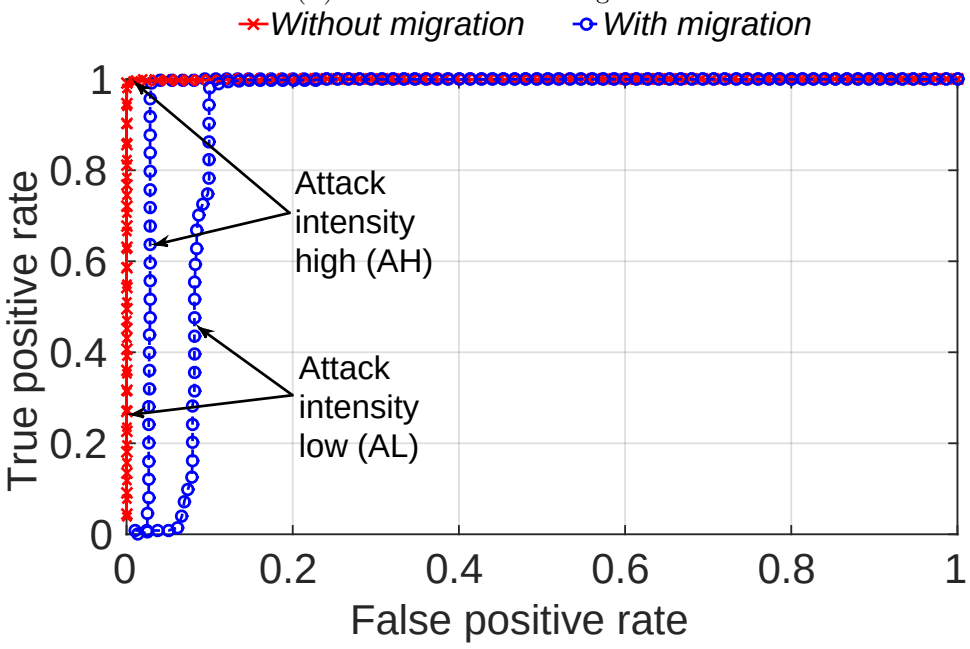
(B) ROC for DoS using K-means

FIGURE 3.16: ROC for attacks using K-means

more while live migration occurs. However, for a volume-based attack such as DoS, K-means performs better in detecting both low and high intensity attacks but still its performance is negatively affected during migration. In particular, it can be observed that for low-intensity netscan, the true positive rate (TPR) is dropped by 35% in the event of a migration. In contrast under high attack intensity, the 65% TPR is achieved in face of migration with less



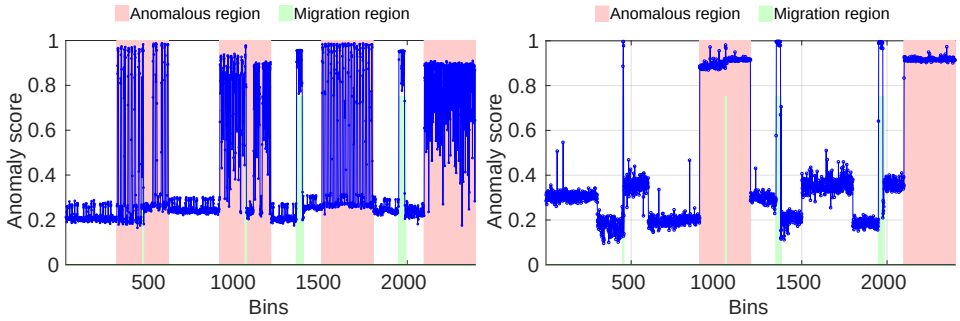
(A) ROC for netscan using PCA



(B) ROC for DoS using PCA

FIGURE 3.17: ROC for DoS using PCA

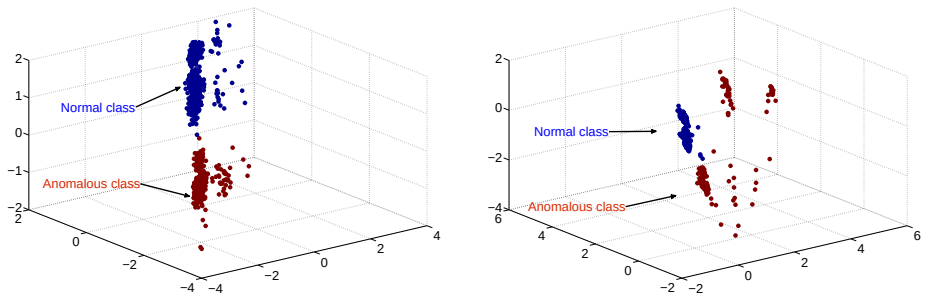
than 10% false positive rates (FPR). Figure. 3.16b quantifies the effect of migration for volume-based attack (DoS), and it can be seen that more than 80% of the TPR is achieved with and without migration. There is a 5% rise of FPR when migration is introduced, which is acceptably small. In parallel, it can be seen in Figure. 3.17a that almost 80% of TPR is achieved under high-intensity netscan with and without migration for the PCA approach. Therefore, it is concluded that migration has considerably less effect under



(A) ASG for netscan using K-means (B) ASG for DoS using PCA

FIGURE 3.18: Anomaly score graphs using K-means & PCA

high-intensity netscan on PCA. However, for the same attack type under low-intensity, the migration affect is more visible. In particular, the migration process impacts the performance by 30% when TPR reaches just about 20% yielding unacceptable number of false positives. For DoS, the PCA performs better comparatively for both attack types and is less affected under migration. As evidenced in Figure. 3.17b the FPR rate is less than 10% for both high and low intensities for more than 80% of the TPR. Finally, the thesis argue that the detection sensitivity as evidenced by the ROC curves is truly affected by migration. This can also be illustrated by Figure. 3.19, where the changing behaviour of features' data points are illustrated. In more detail, Figure. 3.19a and Figure. 3.19b show the effect of the first three features *(packet size, byte size and active flows)* using scatter plots. Clearly, the clusters have dispersed due to the change on the feature distributions caused by migration, thus, new test data would adopt different distances from the pre-defined cluster centroids, making detection accuracy to vary.



(A) Scatter plot for netscan without migration (B) Scatter plot for netscan with migration

FIGURE 3.19: Effect on clustering under migration where blue and red clusters represent normal and anomalous class respectively.

Obviously, a different number of clusters may result in better clusters,

e.g., if the migration already shows distinct periods of very low and very high traffic volume under normal conditions. However, the determination of an optimum number of clusters based on a cluster evaluation criterion is beyond the scope of this work. The results show that, under migration, the probability that a selected instance will be put in the correct cluster has decreased more. These results are similar to those obtained using PCA, indicating that VM migration makes the use of this form of AD technique unreliable for deployment of high assurance services. As expected, similar behaviour is observed for other scenarios (see Appendix A) which indicates that VM live migration affects the performance of several off-the-shelf detection techniques.

3.3.1 Analysis

Scenario	PCA	KM	NB	EMGM
PS-AH	worse	similar	similar	worse $TPR > 50\%$
PS-AL	worse $TPR > 45\%$	worse $TPR > 40\%$; better otherwise	similar	similar
NS-AH	slightly better	slightly worse	similar	similar
NS-AL	slightly better	better	better	better
DoS-AL-MT0	similar	inconclusive	mostly better	inconclusive
DoS-AH-MT0	inconclusive	better $TPR < 88\%$	inconclusive	similar
DoS-AL-MT1	slightly worse $TPR > 80\%$	similar	much better	much better
DoS-AH-MT1	slightly worse $TPR > 50\%$	inconclusive	much better	inconclusive

TABLE 3.2: Interpretation of ROCs for migration effect

Table 3.2 informally summarizes the change in performance of each detector against each scenario when faced with migration, based on interpretation of ROC. NB appears to be unhindered by the presence of migration, while EMGM is only significantly affected by PS-AH. PCA performs worse under PS and when the migrating node is under DoS.

Detection of PS rarely improves under migration, while a low-intensity NS is detected better by all detectors.

3.4 Summary

The number of experiments were conducted by varying migration and attack intensity, and examine the affect of migration. Results from the experiments shows that detection rate changes significantly with respect to higher/lower anomaly intensity. Hence, it can be concluded that migration affects the anomaly detection accuracy in a cloud environment. However, under some low intensities of attack migration can be classified as an anomaly. Such results show that virtual node migration has a higher impact on anomaly detection accuracy that results with higher anomaly intensity.

The choice of k and the choice of normalization technique are yet more variables to consider with these methods. Bin size should be adequate to provide enough data in each bin to support a given sample rate and a given rate of traffic. Longer traces are important so that an outlying anomaly has less effect on the normal subspace.

The experiments suffer from some flaws which could be improved. First, the testbed does not have separate business and management networks, so migration traffic naturally interferes with business traffic (affecting the results), and high-bandwidth attacks interfere with migration traffic (potentially preventing it, and the acquisition of a meaningful trace). It is also biased to one of the physical hosts, which keeps VM images on its own disc, while the other must mount it to access them, creating more business-management traffic interference.

In any dataset, most bins do not involve migration. So, when computing ROCs with $IMI = 1$, most bins are eliminated, reducing the statistical significance of the remainder. This could be solved by taking obtaining many more traces for the same scenario.

Chapter 4

Cloud Resilience Management

In cases where cloud computing is being used to support Critical Infrastructure (CI) services, it is particularly important for clouds and cloud-based services to be *resilient*, i.e., they are able to operate correctly and continuously even in the presence of challenges. The cloud resilience can be informally defined as the ability of a cloud to provide an acceptable level of service in light of challenges. A target level of service can be defined in a Service Level Agreement (SLA) using resilience metrics, such as availability and robustness metrics. The resilience is intended to be a fundamental property of Cloud service provisioning platforms. However, a number of recent outages of cloud data centre demonstrated that cloud resilience are not as resilient. The ENISA report on the security and resilience of cloud suggests: "*the availability of a cloud service is often dependent on the network used to access it and measure should be taken to ensure the resilience of access networks*" [30]. Currently there is little understanding about how to provide a resilient cloud infrastructure that can address challenges in a coordinated manner. To do this, a number of *resilience-supporting mechanisms* are needed at various locations and levels (tenant[1] and physical infrastructures), such as provisioning of redundant components, traffic capturing, anomaly detection and classification, and network reconfiguration. However, it is non-trivial to co-ordinate such mechanisms for several reasons:

- They might work on different time-scales. For example, the deployment function of cloud orchestration might foresee the need for scalable components and react to load changes on a longer timescale by scaling out a service, whereas reacting to an anomaly requires short-term decision-making to mitigate a potential attack.

- Distinct mechanisms might also have different objectives, and attempt to perform conflicting actions. A reaction to an anomaly might be to migrate VMs away from an affected node, but these migrations might conflict with standing policies to keep some VMs apart (because they belong to rival tenants) or together (to reduce latency).

[1] In the NIST cloud computing reference architecture [24] the term tenant is used for consumers who use the cloud based services.

- Reconfiguration of the mechanisms could unexpectedly produce such conflicts and discrepancies.

The related work on resilience is highlighted in Section 2.3.3. The current approaches to ensuring resilience of cloud services do not take a holistic end-to-end approach by considering both networking and system concerns, and how various mechanisms can be applied collectively to mitigate challenges. This can lead to challenges that can severely affect the overall availability of a service.

This chapter introduce a *Cloud Resilience Management Framework* (CRMF), which has the goal to facilitate the interactions of components that implement resilience mechanisms, as well as the (re-)configuration of such a system. The components of this framework aim to detect challenges and events in the system. The challenges could be Denial-of-Service, operational overload, misconfiguration, cyber-attacks, and to react to these events by reconfiguration, in order to maintain the operation of the system and, therefore, the critical services running on it. In order to confront these challenges, a strategy is defined which relies on a number of mechanisms that must co-operatively enforce the resilience of the network, including a rate limiter, and an anomaly classifier. The policies are used to configure and coordinate the interactions between these mechanisms. The use of policies to define configurations of mechanisms that can ensure the resilience of networked systems has been previously advocated in [120, 121, 89, 6]. By having a resilience strategy implemented with the aid of policies, in contrast to having it hard-coded, one can easily change it by adding or removing policies, thereby permitting the modification of the strategy during run-time. Policies sit idle until their events occur and their conditions are met. Their actions may cause reconfiguration of the system, including the deployment or activation of new components, which can in turn generate new events, and trigger further policies. This continuous process will constitute a policy-driven resilience feedback control-loop, which allows (for example) an initial network anomaly to activate closer inspection of traffic, and eventually reconfiguration of a firewall, automatically handling a challenge to the system, and without having expensive mechanisms (such as the closer traffic inspection) active at all times (see Chapter 2 for more details).

It is of paramount importance that cloud administrators are able to effectively manage the underlying technologies which enable configurations, elasticity, dynamic invocation of services, and monitoring activities such that security and resilience requirements of the critical service users and providers are met. The set of policies defining the possible reconfiguration actions is not fixed, and different policies may be loaded or unloaded over time to reflect better resilience practices or a better understanding of the challenges. The expression of resilience mechanisms and strategies as policies allows new

policies to be studied and analysed off-line before deployment, to determine correct interaction with existing policies.

This chapter has the following structure. Section 4.1 presents the resilience management framework and Section 4.2 provides more specific details about its components. Section 4.3 evaluates the framework qualitatively against the considered use case. Section 4.4 concludes the chapter.

4.1 Cloud Resilience Management Framework

The Cloud Resilience Management Framework (CRMF) defines several automatic functions of an administrative cloud domain, and at various levels of the architecture, and how they interact to make the domain resilient to unplanned challenges. The CRMF consists of two main functions, the *Deployment Function (DF)* and *Resilience Manager (RM)*. The Deployment Function is concerned with supplying the Virtual or Cloud Infrastructure Manager with configurations describing the creation and deployment of virtual machines for instantiating a service. It ensures that virtual instances are created and destroyed according to service requirements (including resilience to high load). If high-availability service components needing (for example) backup VMs are defined, it deploys redundant instances for higher availability and places them so that the given levels of availability of the service components are met. DF also ensures that the necessary components of resilience management (data collectors, anomaly detectors, etc.) are correctly instantiated with the virtual instances. On the one hand, DF acts as controller for RMs by placing virtual machines in parts of the infrastructure supervised by an RM instance. Alternatively, it can also provide input to the RMs in terms of types of instances started and their respective locations, which may allow detection mechanisms to work. Details of the deployment function are presented in Section 4.2.1 below.

Each Resilience Manager (RM) is composed of three software components, which are shown in Figure. 4.2. (**A** in the figure represents a single hardware node in the cloud, and **B** represents the RM, with the aforementioned software engines residing within it.) For simplicity, only three nodes are shown and the network links that connect them have been omitted. The anomalies of particular interest are those caused by any activity associated with anomalous behaviour in cloud such as malware on cloud VMs or Denial-of-service attack on host. The software components within each RMs are: The *Anomaly Detection Engine (ADE)*, *Policy Engine (PE)* and the *Coordination and Organization Engine (COE)*. The RM on each physical node performs local detection based on features gathered which is done using subcomponent called *Data Collection Engine (DCE)* from its nodes' VM and its local network view; this is handled by the *System Analysis Engine (SAE)*

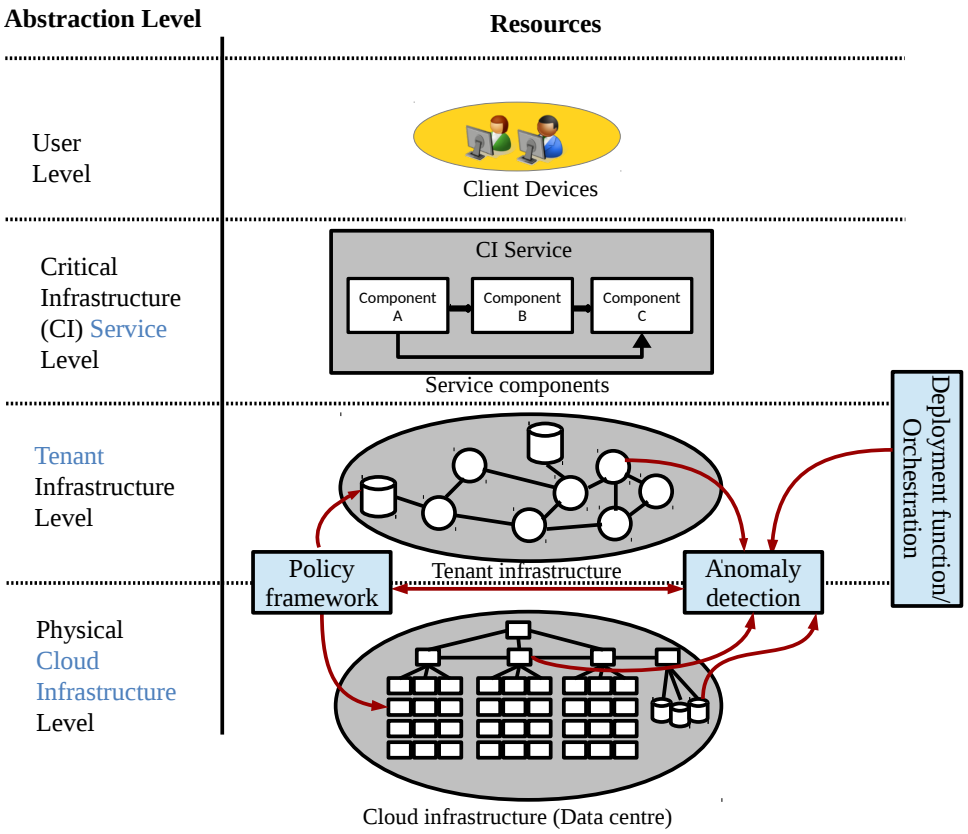


FIGURE 4.1: Components mapping to architecture

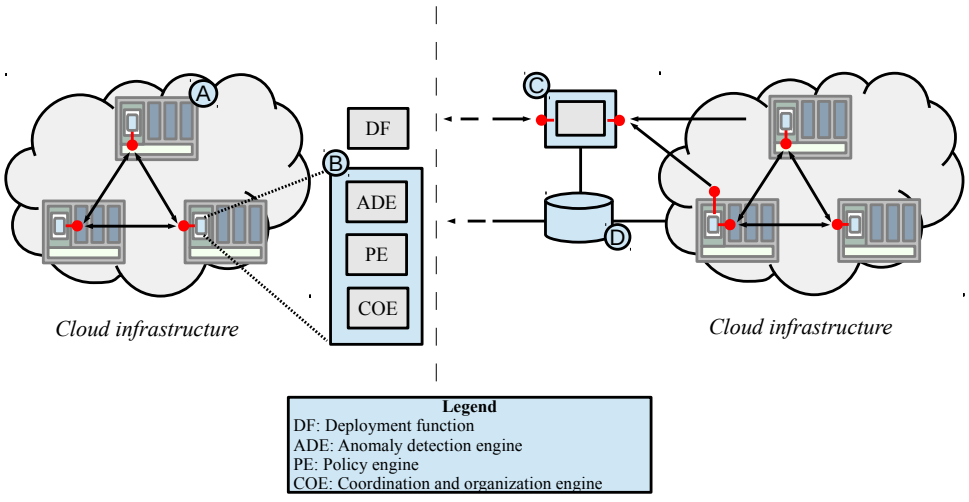


FIGURE 4.2: An overview of the CRMF architecture

and *Network Analysis Engine (NAE)* components respectively. The PE component is in charge of remediation and recovery actions based on the output from the analysis engines (i.e. NAE and SAE), which is conveyed to it by the COE. Finally, the COE component coordinates and disseminates information between other instances and the components within its own node. It is ultimately in charge of the maintenance of the connections between its

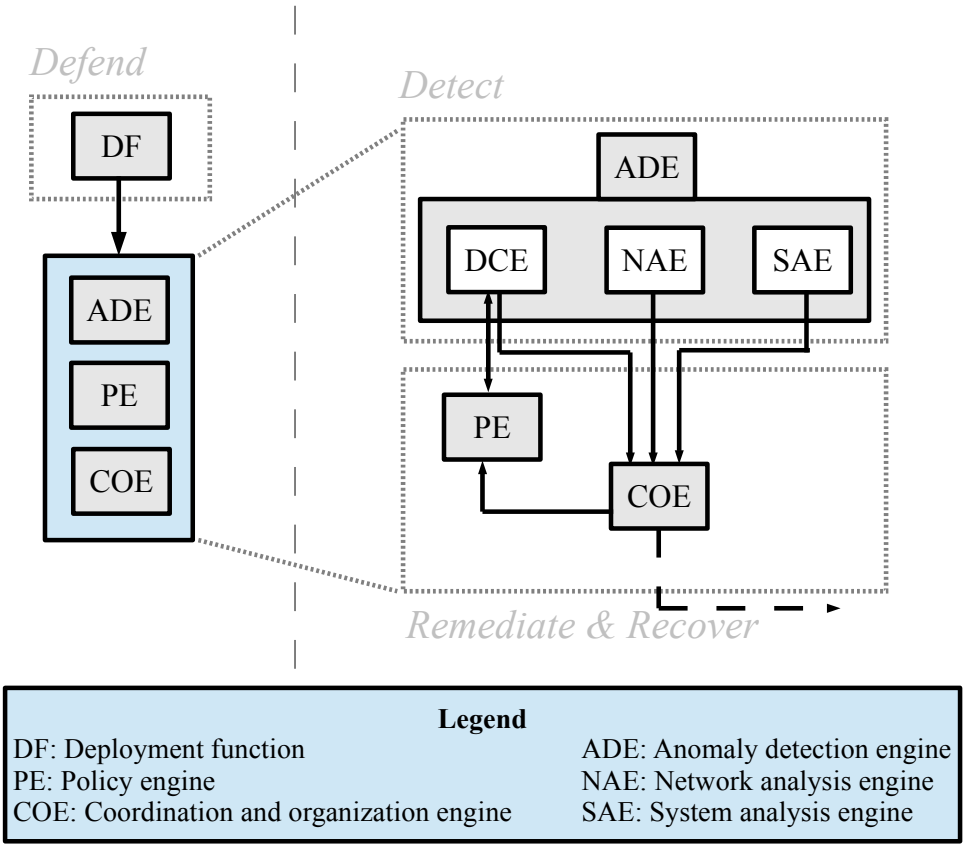


FIGURE 4.3: The flow of information and hierarchy of engines within a CRMF

RM's peers and embodying the self-organizing aspect of the overall system. The COE acts autonomously to control the other components within its RM while at the same communicating with its RM peers.The dashed arrow in Figure. 4.3 indicates communication between the local COE and the remote COEs of each peer RM. In practice, and as depicted in Figure. 4.2, there are various system/network interfaces that act as information dispatch points in order to allow efficient event dissemination [90]. In addition to physical node level resilience, the CRMF is capable of gathering and analysing data at the network component level through the deployment of network RMs as shown by **C** in Figure. 4.2. The DCE can also gather features from all traffic passing through ingress/egress points of the local network (D in Figure. 4.2). More detail about individual components will follow in subsequent sections below.

4.2 CRMF Component Details

Based on the presented CRMF design and the D^2R^2+DR resilience strategy described in Chapter 2, Section 2.3.1, illustrates how the cloud infrastructure could be enhanced to cope with challenges.

4.2.1 Deployment Function

The deployment function concerns itself with supplying the virtual or cloud Infrastructure manager with configurations describing the creation and deployment of virtual machines for the critical infrastructure. Its task is to translate high-level service descriptions and SLAs into automatically deployable descriptions, such as Heat templates in the OpenStack environment. It is therefore a part of a Tenant Infrastructure Management System (TIMS), being located between the cloud user and the Cloud Infrastructure Operator, Figure. 2.6. In particular, a deployment function is needed that places instances of virtual resources according to the resilience and performance requirements of the service user. For example, the placement of virtual machines can be done in cooperation with the anomaly detection component using *Heat* templates in the *OpenStack* environment, towards at least two ends: a) to place critical instances in AD-supervised parts of the infrastructure, and b) to provide input to the AD in terms of types of instances started and their respective locations, which may allow to conclude standard traffic and behavioural patterns of and between these instances. The aforementioned functionality can be separated into the following sub-components (Figure. 4.4), each with a specific task (see also Chapter 6):

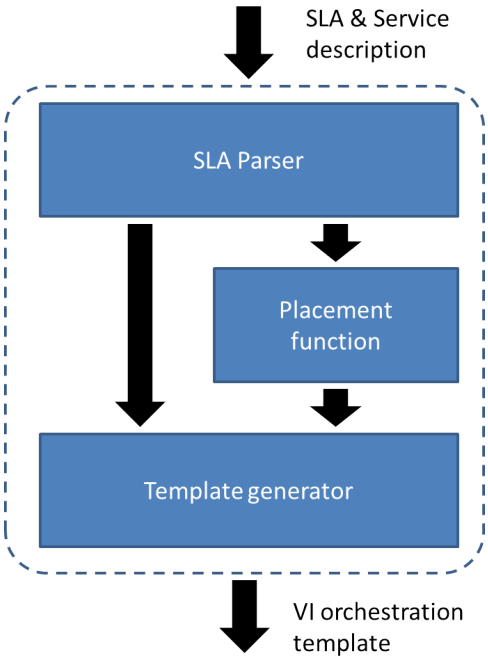


FIGURE 4.4: Internal structure of the orchestration component [126]

SLA Parser

This subcomponent is responsible for parsing a user description of their service, i.e., the needed service components and their relation/connectivity, as well as the required performance and availability.

Placement Function

The requirements extracted by the SLA parser translate into a set of virtual machines that need to be deployed in the physical infrastructure. The placement function maps these instances to the offered infrastructure of the cloud infrastructure provider.

Deployment Template Generator

Finally, the mapping must be provided in a form that is processable by the cloud management system. This means generating a deployment, e.g., a Heat template, using as input the placement and the additional service description parts generated by the SLA parser.

4.2.2 Anomaly Detection Engine

The anomaly detection engine is one of the core components of CRMF and comprises of three sub-components: Data Collection Engine (DCE), Software Analysis Engine (SAE) and Network Analysis Engine (NAE). The DCE monitors and collects data from each VM and host for various metrics in order to produce a feature vector for subsequent detection engines (i.e NAE and SAE). System and network level engines identify potential symptoms of anomalous behaviour without performing a lot of computation. They denote the presence of anomalies, but merely suggest the possibility of one. Further analysis would be needed to confirm if anomaly is present and if so, classify it. The initial event is generated by looking at each metric for system and network level in isolation. Hence, an event generated indicates that either host or VM is experiencing anomaly. A broader correlation between metrics can be very expensive as the number of such comparisons grows exponentially with the number of VMs and the monitored metrics.

A *Fine Grain Analysis (FGA)* is then required that uses classification algorithms such as *AutoClass* [103] to identify the cause of anomalies and localize them. The idea is to generate and update inference rules, in a fully unsupervised manner, creating different classes of anomalies being detected in sucha a ways that it allows to update fuzzy inference rules autonomously for efficient localization of anomalies [83, 103]. This analysis of the monitored data is required to further understand anomalous behaviour and to narrow down the scope of remediation. It can also generate an alert in case the anomaly can not be classified for which manual intervention is required.

Below section give a brief overview of sub-components of anomaly detection engine.

Data Collection Engine

An essential goal for resolving the resilience puzzle is what information should be provided and where the information should come from. Therefore, the DCE is designed in such a way that it can collect and process various relevant metrics pertaining to system (such as CPU, memory etc.) and network (such as number of packets, number of bytes and throughput) for every VM and physical host. All metrics are collected at periodic intervals with a configurable monitoring interval parameter. It has sub-components which perform normalization and smoothing of data and produce feature vectors which are a sequence of values over a fixed interval of time and form the basic input into subsequent detection engines (i.e, NAE and SAE).

The first stage of online detection is data collection which is composed of a set of scripts providing feature extraction and normalisation, which within the SAE is achieved through the use of Volatility[2] in conjunction with libVMI[3]. At 3 second intervals the Volatility tool is invoked with the custom plug-in that crawls VM memory for every resident process structure. From each process a number of raw features are extracted which include:

- the current size of virtual memory belonging to each process
- the peak virtual size (i.e. the requested memory allocation) of each process
- the number of threads belonging to each process
- the total number of handles belonging to each process (which includes process threads, file handles, registry entries, etc.)

The raw features are per process, which is not useful for each sample, or snapshot, as a single feature vector. Therefore, the raw features are used to build meta-features which include: the *mean*, *variance* and *standard deviation* of each feature across all processes. The result of feature extraction is a feature vector of the form $x = (x_1, x_2, \ldots, x_n)$, where n = 12 due to the three groups of four meta-features.

At the network level the NAE collects traffic data through tcpdump[4] from each host's network at bridge interface (br0). This traffic is then passed onto a *Summary Extraction Script* which is based on libpcap[5] and converts the traffic into normalised statistical properties as per packet basis. In order

[2]Volatility framework:https://code.google.com/p/volatility/
[3]libVMI: https://code.google.com/p/vmitools/
[4]tcpdump/libpcap: http://www.tcpdump.org/
[5]libpcap API:http://www.tcpdump.org/

to capture the dynamics of varying attack types, both the volume-based features (e.g., count of bytes and packets) and distribution-based features (computed as the Shannon entropy of all values observed in the bin, as used in many seminal pieces of work [78]) are extracted. The resulting feature vector therefore has dimension n = 8 and contains:

- Number of packets
- Number of bytes
- Number of active flows in each bin
- Entropy of source IP address distribution
- Entropy of destination IP address distribution
- Entropy of source port distribution
- Entropy of destination port distribution
- Entropy of packet size distribution

Figure. 4.5 below shows the overview of data collection engine.

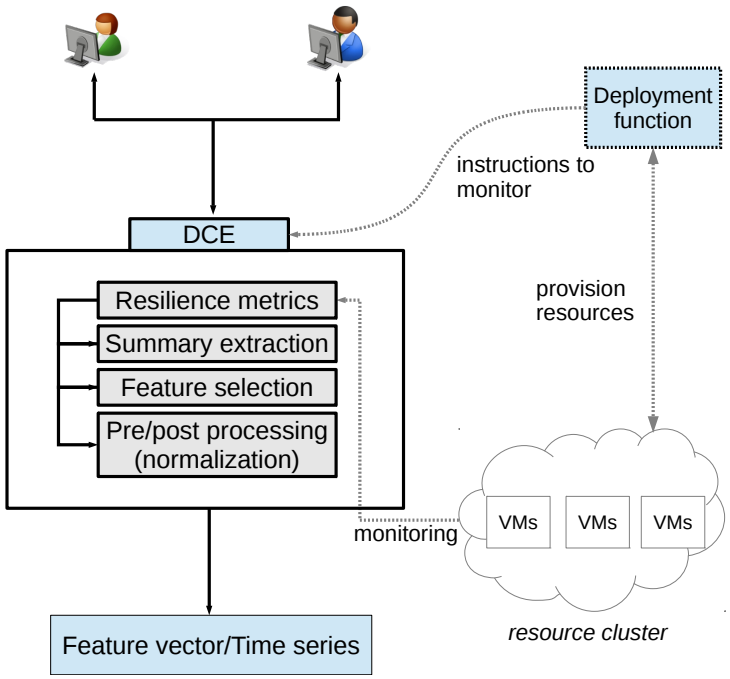


FIGURE 4.5: Overview of the Data Collection Engine

Network Analysis Engine

The purpose of the NAE is to detect anomalous traffic at the physical node level of the cloud. This is achieved by modelling normal traffic patterns

and identifying anomalies through online/offline monitoring of traffic on the network interfaces of the cloud node with aid of DCE. The NAE provides a reference implementation of different anomaly detection techniques for offline and online analysis. The one-class Support Vector Machine (SVM) algorithm is chosen for the implementation of SAE and the Recursive Density Estimation (RDE) [12] technique is used for the implementation of NAE.

System Analysis Engine

The System Analysis Engine (SAE) is designed to detect anomalies through the observation of VM properties. The SAE builds a model of normal VM operation and detects deviation from the normal through selected anomaly detection technique.

Using the toolchain and experimental setup described in Chapter 3, the detection aspects of the system and network wide unified resilience architecture has been tested under malware scenario. The *Kelihos (Trojan.Kelihos-5)* trojan is chosen to test the performance of SAE and NAE components. Since the trojan is *Win.32* binary, which allows it to be executed on the target VM. Upon execution, the malware spawns many child process and subsequently exits from its main process. This is likely an obfuscation method to avoid detection, but has the effect of skewing various features (system and network) resulting in an anomaly. Based on the features obtained from DCE the system and network features are aggregated into a single dataset. This dataset is then applied to the detector (implementing PCA technique).

In order to validate if features provided by DCE can also apply in the elastic scenario of cloud, the VM migration is performed during experiments. The results indicated that malware is spread and re-initiated when intra or inter-cloud VM/Service migration is performed. For the experiment presented in Figure. 4.6, each VM runs Apache HTTPd. The client host runs custom scripts to initiate random HTTP requests from the VMs. The SAE acquires VM memory at the hypervisor level using introspection to collect raw-system-features such as process and memory usage. For 20 minutes run, web traffic occurs continuously at a fixed rate, and so generate the system-level (normal) activity. At 9 minutes into a run, Kelihos is injected, to generate malicious activity in system. At exactly 10 minutes, a migration of infected VM is initiated. A run therefore generate two 10 minutes system-level datasets (features) as result of migration, the trace from the node of the arriving VM and from the node of the departing VM.

Each dataset generated is divided into 3-second bins, and each bin is converted into a (200×12) and (200×8) system and network level feature vectors per node respectively. This yielded 400×12 and 400×8 feature vector for outward and inward node. The combined feature vector is submitted to PCA based detector to obtain the k-subspace which corresponds to the

normal behaviour of the traffic, and spans from pc_1, through pc_k, whereas the remaining subspace (i.e, pc_{k+1} through pc_m) maps the anomalous behaviour with respect to the variance of the dataset. Subsequently, the magnitude of the projection of data point x_i is computed into the anomalous subspace to quantify its malicious behaviour which is used to produce a Anomaly Score Graph (ASG). ASG is a time-series representation which summarizes the anomalous score of each bin in the trace indicating how anomalous each time bin is with respect to others.

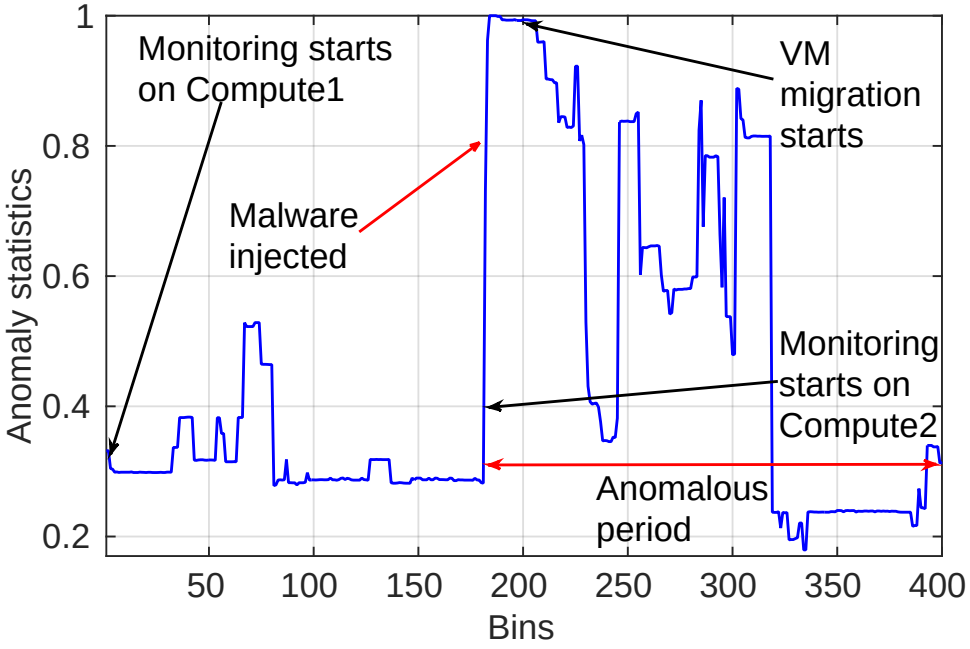


FIGURE 4.6: Results of detection for *Kelihos* using system and network wide features

4.2.3 Policy Engine

The central component of the CRMF is the Policy Engine (PE) as shown in Figure 4.7, which makes use of a policy-based decision, to activate management actions on the Cloud infrastructure at both the VM and host level, as well as functionality exposed by existing network elements, such as routers and switches. The policy engine is using IND^{2}UCE[6] which is a policy enforcement framework based on the ECA principle. In the CRMF context, output from the anomaly detection (AD) or from the fine grain analysis (FGA) is used as a trigger event for the policies that can perform remediation actions. The output depends on the detection algorithm in use by the components, but is ultimately an indication of the current health of the VMs and the host on which the VM is residing and takes the form of an event. Depending

[6] http://www.iese.fraunhofer.de/en/competencies/security/usage_control/philosophy_uc.html

on the deployed policy mechanisms, remediation actions can be performed in the system such as migrating virtual machines to a dedicated host in the cloud environment (sandboxing) or starting another instance of the virtual machine to compensate overload.

An incremental approach is proposed to challenge identification, one of the outputs of situational comprehension, whereby an evolving understanding of the nature of a challenge is developed. The aim is to enable early remediation to protect Cloud Infrastructure from potential collapse, using imperfect information, and subsequently use more-specific remediation activities as a more detailed picture is constructed, i.e., as comprehension improves. This approach is similar to that proposed by [47]. However, the proposed approach introduce greater flexibility and re-usability of identification strategies by using policies to orchestrate the identification process. When a challenge has

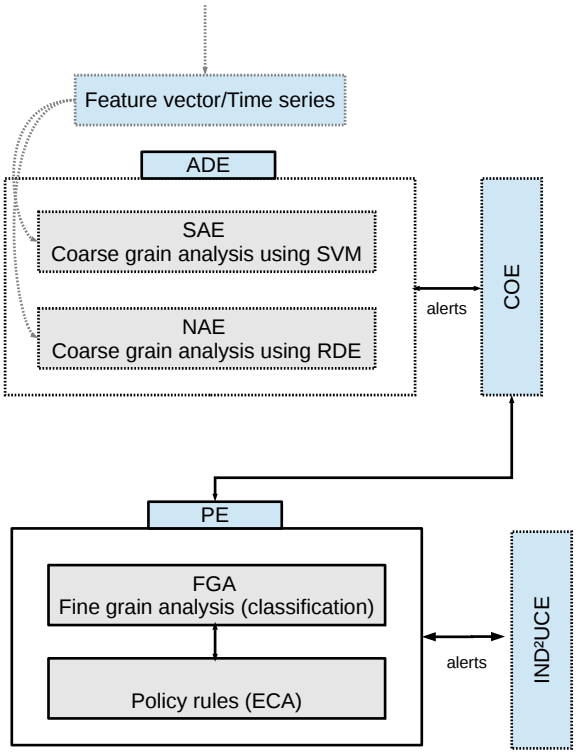


FIGURE 4.7: Overview of the Policy Engine

been identified, or an hypothesis about the nature of an ongoing challenge is reached, the anomaly detection engine generates an alert. As an example of how the architecture can be applied, consider a high traffic volume challenge, such as a Distributed Denial of Service (DDoS) attack or a flash crowd event. Initially, NAE generates an alarm as soon as the volume of traffic goes over a given threshold. On detection of high traffic volumes, the link on the host could be rate limited, e.g., by randomly dropping packets, to protect the challenge target. A Fine Grain Analysis (FGA) engine could then be invoked to determine the target of the challenge, leading to more specific rate

limiting of traffic to this destination. Finally, actions can then be enforced using policy enforcement points such as resulting in flows being blocked if they are deemed to be malicious, or re-directed or no longer subjected to rate-limiting if they are seen to be benign.

Below Listing 4.1, lists a few example obligation policies that can be used to reconfigure resilience services in response to events generated by monitoring mechanisms, NAE and FGA. This example highlights the use of refined policies to respond to challenges such as a Denial of service (DoS) attack targeted at the cloud infrastructure layer. In order to confront these challenges, resilience mechanisms can be used that must co-ordinate and co-operate to ensure resilience. Clearly, it is important that an attack be mitigated rapidly to reduce the impact on the other tenants and protect the infrastructure. Such mechanisms include but are not limited to anomaly detector, flow classifier, resilience metrics reporter, malware differentiators etc.

4.3 Qualitative Evaluation

Below a use-case is presented which describes the type of Cloud infrastructure and services CRMF is expected to support. A scenario is considered of a critical infrastructure provider, in this example a traffic control centre, that in the past has not used any cloud service and stores all information on local servers, where also the corresponding services and applications are run. The managers of the infrastructure would like to move, as a first step, some data to the cloud, they consider that it can save some operational and maintenance costs, but they are concerned about security, availability once data has been moved to the cloud, the occurrence of any malfunction or undesired behaviour in the network may cause failures in the system. This is the basis for a aforementioned test-case in order to evaluate the CRMF and how different components interact to provide overall resilience.

Service of traffic management system is not available due to Denial-of-Service attack is a test-case focuses on the *"cloudification"* of the traffic control system. The system receives traffic information from sensors (e.g. how many cars pass a specific junction in a given time span) and analyses it, for example to control traffic lights in order to direct the traffic flow efficiently (see [25] for details). The *Traffic Management System* (TMS) takes advantage of information that can be provided by roadside traffic sensors. The traffic control centre uses available traffic information to develop optimal traffic control strategies addressing traffic needs at all single intersections, a given corridor, or throughout a given area.

The traffic forecasting system relies on several databases to predict what will happen next, with regards to traffic flow in the city. Several kinds of

```
on highUtilisation(link, VM_ID)
        do FlowExporter enable(link, VM_ID) && sandBox (VM_ID,
            newLocation);

Flow exporter is disabled when link utilisation decreases (green
    policy)
and VM will not be sandboxed for further analysis.

on lowUtilisation(link, VM_ID)
        if (LocalManager.anomalyList isEmpty(link, VM_ID))
                do FlowExporter disable(link, VM_ID);


Configuration policy for handling high risk alert
on highRisk (link, src, dst, VM_ID)
        do
        {
              FlowExporter notify(highRisk(link, VM_ID));
              LocalManager.anomalyList add(link, src, dst);
        }

Configuration policy for handling high risk alert
on highRisk (link, src, dst, VM_ID)
        if (LinkMonitor getUtilisation() >= 75%)
                do RateLimiter limit(link, 60%);

Configure local manager for handling Fine grain analysis
on FGA_classification(flow, value, confidence, VM_ID)
        if ((value == DDoS) && (confidence < 0.4))
                do
                {
                      Visualisation notify(alert(high));
                      RateLimiter limit(flow.src, flow.dest, x%);
                }
        if ((value == DDoS) && (confidence >= 0.4) && (confidence <=
            0.8))
                do
                {
                      Visualisation notify(alert(high));
                      RateLimiter limit(flow.src, flow.dest, y%);
                }
        if ((value == DDoS) && (confidence > 0.8))
                do
                {
                      Visualisation notify(alert(high));
                      Firewall block(flow.src, flow.dest);
                }

Configure local manager for handling low risk alert (recovery)
on lowRisk (link, src, dst)
        if ((LocalManager.anomalyList remove(link, src, dst, VM_ID))
            isEmpty(link))
                do
                {
                        FlowExporter notify(lowRisk(link, VM_ID));
                        RateLimiter limit(link, 100%);
                }
```

Listing 4.1: Example obligation policies for resilience

data are processed, including historical and actual ones plus data coming from simulations that traffic operators normally perform. The processed data is stored in a traffic forecasting database that is hosted in the cloud. A situation is considered in which parts of the databases that the system uses for its traffic forecasting system are not processed. Due to malfunction or misbehaviour such as DDoS attack on the database server data is not

available, rendering the system unusable. The operators of the traffic control system want to know where the malfunction or misbehaviour occurred: cloud or central premises. Hence a focus lies on the usage of the resilience management framework (specifically anomaly detection) to detect and recover the system to its normal operation.

To demonstrate the feasibility of the proposed approach the results are shown from an earlier simulated high traffic volume example as described in [149] and slightly modified for the considered use-case. The deployment function starts off by analysing the security requirements and instantiating VMs for traffic management and forecasting systems, and places critical instances, such as the databases and the TMS, on machines that are supervised by anomaly detection. Moreover, it will deploy redundant instances of such high-availability components at a sufficient degree of separation in the physical infrastructure (the configuration of a load distribution entity or active/backup server instances are application dependent and need to be provided by the service operator). Components that need to deal with varying load can be deployed in a scalability group.

DCE starts collecting network (such as incoming ingress and egress links) and system (VMs) data to create the feature vectors for NAE and SAE respectively. Figure. 4.8 shows the onset of a DDoS attack on the ingress

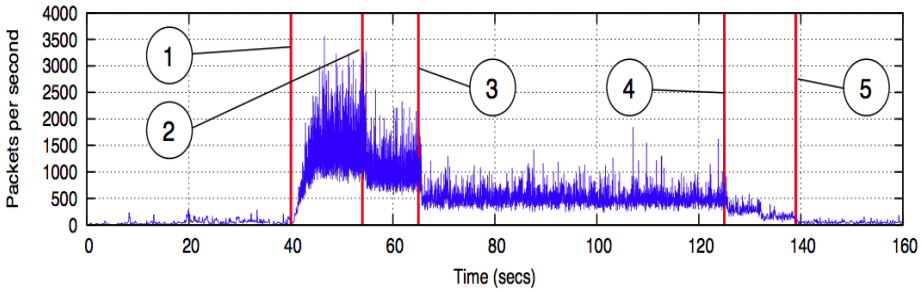


FIGURE 4.8: Results from simulations that implement an incremental detection and remediation approach to a DDoS attack [149]

link of a host at approximately 40 seconds (1). The raising of an alarm by a NAE is seen at 55 seconds (2), whereby deviation from normal behaviour is detected in the form of an excess load of traffic. Shortly thereafter, the effects of the initial rate limiting of the link by a rate limiter remedy can be observed. The limiter is configured by coarse grain resilience policies as described in Policy Engine (PE), to discard 70% of all incoming traffic in order to protect traffic management servers and infrastructure.

At same time the PE initiates a fine grain analysis (FGA) which performs classification to localise the anomalies. At 65 seconds (3), the FGA identifies the destination IP address of the victim. This is achieved in this case by

examining the destination address of each incoming packet, and raising an event when one destination accounts for 60% of all packets. The rate limiter is now reconfigured to drop 70% of the traffic destined for the victim only. Some legitimate traffic that is not destined for the victim, which previously was blocked, is now not filtered. Also, a classifier is initiated at (3) and flow exporting from the ingress link has started. Hereafter, the FGA classifies, receiving flow records from the flow exporter, attempts to identify the specific attack flows. At 125 seconds (4), rate limiting is confined just to the attack flow and legitimate traffic to the traffic management/forecasting service continue. After 139 seconds (5), all the malicious traffic is blocked bringing back the traffic management system to normal operational state.

4.4 Summary

The idea behind cloud computing has been around for several decades in the form of utility computing. However, the lack of mature virtualization originally prevented its growth. Recent advancements in virtualization have created an environment for the deployment of cloud even for CI. Although relatively new, a fair amount of work [88, 127] has been done in order to examine current and future challenges for both users and providers of cloud computing with respect to security and resilience.

This chapter introduces a Cloud Resilience Management Framework, which has a goal to facilitate the interactions of components that implement resilience mechanisms, as well as the (re-)configuration of such a system. The components of this framework aim to detect challenges and events in the system, such as an attack or a network overload, and to react to these events by reconfiguration, in order to maintain the operation of the system and, therefore, the critical services running on it. Many of the decisions to react to events are expressed as Event-Condition-Action (ECA) policies, which bind the components and mechanisms together. Policies sit idle until their events occur and their conditions are met. Their actions may cause re-configuration of the system, include the deployment or activation of new components, which can in turn generate new events, and trigger further policies. This continuous process will constitute a policy-driven resilience feedback control-loop, which allows (for example) an initial network anomaly to activate closer inspection of traffic, and eventually reconfiguration of a firewall, automatically handling a challenge to the system, and without having expensive mechanisms (such as the closer traffic inspection) active at all times. The salient features of the proposed cloud resilience management framework are detailed below.

- It is designed to be reactive and pro-active to challenges for fast detection and identification, preferably before challenges causes noticeable degradation.

- Modular design that allow to re-use resilience strategies. This is imperative in case there is failure of surrounding components.
- The components of framework are easy to integrate with each other to allow complete flexibility and adaptability.
- Easy to modify or integrate as per security and resilience requirements.
- Provide easy integration with existing cloud management systems.
- CRMF provide functionality for efficient and effective management of resilience where needed.
- Use of policies to realize overall resilience.
- Composed of various distributed components to provide overall resilience which is desirable in case of failure of individual components.

Chapter 5

Anomaly Detection using Data Density

Cloud environments are hugely popular because they offer a range of beneficial properties, such as on-demand self-service, resource pooling, rapid elasticity and measured service, using virtualization as an enabling base technology [94]. However, they are susceptible to traditional challenges including Denial-of-Service (DoS) attacks, malware and misconfiguration. In addition, cloud properties can also exacerbate these challenges in both their detection and impact, hence these challenges increase the management costs, and lead to Service Level Agreement (SLA) violations [146] [23]. A recent study by security-as-a-service provider Alert Logic, indicates that application attacks aimed at cloud deployments grew 45% over the previous year (2014) [82]. Their research is based on an analysis of one billion events during the whole of 2014 across more than 3000 of its customers.

Motivated predominantly by cost reduction, cloud environments are also being used by sectors operating critical services, such as safety-critical operations (e.g., Air Traffic Control networks), critical manufacturing services (e.g., utility networks and industrial control systems), and critical real-time services (e.g., transportation and surveillance systems) [71] [108] [5] [20]. For these critical infrastructures, there are stringent security and resilience requirements, which are arguably higher than traditional Information Technology (IT) services because attacks on these high-assurance IT services that support critical infrastructures could have severe implications. Consequently, the detection of anomalies, which can signal attacks and challenges, is a vital element of operations in clouds. The dynamic nature of cloud environments and the diverse workload patterns of the applications they run impact the detection ability of anomaly detection techniques [1]. Therefore, a technique which allows the construction, accumulation, and self-learning of a dynamically evolving information model of "normality", and which can operate in real time, would be of great value.

The previous work [128], investigated how intra-cloud live Virtual Machine (VM) migration affects state-of-the-art anomaly detection techniques,

potentially making them unreliable for services running in cloud environments. In order to quantify the impact of VM migration on Anomaly Detection (AD) techniques, and to understand their computational complexity, a toolchain is developed which provides reference implementation for six detectors based on K-means[1], PCA[2], Wavelet[3], GMM[4], SVM[5] and Naïve Bayesian[6] techniques. The results suggested that in some configurations challenge anomalies are missed and some configuration anomalies are wrongly classified. Moreover, it is noticed that the computational complexity of these techniques is high because they require complex statistical computations, as well as the storage of the complete historical metrics to detect anomalous patterns. Based on these results, the argument is made that there is a need for robust, preferably real-time, anomaly detection for cloud environments with high degree of accuracy where migration is recognized as normal.

There exist many techniques for anomaly detection in several disciplines, and they have exhibited sufficient detection accuracy in a variety of scenarios, as indicated in a comprehensive survey [33]. In the context of cloud computing, a few anomaly detection techniques have been adopted and redefined [143, 142, 55, 14]. However, these approaches use complex statistical measures, lack scalability, and often require prior knowledge of the behaviour of relevant patterns, making them potentially unusable in a cloud-operational context. Further, the detection techniques used in industry are threshold-based [63] [137] [48] which usually require upper/lower threshold values which comes from predictions on long-term historical data analysis or from predefined performance knowledge. The main issue with such methods is that they detect anomalies after they occur instead of noticing their imminent arrival.

In contrast, a proposed technique is lightweight and aims to address the scalability needs of cloud infrastructure. It employs a data density-based technique that can easily scale as metric volume grows because it does not require storing of metrics. The density computation is done on distribution of metrics rather than individual metric thresholds. In addition, there is no need for prior knowledge about anomalous behaviour, so it operates unsupervised. As a result, anomalies can be detected that are not well understood (i.e., there are no prior models) or have not been experienced previously.

The importance of anomaly detection in cloud computing is due to anomalies in data translating to important actionable information. For example, an anomalous traffic pattern could indicate the request rate for VMs suddenly increasing, which could eventually decrease service availability to authorized

[1] https://en.wikipedia.org/wiki/K-means_clustering
[2] https://en.wikipedia.org/wiki/Principal_component_analysis
[3] http://www.pybytes.com/pywavelets/regression/wp.html
[4] http://scikit-learn.org/stable/modules/mixture.html
[5] http://scikit-learn.org/stable/modules/svm.html
[6] http://scikit-learn.org/stable/modules/naive_bayes.html

users. In general, anomaly detection is applied at various levels of cloud computing, due to its native strength in identifying unknown attacks.

Wang et al. [141] proposed the EbAT system to allow the on-line analysis of multiple metrics obtained from system-level components (e.g., CPU utilization on rack servers, memory utilization, read/write counts of the OS, etc.). The system showed potential in detection accuracy and monitoring scalability, but it was not evaluated in the context of adequately pragmatic cloud scenarios. Guan et al. [58] and Garfinkel et al. [50] proposed multi-level anomaly detection techniques to detect intrusions at different levels of a cloud system. The techniques appear to be rather inflexible and the application of those techniques in an operational context requires better clarification. Lee et al. [81] proposed a multi-level approach, which provided fast detection of anomalies discovered in the system logs of each guest OS. One of its disadvantages is the apparent lack of scalability, since it required increasingly more resources under high system workload. Also, it is only specific to detection in logging data.

Similarly, Dastjerdi et al. [41] proposed an approach based on mobile agents for an intrusion detection system for cloud systems. However, scalability appears to be an issue due to the high number of virtual machines that are required to be attached to the agent. The authors in [105] instrumented a real-time adaptive anomaly detection framework that was able to detect anomalies through the analysis of runtime metrics using the traditional two-class Support Vector Machine (SVM) algorithm. However, the main issue raised by this study was that the formulation of the two-class SVM algorithm suffered from the data imbalance problem, which affected the training phase, and consequently led to several mis-classifications of newly tested anomalies.

PREPARE [134] and DAPA [73] are two recently proposed frameworks for performance evaluation based on anomaly detection for virtualized environments. Although, their main focus is to identify SLA violations. These frameworks only address application-related issues which are manifested into performance anomalies. However, none of these approaches focus on the impact of elasticity of the cloud such as VM live migration and high volume of data due to heterogeneity of the cloud. In this work an attempt is made to propose a technique that can work under these challenges.

The remainder of this chapter is organized as follows: Section 5.1 details the proposed technique for detecting anomalies. Section 5.2 presents an evaluation of the proposed technique. Section 5.3 describes the outcomes of the analysis and discusses the obtained results while Section 5.4 summarizes the contributions.

5.1 Proposed Technique

The proposed technique is based on the concept of data density introduced in [12], which uses a non-parametric Cauchy function [84] that can be updated recursively. Due to being non-parametric, only a very small amount of data—only the mean of all data samples μ_k and the scalar product quantity Σ_k calculated at the current moment in time k—needs to be stored in memory and updated [9]. This has significant implications in a cloud-operating context since it allows theoretically an infinite amount of data (infinitely large monitoring metrics) to be processed in near real time without having to store the historical data itself.

Let all measurable physical variables form the vector $x \in R^n$ be divided into several clusters. Then, the local density d_Λ of cluster Λ, based on Euclidean distance, is defined as:

$$d_\Lambda = \frac{1}{1 + \frac{1}{N_\Lambda} \sum_{i=1}^{N_\Lambda} \|x_k - x_{f_i^\Lambda}\|^2} \tag{5.1}$$

where N_Λ denotes the number of data samples associated with cluster Λ. f_i^Λ transforms i (identifying a vector contributing to the cluster) into the domain of k (identifying a vector from the complete set). In the case of anomaly detection, x_k represents the feature vector with values for the instant k. It can be shown, that this formula can be derived as an exact quantity [10]:

$$D_k = \frac{1}{1 + \|x_k - \mu_k\|^2 + \Sigma_k - \|\mu_k\|^2} \tag{5.2}$$

where both, the mean, μ_k and the scalar product, Σ_k can be updated recursively as follows:

$$\mu_k = \frac{k-1}{k}\mu_{k-1} + \frac{1}{k}x_k, \mu_1 = x_1 \tag{5.3}$$

$$\Sigma_k = \frac{k-1}{k}\Sigma_{k-1} + \frac{1}{k}\|x_k\|^2, \Sigma_1 = \|x_1\|^2 \tag{5.4}$$

The data is collected continuously, in on-line mode during the detection process. Some of the new data reinforce and confirm the information contained in the previous data. Other data, however, bring new information, which could indicate a change in operating conditions, development of an anomaly or simply a more significant change in the dynamic of the system [11]. In order to detect anomalous behaviour, the variable ΔD_k is, then, calculated as follows:

$$\Delta D_k = |D_k - D_{k-1}| \tag{5.5}$$

where D_k is the density calculated for the current data sample x_k and D_{k-1} is the density calculated for the immediately previous data sample x_{k-1}. The mean of all densities so far $\bar{D}_k$ can also be calculated, and compared to indicate whether the system is in an anomalous state (indicated when $D_k < \bar{D}_k$) or a normal state. $\bar{D}_k$ is calculated as follows:

$$\bar{D}_k = \left(\frac{k_s - 1}{k_s}\mu_{D_{k-1}} + \frac{1}{k_s}D_k\right)(1 - \Delta D_k) + D_k \Delta D_k \tag{5.6}$$

where k counts the number of data samples which are read, and k_s counts the number of time steps in which the system remains in the same status (normal/anomalous). Since the computation of $D_k < \bar{D}_k$ is based entirely on the concept of data density, it is highly suitable and applicable for real-time anomaly detection in cloud environments. Moreover, the time complexity for calculating the new density D_k and mean density $\bar{D}_k$ from their previous value is classified as $O(n)$.

5.2 Evaluation

The technique is evaluated using network traces obtained from a controlled testbed resembling a cloud environment, featuring VM migration as a normal cloud operation, plus network attacks that should be regarded as anomalies. The main reason for deploying network level attacks is because usually in cloud there are more end users consuming resources and this therefore increases the attack surface, which can be more damaging. The cloud can offer more computational power in the form of virtual resources, to cope with additional work loads. This to some extent supports the attackers by enabling them to perform a loss of availability on the intended service [131].

The proposed technique could not be evaluated without the ability to generate anomalies within a testing environment with absolute ground truth. The testbed allows the traces to be labelled with ground truth, about both the expected anomalies and the presence of a migration. Comparison of the output of a detector with ground truth allows us to determine the performance of the technique as a detector.

5.2.1 Testbed

The cloud testbed consists of two hosts which serve as compute nodes running multiple VMs, each VM running Apache HTTPd. The client host acts as a controller to initiate migrations, and also to generate background traffic. The 'challenger' host runs custom attack scripts to generate attack traffic directed towards the VMs' address range for a selected attack type and intensity (i.e.,

Algorithm 1 Proposed anomaly detection technique

Let x_1 be the first feature vector;
Let $S = 0$; (normal state)
Let $k = 1$;
Let $D_k = 1.0$;
Let $\bar{D}_k = D_k$;
Let $\mu_k = x_k$;
Let $\Sigma_k = \|x_k\|^2$;
Let $c = 0$;
while more vectors **do**
  Let $k = k + 1$;
  Let x_k be the next feature vector;
  μ_k = update by equation 5.3;
  Σ_k = update by equation 5.4;
  D_k = update by equation 5.2;
  $\Delta D_k = |D_k - D_{k-1}|$;
  if $S = 0$ **then**
    $\bar{D}_k$ = update by equation 5.6;
    Let $k_s = k_s + 1$;
    if $D_k \leq \bar{D}_k Th_1$ **then**
      Let $c = c + 1$;
      if $c \geq Ws_1$ **then**
        Let $S = 1$; (anomalous state)
        Let $k_s = 0$;
      end if
    else
      Let $c = 0$;
    end if
  else
    $\bar{D}_k = \bar{D}_{k-1}$
    Let $k_s = k_s + 1$;
    if $D_k \geq \bar{D}_k Th_2$ **then**
      Let $c = c + 1$;
      if $c \geq Ws_2$ **then**
        Let $S = 0$; (normal state)
        Let $k_s = 0$;
        Let $D_k = 1.0$;
        Let $\bar{D}_k = D_k$;
        Let $\mu_k = x_k$;
        Let $\Sigma_k = \|x_k\|^2$;
      end if
    else
      Let $c = 0$;
    end if
  end if
end while

the volume of traffic it generates). Tcpdump[7] is used to simultaneously collect packet traces from the two virtual bridge interfaces, one in each physical

[7] http://www.tcpdump.org/

node, and so these traces represent aggregated traffic to/from all VMs on a node. All are connected to a LAN, as shown in Fig. 5.1. Each physical node runs KVM[8] as virtualization infrastructure, and QEMU[9] provides hardware emulation. Migration is achieved with *libvirt*[10]. All VMs on a node are connected to a virtual bridge interface `virbr0`, so their own interfaces appear to be part of the LAN. Further details about the architecture of the system, as well as, how the anomaly detection component may operate in the context of a cloud infrastructure can be found in [126].

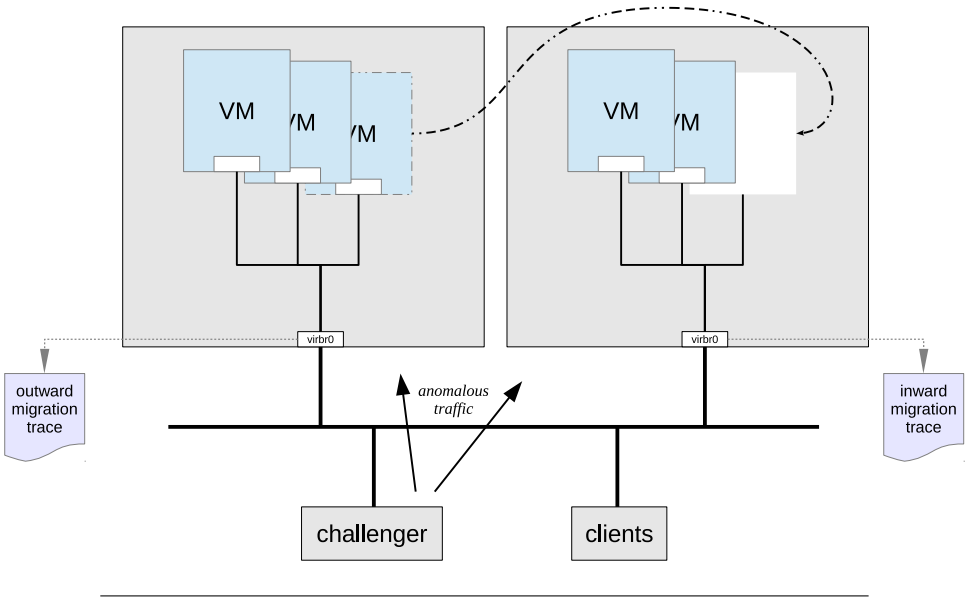


FIGURE 5.1: Experimental setup

This set-up allows experiments to be run in which the legitimate traffic of several web servers is continuously populated, while anomalous traffic is created by injecting attack traffic into the legitimate traffic at some time during the experiment. Independently, one of the VMs running a webserver can be migrated live between the nodes during a period of either normal or anomalous traffic. Traces obtained at the virtual bridges are fed into the detector to observe its reactions to normal/anomalous traffic. Network traces are split into 1-second bins, and then a set of statistical properties (features) of the traffic in each bin is computed; each feature vector is submitted to the detector. The vector includes following features:

- number of packets
- number of bytes
- number of active flows in each bin
- entropy of source IP address distribution

[8] http://www.linux-kvm.org/
[9] http://www.qemu.org/
[10] http://libvirt.org/libvirt2

- entropy of destination IP address distribution
- entropy of source port distribution
- entropy of destination port distribution
- entropy of packet size distribution.

For evaluation, each vector is also labelled with ground truth about the presence of anomalies and migration. Packet traces are captured with `libpcap`[11]. Because the testbed allow to have a control over when anomaly and migration occur, the obtained traces can be confidently labelled with ground truth about both conditions, and therefore assess the performance of proposed detection technique.

5.2.2 Evaluation Metrics

The single metric alone is not sufficient to make a firm conclusion about performance of underlying anomaly detection technique [80]. Therefore, the effectiveness of the proposed technique is evaluated using several metrics. Each input entry submitted to the detector describes the features of monitored network traffic during a given time period (bin), and the detector then computes deviation from normal traffic. Therefore, the performance can be assessed by determining the difference between the class it produces for a given input and the class it should have.

Correctly identified negatives are True Negatives (TN), incorrectly identified negatives are False Positives (FP), correctly identified positives are True Positives (TP) and incorrectly identified positives are False Negatives (FN). From this output it allows computation of the *true-positive rate* (TPR, sensitivity or recall; $TP/(TP+FN)$), the *false-positive rate* (FPR; $FP/(FP+FN)$), the *precision* ($TP/(TP+FP)$), the *accuracy* ($TP+TN/TP+TN+FP+FN$), the *F score* ($2\times$ ($Precision \times Recall$) / ($Precision + Recall$)), and the *G mean* ($\sqrt{Precision \times Recall}$).

Accuracy is the degree to which the detector classifies data samples correctly; precision is a measure of how many of the positive classifications are correct, i.e. the probability that a detected anomaly has been correctly classified; and recall is a measure of the detector's ability to correctly identify an anomaly, i.e. the probability that an anomalous sample will be correctly detected. The final two metrics are the harmonic mean (F score) and geometric mean (G mean), which provide a more rounded measure of the performance of a particular detector by accounting for all of the outcomes to some degree. The Precision and Recall are important for evaluation since there could be a possibility that some of the data instances are few in number relative to others. For example if the *portscan* attack represented 95% of total traffic

[11] https://github.com/the-tcpdump-group/libpcap

in the captured traffic and the normal data instances were 5%. If all the predicted instances were *portscan* then the overall accuracy will be 5% in spite of the lost predicted normal class of data.

5.2.3 Tuning the Algorithm

The background traffic on the testbed consists of random client HTTP requests producing substantially oscillatory behaviour that could confound a detector. To overcome this, first analysed the various experimental runs, and introduced tuning parameters/augmentations with respect to the captured dataset detailed below: It is worth mentioning that tuning does not affect the generic nature of the approach and it can be used for any type of data and application domain.

1. In order to have more defined normality regions, the mean of density $\bar{D}_k$ is recursively updated only for normal data.

2. Transition from one state to another is controlled by two *tolerance thresholds* Th_1 and Th_2 and two windows Ws_1, Ws_2, where;

   - the detector output can only switch from normal to anomalous when $D_k \leq \bar{D}_k Th_1$ for Ws_1 successive bins, and
   - the detector output can only switch from anomalous to normal if $D_k \geq \bar{D}_k Th_2$ for Ws_2 successive bins.

   The windows Ws_1 and Ws_2 are intuitive enter/exit thresholds, which permit a good trade-off between response time and stability of the detector.

3. Once the system is back in a normal state the density of a current sample is reset back to one, i.e., $D_k = 1$, to mitigate the impact of current anomalous data density onto subsequent data density computation. Correspondingly, other recorded values are reset, namely, $\bar{D}_k = D_k$, $\mu_k = x_k$ and $\Sigma_k = \|x_k\|^2$.

The coefficient $(1 - \Delta D_k)$ will lead $\bar{D}_k$ to near the actual mean of density when there is a smooth change in the data, and ΔD_k will lead $\bar{D}_k$ to near the new value of D_k in the presence of an anomaly.

The detection technique with these modifications is shown in Algorithm 1.

5.3 Analysis of Results

To validate the tuning strategy, a collection of traces previously obtained from the testbed is analysed. Each experimental run yields a pair of packet traces that are labelled with the ground truth regarding the presence of attack

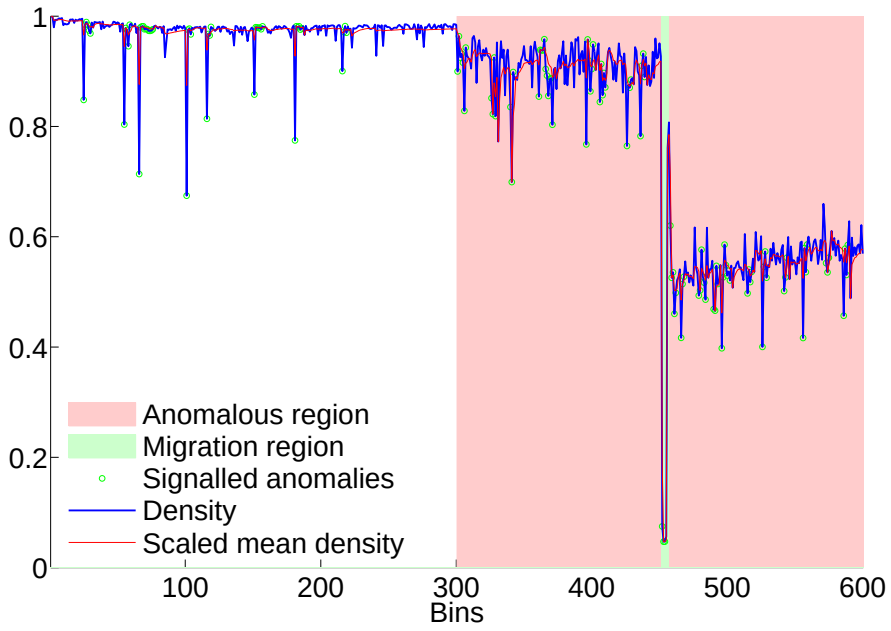


FIGURE 5.2: Before tuning

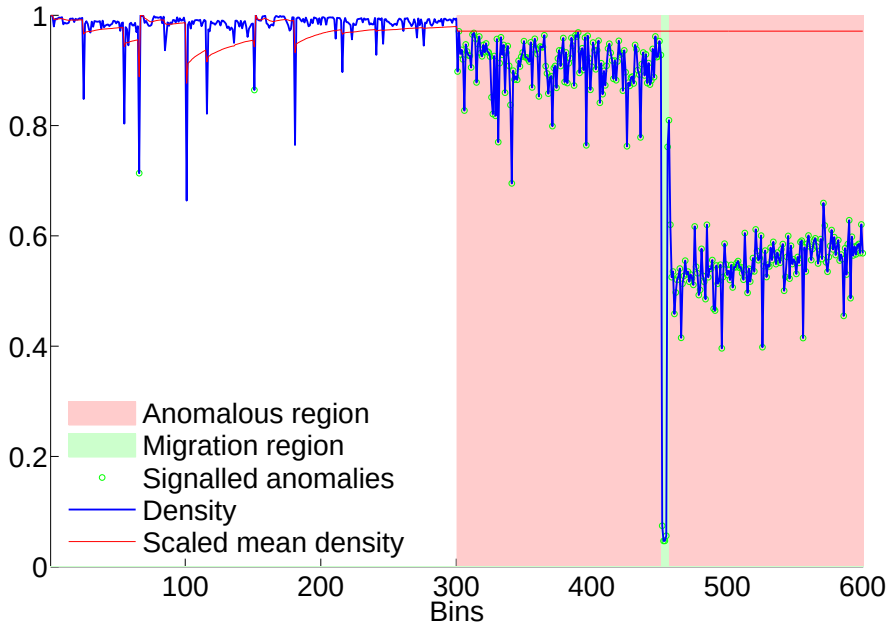


FIGURE 5.3: After tuning

traffic and migration in the trace. In each 10-minute run, background traffic occurs continuously and hence appears throughout the trace. After the first 5 minutes in the first run, an attack script starts, hence its traffic appears in each trace from the midpoint. At either 2.5 minutes or 7.5 minutes, a migration of one of the VMs is initiated. A run can therefore be characterised

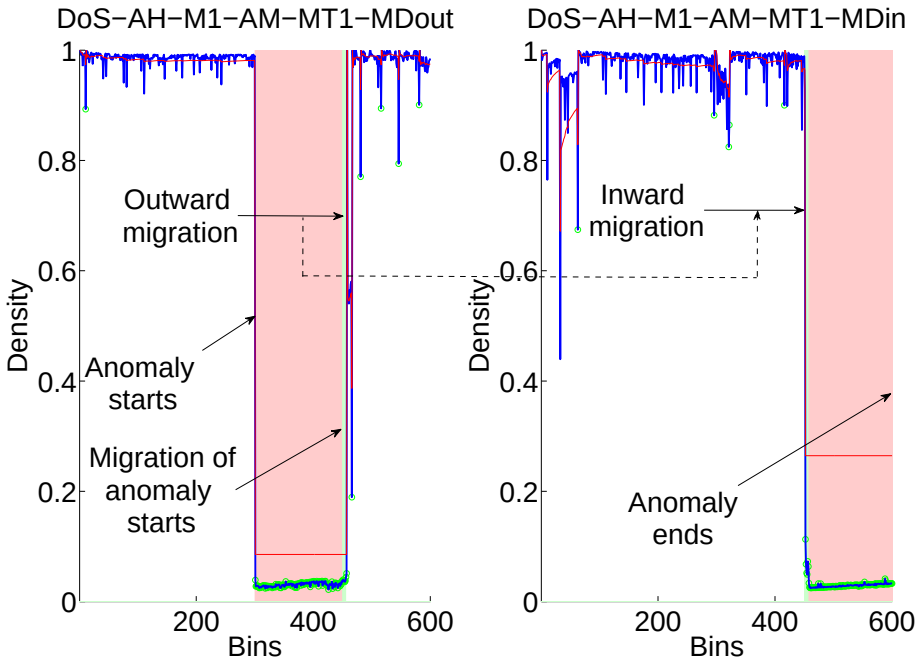


FIGURE 5.4: Migration during anomalous period

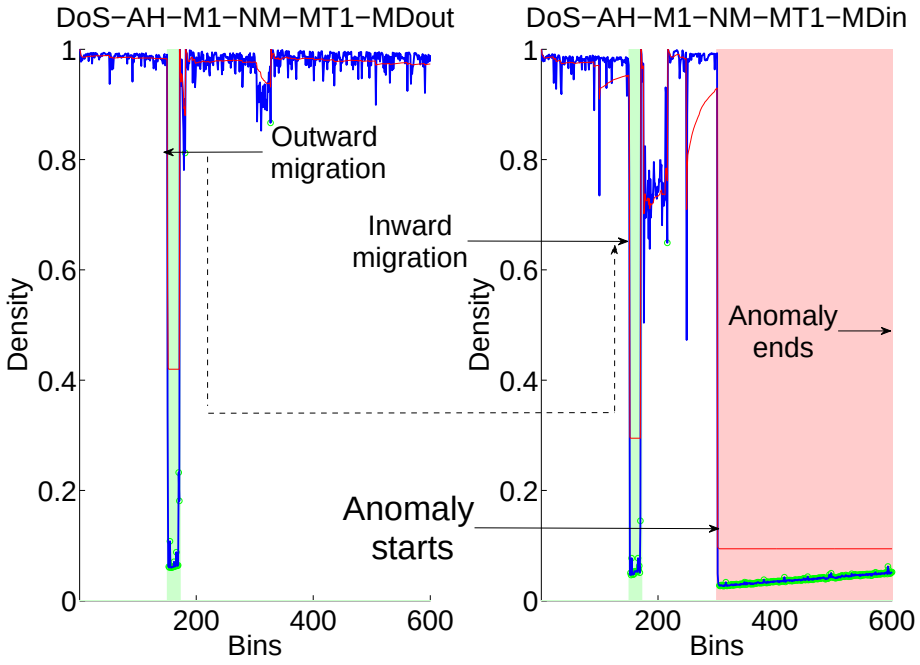


FIGURE 5.5: Migration during normal period

by the attack type (labelled DoS for denial-of-service, NS for netscan and PS for portscan) and intensity AH for high; AL for low, and whether the migration occurs during the attack (AM) or during the normal period (NM) (i.e., 'migration overlap'). Each trace from a run can be further characterised by whether the node it was taken from experienced an outward (MDout) or

TABLE 5.1: Detection results of DoS attack with MT0 under high and low intensity

Scenario	Recall	Precision	Accuracy	F-score	G-mean
High-intensity(AH)					
BC0-DoS-AH-M1-AM-MT0-MDin	0.996667	0.993377	1.0000	0.996678	0.996683
BC0-DoS-AH-M1-AM-MT0-MDout	0.991653	0.993289	0.989967	0.991625	0.991626
BC0-DoS-AH-M1-NM-MT0-MDin	0.923333	0.869186	0.996667	0.928571	0.930746
BC0-DoS-AH-M1-NM-MT0-MDout	0.936561	0.887240	1.00	0.940252	0.941934
Low-intensity(AL)					
BC0-DoS-AL-M1-AM-MT0-MDin	0.978333	0.967427	0.990000	0.978583	0.978648
BC0-DoS-AL-M1-AM-MT0-MDout	0.988333	0.980328	0.996667	0.988430	0.988464
BC0-DoS-AL-M1-NM-MT0-MDin	0.8900	0.825000	0.990000	0.900000	0.903742
BC0-DoS-AL-M1-NM-MT0-MDout	0.948247	0.908537	0.996656	0.950558	0.951577

inward (MDin) migration of the VM. A run is also characterised by whether the attack targets the VM that migrates (MT1) or a VM that does not (MT0). The fixed background traffic involves five VMs running identical HTTP servers. Three VMs on one physical host, and two on the other. A host external to the VM infrastructure runs HTTP clients repeatedly connecting to each VM, two per VM. The improvement in performance is observed when detector is augmented with the additional parameters and behaviour. For example, for a high-intensity DoS attack with migration during the attack DoS-AH-AM-MT0, Figure 5.2 shows the output of density-based detector that simply signals when normalized $\bar{D}_k$ (red) drops below D_k (blue), which oscillates considerably even during the normal period. With $\bar{D}_k$ relatively stable, many normal bins are reported as anomalies, as indicated by the green circles. In comparison, Figure 5.3 shows the same trace fed through the augmented density-based algorithm. The window Ws_1 has caused fewer normal bins to be signalled as anomalies, reducing the false-positive rate. Also, with $\bar{D}_k$ not being updated once an anomaly has been signalled, fewer

anomalous bins are signalled as normal, increasing the true-positive rate.

TABLE 5.2: Detection results of DoS attack with MT1 under high and low intensity

Scenario	Recall	Precision	Accuracy	F-score	G-mean
High-intensity(AH)					
DoS-AH-M1-AM-MT1-MDin	0.991667	0967532	1.0	0.983498	0.983632
DoS-AH-M1-AM-MT1-MDout	0.978333	0.920245	1.0	0.958466	0.959294
DoS-AH-M1-NM-MT1-MDin	0.958333	0.933754	0.986667	0.959481	0.959846
DoS-AH-M1-NM-MT1-MDout	0.96500	0.0	NaN[12]	0.0	NaN
Low-intensity(AL)					
DoS-AL-M1-AM-MT1-MDin	0.981667	0.937107	0.993333	0.964401	0.964811
DoS-AL-M1-AM-MT1-MDout	0.978297	0.925466	0.993333	0.958199	0.958799
DoS-AL-M1-NM-MT1-MDin	0.980000	0.973684	0.986667	0.980132	0.980154
DoS-AL-M1-NM-MT1-MDout	0.984975	0.0	NaN	0.0	NaN

5.3.1 Detection with Migration

Clouds are also characterised by the elasticity that virtualisation enables. As such it is also important to test detection in scenarios that utilise the elastic nature of the cloud. One such elasticity measure is migration, which allows load balancing and fail over. Using a feature set that is capable of encapsulating changes to the volumetric properties of traffic on the network, the Denial-of-Service (DoS) attacks were detected on VMs using the proposed detector with a high degree of accuracy during migration. Figures 5.4 and 5.5 visually represent detection performance under high-intensity DoS attack with MT1 (i.e., the VM experiencing the attack is migrating) where migration happens during anomalous and normal periods respectively. The results show that the anomalous region precisely detected (marked by green circles) with

[12]The missing values are simply a matter of the detector not producing any true positives or false positives, therefore these metrics could not be calculated

TABLE 5.3: Detection results of netscan and portscan attacks under high and low intensity

Scenario	Recall	Precision	Accuracy	F-score	G-mean
Netscan (NS))					
BC0-NS-AH-M1-AM-MDin	0.836667	1.0	0.673333	0.804781	0.820569
BC0-NS-AH-M1-AM-MDout	0.963272	1.0	0.926421	0.961806	0.962508
BC0-NS-AH-M1-NM-MDin	0.828333	0.818770	0.843333	0.830870	0.830961
BC0-NS-AH-M1-NM-MDout	0.855000	0.863481	0.843333	0.853288	0.8553348
BC0-NS-AL-M1-AM-MDin	0.781302	0.7800000	0.782609	0.781302	0.781303
Portscan (PS)					
BC0-PS-AH-M1-AM-MDin	0.960000	0.925926	1.0	0.961538	0.962250
BC0-PS-AH-M1-AM-MDout	0.973333	0.949367	1.0	0.974026	0.974355
BC0-PS-AH-M1-NM-MDin	0.894825	0.837143	0.979933	0.902928	0.905728
BC0-PS-AH-M1-NM-MDout	0.909850	0.847025	1.0	0.917178	0.920340
BC0-PS-AL-M1-AM-MDin	0.863333	0.791444	0.986667	0.878338	0.883681

very few false-positives[13]. The output of the detector was used to produce evaluation metrics according to the formulae in Section 5.2.2.

The results in Table 5.1 and Table 5.2 show that the choice of network features is appropriate and sufficient for detecting network based DoS attacks with high and low intensity. In order to aid the evaluation process, the Table 5.3 presents detection results for netscan (NS) and portscan (PS). The results of detection are promising, with excellent detection observed for network level attacks under various intensities. In addition, this feature set could be expanded to include statistics derived from other resources (such

[13]The overall results are the best measure of the performance of the detector and were calculated by combining the results of both components as if they had been produced by a single detector.

as vCPU usage). There is a trade-off since more computation resources will be required, but this could be beneficial in the detection of other types of anomaly.

5.4 Summary

In summary, and compared to work reported in the literature on cloud anomaly detection, the chapter presents:

1. a novel, memory-less technique for anomaly detection using density, which can be used in cloud environments where metrics are monitored in large volumes and in real time.

2. evaluation of the proposed technique using real data captured from a cloud testbed with respect to various attack types and intensities in the face of migration. Results show that the technique is effective in detection high and low intensity network level attacks with 98% accuracy.

The cloud-specific dataset, labelled with the ground truth of migration and anomalies on which this work is based, is available on request. The implementation of the density-based detector in Python is available online[14].

[14] https://forge.comp.lancs.ac.uk/svn-repos/seccrit-internal/rde-detector/tags/published-2015/

Chapter 6

Anomaly Detection-as-a-Service

The rapidly growing popularity of cloud environments for a number of always-on-services has created many challenges that are not being sufficiently addressed [144]. These challenges affect several domains that aim to improve the overall Quality-of-Service (QoS) and clearly reduce the overall reliability and availability of the cloud, i.e., it is less resilient to challenges (where resilience is the ability of the system to perform its function even in the presence of challenges [132]). This implies that it is of utmost importance to clearly understand and define what constitutes the correct, normal behaviour so that deviation from that can be detected as anomalies and consequently a more resilience in cloud system can be engineered. In addition, cloud properties can also exacerbate these challenges in both their detection and impact, and so increase management costs, and lead to SLA violations [146, 23]. As a result, detection becomes of paramount importance, both to the tenants as well as the cloud providers, who aim to ensure continuous system operation. A recent study by security services provider Alert Logic says that application attacks aimed at cloud deployments grew 45% over one year [82]. Their research is based on an analysis of one billion events, during the whole of 2014, and across more than 3000 of its customers. These challenges evidently reduce the overall reliability of the cloud computing. There are several significant challenges in securing cloud infrastructures from different types of attacks. For instance, on 29th June 2010, `amazon.com` experienced hours of anomalous behaviour whereby customers were unable to place the orders. Based on their 2010 quarterly revenues, such downtime cost Amazon up to $1.75 million per hour.

Anomaly Detection (AD) is one of the key techniques used for securing infrastructures against potential threats in cloud environments, despite these techniques often generating a large number of false alarms [97]. The dynamic nature of clouds makes AD especially challenging, as multiple applications with diverse and evolving workload patterns impact detection ability [1]. Furthermore, the huge cost of downtime of services as a result of challenges

drives the need for AD mechanisms that can offer online and real time detection in order to further support remediation mechanisms. Therefore, the near real time detection of anomalies with better false-positive rate is a vital element of operations in clouds in providing the operators with a more timely detection of potential security issues within the infrastructure. However, as the scale and complexity of modern cloud system increase, the underlying anomaly detection mechanisms require significant higher levels of automation i.e., continuous monitoring of resources and detection based on real time streams of system events. But continuous monitoring in cloud lead to profuse volume of data which may cause low detection accuracy by obscuring the detection of anomalies and high computational complexity [125]. Therefore, a technique which allows the building, accumulation, and self-learning of a dynamically evolving information model of "normality", and which can operate in real time, is of great value.

Chapter 3 present experimental results where the investigation is made about how elastic behaviour in cloud affects state-of-the-art anomaly detection techniques, potentially making them unreliable for services running in cloud environments. The presented results suggested that in some configurations, challenge anomalies are missed and some configuration anomalies are wrongly classified. Moreover, it is noticed that the computational complexity of these techniques is high because they require complex statistical computations, as well as storage of the complete historical metrics. Section 6.2.5 illustrates the computational complexity analysis. Based on these results, the argument is build that there is a need for robust, preferably real time, anomaly detection technique embodied in a flexible and scalable service model that a cloud provider can offer to its tenants and their customers.

There exist many techniques for anomaly detection in several disciplines and they have exhibited sufficient detection accuracy in a variety of scenarios, as indicated in the comprehensive survey by [33]. In the context of cloud computing, a number of anomaly detection techniques have been adopted and redefined [143, 142, 55, 56, 14]. However, these approaches use complex statistical measures, lack scalability, and often require prior knowledge about anomalous or normal behaviour, making them potentially unusable in a cloud operational context due to high dynamic workload and large volume of metrics in these environments. In contrast, the 'Anomaly Detection as a Service' ($ADaaS$) model, features density-based technique that is lightweight and aims to address the scalability needs of cloud infrastructure. It can easily scale as volume of metrics grows because it does not require storage of past metrics. In addition, there is no need for prior knowledge (supervised) about anomalous/normal behaviour, so it operates unsupervised. As a result, it can detect anomalies that are not well understood (i.e., there are no prior models) or have not been experienced previously.

ADaaS is a multi-tenant, highly scalable, anomaly detection-as-a-service solution based on a monitoring API called *Monasca*[1](more details will follow in subsequent Section 6.1). Providing *ADaaS* to the cloud administrator and users brings a number of benefits. First, *ADaaS* minimise the cost of ownership by exploiting a state-of-the-art detection tool. *ADaaS* makes it easier for users to deploy AD at different cloud abstraction levels, in contrast with ad-hoc monitoring tools or the setting up of dedicated monitoring hardware/-software. *ADaaS* also enables a pay-as-you-go model for detection, e.g., a Critical Infrastructure (CI) provider may want to have full-featured detection services based on their stringent security and resilience requirements as opposed to a non-CI provider offering simple services. Finally, *ADaaS* enables cloud providers to deliver continuous improvement on detection services. They can develop value-added services for secure cloud environments and create new revenue. This value addition can be achieved by clearly understanding the security and resilience requirements of a tenant and enforcing those using deployment function provided by ADaaS. Further, comparing the service performance and infrastructure management logs before and after the instantiation of ADaaS would enable cloud providers to reason the value-add in terms of how many challenges are detected and reported in timely manner. It is envisaged that the *ADaaS* paradigm will become an important factor for on-demand computing in future cloud environments, and would thus help cloud providers in their offerings of secure and resilient infrastructure.

Technical Challenges

Accomplishing anomaly detection as a service faces various technical challenges relating to the collection and delivery of data in a cloud environment. Real time data must be collected from a huge number of virtual instances on multiple physical hosts, and the demand for monitoring must adapt to the dynamic nature of service invocation and resource provision associated with virtualization. The detection as a service also require the support of advanced techniques to achieve high accuracy and reliable distributed monitoring. For example, some task such as network traffic monitoring incur high data collection cost in terms of storage and manipulations. Therefore, achieving accurate yet efficient monitoring of these metrics for detection service is difficult. Moreover, the service model must also support concurrent use by multiple tenants.

[1] http://monasca.io/root/about/

Problem Statement and Technical Contribution

As discussed earlier, due to the scale and complexity of cloud, detection based on continuous real time infrastructure monitoring becomes challenging. Because the monitoring leads to overwhelming volume of data which affects detector's ability in analysing the data. The increasing metrics in sheer volume, may cause low detection accuracy due to the fact that existing anomaly detection techniques do not tackle high metric dimensionality explicitly [68]. In this work, the comprehensive evaluation of anomaly detection techniques in cloud infrastructures is comprehended and a need for a real time detection for over all resilience against challenges. A critical analysis of existing techniques is presented along with a description of the need for an enhanced real time and memory-less detector. An initial comparison concludes that, whilst a large number of detection techniques have been proposed, none of them are suited to work with cloud operational context [127, 125]. They either require an extensive prior knowledge (training, signatures) from human experts for a system to be run effectively i.e, provide high level of accuracy and low false rate. Therefore, the overall objective of the research has been to establish a light-weight real time anomaly detection technique which is more suited to a cloud-operational context by keeping low false rate without the need of prior knowledge and enable the administrator to respond to threats effectively. This has culminated in the formulation of a density-based approach on which an unsupervised technique for detection of anomalies in cloud is founded. From this formulation, a novel *ADaaS* architecture has been modelled as the means of implementation of the detection technique. The chapter describes the design and features of the proposed design, focusing on the key components forming the underlying architecture, the technique and the way the metrics are processed and applied for overall resilience. The main concepts of the novel architecture are validated through the implementation of a prototype system for OpenStack[2]. A series of experiments were conducted to assess the effectiveness of *ADaaS*. This aimed to prove the viability of implementing the system in an operational context.

The remainder of this chapter is organised as follows: Section 6.1 details the *ADaaS* architecture and Section 6.2 describes implementation, highlighting data flow model and its components. Section 6.3 presents an experimental evaluation and describes the outcomes of the analysis and discusses the obtained results. Section 6.4 concludes the paper.

[2] https://www.openstack.org/

6.1 ADaaS Architecture

ADaaS is a multi-tenant, highly scalable, anomaly detection-as-a-service solution based on a monitoring API called *Monasca*[3]. The Monasca API features a REST API through which *ADaaS* can interact with the cloud infrastructures in order to query it or send metrics for processing to anomaly detector (C in the Figure 6.1). The current implementation is open source and based on OpenStack. The *ADaaS* monitors both the OpenStack infrastructure as well as the virtual machines which run on it (A in the Figure 6.1). The *Monasca* API gives flexibility to *ADaaS* to collect raw data at scale which is then fed into detector for real time detection of anomalies. The authentication of all requests are done via OpenStack *Keystone*[4] service and all data is associated with an OpenStack tenant to support multi-tenancy. In addition, *ADaaS* offers flexibility to deploy the tenant instances as per their security and resilience requirements using Deployment Function (DF) (B in the Figure 6.1). In particular, it generates the service descriptions which can be processed by the cloud orchestration framework such as *Nova*, to actually provision the resources for the service. For the example of OpenStack, a *Heat* template can be generated that contains all steps required to automate the *ADaaS* process. The high level architecture of *ADaaS* is shown in Figure 6.1.

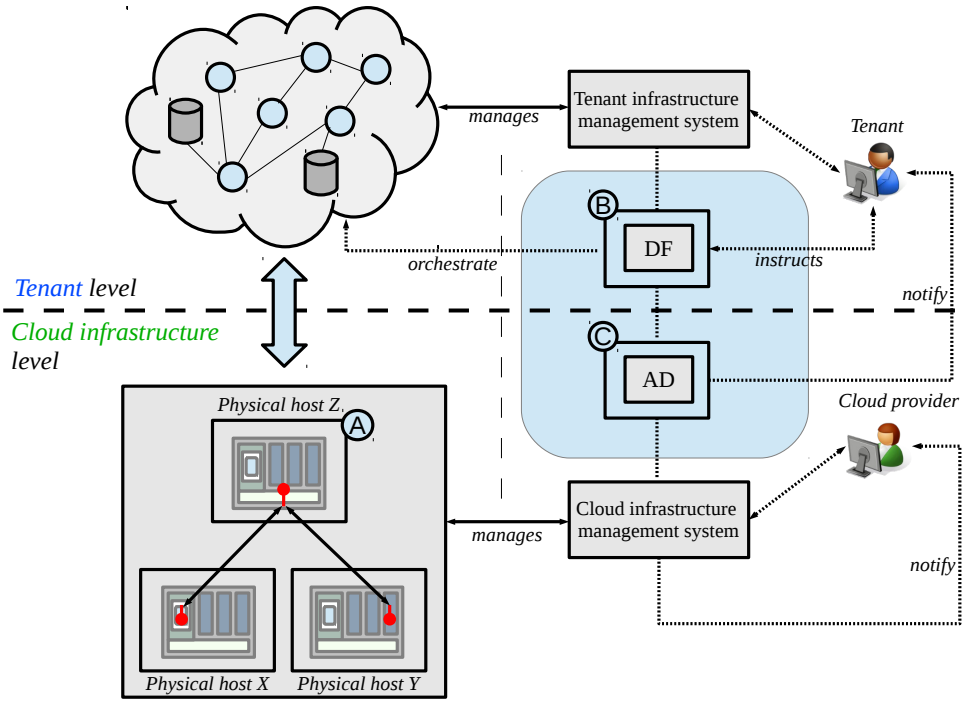


FIGURE 6.1: High level architecture

[3] https://wiki.openstack.org/wiki/Monasca
[4] http://developer.openstack.org/api-ref-identity-v3.html

6.1.1 Component Description

Deployment Functions (DF) is concerned with supplying the configurations describing the creation and deployment of virtual machines for instantiating a service. It ensures that virtual instances are created and destroyed according to service requirements. For example, If tenant request resilience against network level attacks, then DF also ensures that the the necessary components of *ADaaS* (agents with respective plug-ins, detectors, etc.) are correctly instantiated with the virtual instances. Alternatively, it can also provide information to the virtual or cloud infrastructure management in terms of types of instances started and their respective locations, which may allow detection and remediation mechanisms to work in collaboration. Details of the deployment function are presented in Section 6.2.2 below.

***ADaaS* Docker** is a main core engine which integrates feature extraction scripts and anomaly detection engine implementing a density based technique with *Monasca*'s core components, which are extended to work in conjunction for real time detection of anomalies. These components include: Message bus (Kafka[5]), metrics, events and alarms database (Influxdb), configuration database (MySQL), threshold engine and notifier. These components support the *ADaaS* for the management of alarm definition, notification and management of metrics.

6.2 Implementation

The container approach is used for integrating a detection technique with *Monasca* stack which together form the *ADaaS* model. *Monasca* core stacks consists of a large number of interfacing components, each with different configuration, installation and operating processes, some often interfering with existing OpenStack services when being installed on a compute node. The Docker container[6] approach allow us to encapsulate all of the *Monasca*'s components preventing interferences, control the complex installation process with a consistent environment, and only expose the interfaces required for *ADaaS* delivery model. The general architecture is designed to provide stream processing of metrics which are sent via message queue (Kafka) which acts as a general cloud bus through the publish-subscribe principle.

6.2.1 Data Flow

The agent sends the metrics using plugins which provide feature extractions based on their implementations. The metrics are received by the *ADaaS* and published to a message queue. The transformation engine then calculates the

[5] http://kafka.apache.org/
[6] https://www.docker.com

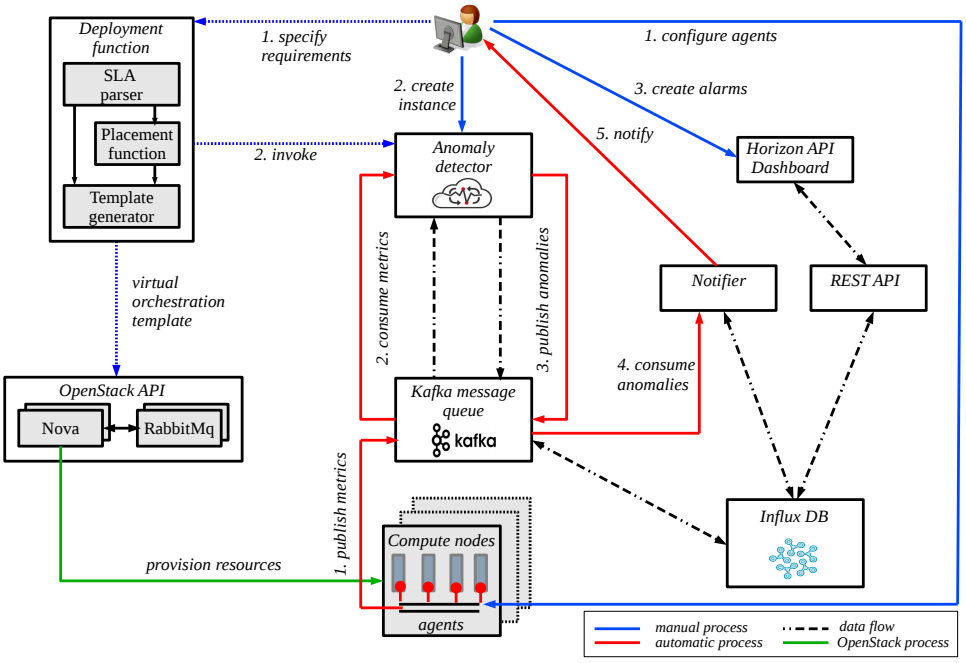


FIGURE 6.2: *ADaaS* data flow.

time based derivatives, and creates metrics that are published back to the message queue. The anomaly engine then consumes metrics from the message queue, predict likelihood of anomalies using data density computation and evaluates anomaly score. The results are published back to the queue as metrics. Alarm definitions can be associated with output of the detector i.e, anomaly score and data density metrics. For example, alarm definition can be created to alarm on the anomaly score or data density rather than a specific value of a metrics. The anomaly engine implements proposed density based technique which is a non parametric and unsupervised technique to identify anomaly from this stream data. Further details to follow in Section 6.2.4.

ADaaS leverages threshold engine provided by *Monasca* which consume metrics and computes moving window thresholds. Alarm state transition events are then published back to message queue. Alarms are automatically created by the threshold engine based on pattern matching of incoming merits to the alarm definitions. This is particular important feature for *ADaaS* to allow new alarm creation without the intervention of an operator. The notification engine then consumes alarm state transition message and sends notification, such as emails when alarm state transition occurs that match alarm definitions. Figure 6.2 illustrates the *ADaaS* data flow.

6.2.2 Deployment Function

The deployment function concerns itself with supplying the Virtual or Cloud Infrastructure Manager with configurations describing the creation and deployment of virtual machines for the cloud infrastructure. Its task is to translate high-level service descriptions and SLAs into automatically deployable descriptions, such as Heat templates in the OpenStack environment. It is therefore a part of a Tenant Infrastructure Management System, being located between the cloud user and the Cloud Infrastructure Operator. In particular, a deployment function is needed that places instances of virtual resources according to the resilience and security requirements of the service user. For example, the placement of virtual machines in dedicated security zone and invocation of *ADaaS* instances using *Heat* templates in the OpenStack environment. The DF could also provide information in terms of types of instances started and their respective locations, which may allow to define remediation strategies based on standard traffic and behavioural patterns of and between these instances. Below section details the sub-components of a deployment function.

SLA Parser

This sub-component is responsible for parsing a user description of their service, i.e., the needed service components and their relation/connectivity, as well as the required performance and availability.

Placement Function

The requirements extracted by the SLA parser translate into a set of virtual machines that need to be deployed in the physical infrastructure. The placement function maps these instances to the offered infrastructure of the cloud infrastructure provider.

Deployment Template Generator

Finally, the mapping must be provided in a form that is processable by the cloud management system. This means generating a deployment, e.g., a Heat template, using as input the placement and the additional service description parts generated by the SLA parser.

6.2.3 Feature Extraction

The first stage of online detection is data collection, which within the *ADaaS* is achieved through the use of agents and custom plugin. For instance at the network level the plugin gathers traffic and generate a time series of features by converting the traffic trace into various normalised statistical properties

on a per packet basis over 3 second bins. These feature includes both volume-based features ("number of ...") and distribution-based features ("entropy of ..."), and are chosen to capture the dynamics of varying anomaly types. Herein, some exemplar features are listed which are obtained using network and database plugins. The network plugin extract features for each bin which include: number of packets; number of bytes; number of active flows in each bin; entropy of source IP address distribution; entropy of destination IP address distribution; entropy of source port distribution; entropy of destination port distribution and entropy of packet size distribution. Similarly for database plugins the features include: number of reads; bytes read; IO stall read; number of writes; bytes written; IO stall write; IO stall and bytes on disk. Appendix B list various metrics which are collected to build evaluation scenarios for *ADaaS*. It is worth mentioning that *Monasca* provides various plugins to monitor service and API such as (memory, disk, and CPU utilisation) which can aid *ADaaS* via agents towards feature extraction and selection process, and can be used as an input to anomaly detector.

6.2.4 Density-based Technique

For online detection the *ADaaS* employs density based technique described in Chapter 5. The technique concerned with identifying anomalous behaviour in real time. The data being collected by agents as per their implementation is processed and converted into feature vectors x_k. x_k represents the feature vector with values for the instant k. Data density values are then computed based using Equation 5.2. Based on the value of data density, the variable ΔD_k is, then, calculated for the current data sample and the immediate previous data sample from the feature vector. Some of the new data reinforce and confirm the information contained in the previous data. Other data, however, bring new information, which could indicate a change in operating conditions, development of an anomaly or simply a more significant change in the dynamic of the system [11]. The mean of all densities so far $\bar{D}_k$ can also be calculated, and compared to indicate whether the system is in an anomalous state (indicated when $D_k < \bar{D}_k$) or a normal state using Equation 5.6. More details on the detection technique is available in the previous work [125].

6.2.5 Computational Complexity

First step towards integrating detection technique for proposed model of a services is to recognise that detection approaches vary widely in terms of computational complexity and implementation. The expert evaluation[7] is carried out of various core algorithms, including statistical and machine

[7]The expert evaluation include the observations made during reference implementation of these core algorithms for previous work [127, 27] on evaluation and also literature survey of articles describing computational complexity of these techniques.

learning methods which are being used in these detection approaches. The computational complexities of these techniques range from linear to exponential, with difference in their computation styles as well. For instance, some of these approaches can be used in distributed or centralised fashion to minimise the overhead. For comparison Table 6.1 list some of the approaches, core algorithms and their computational complexity:

TABLE 6.1: Computational complexity analysis of anomaly detection algorithms

Core Algorithms	**Framework**	**Computational Complexity**	**Reference**
K-Clustering	Signatures	$\mathcal{O}(N^{-nk+1}) \log N$	[65]
Incremental Clustering	Magpie	$\mathcal{O}(n(N+\Delta))$	[28, 17]
Naïve Bayesian	Signatures, Pranaali	$\mathcal{O}(n^2 N)$	[37, 45]
Convolution Algorithm	Blackbox	$\mathcal{O}(em + eS \log S)$	[3]
Traversing	Max	$\mathcal{O}(nN)$	[139]
Sorting	Top-K	$\mathcal{O}(nN \log N)$	[139]
PCA	-	$\mathcal{O}K(MN)^3$	[42]
Density-based	*ADaaS*	$\mathcal{O}(N)$	[125]

6.3 Experimental Evaluation

The detector implementation is evaluated against various traces obtained from a controlled testbed resembling a multi node[8] cloud environment. The testbed allows traces to be labelled with ground truth, about the presence of anomalies. Comparison of the output of a detector with ground truth allows us to determine the performance of the technique as a detector.

6.3.1 Testbed

The cloud testbed consists of three physical hosts which serve as a controller and two compute nodes, running the latest version of OpenStack (Mitaka) which is widely deployed cloud software in the market [118]. The compute node consist of Dell PowerEdge R420 server end machines, while controller runs on Dell OptiPlex 990 workstation. These nodes communicate with each other via dedicated management network. The controller node is responsible to centrally manage other nodes by scheduling them several tasks, and

[8] http://docs.openstack.org/mitaka/install-guide-ubuntu/overview.html

also provides interface to external users via dashboard (*Horizon*). The compute nodes hosts multiple VMs and serves users or tenants with virtual resources. The controller node is installed with several services which include *Keystone* for authentication, *AMQP* for handling inter-process messaging, *Horizon* which provides web based dashboard, *Glance* for managing image services and *Neutron* for managing networking services. Both compute nodes are installed with *Nova* which handles the provision of compute resources to VM, and *Neutron* agents which handle the provision of networking resources to VM. Moreover, both compute nodes use *QEMU* hypervisor to run and host multiple VMs and all of the VMs are bridged together as shown in Figure. 6.3.

Several VMs are hosted in order to simulate real cloud operation and also to generate background traffic during experiments. Each of the VMs is assigned with a specific role which simulates services on the cloud. The services include a Web server (*Apache HTTPd*), File server (*vsftpd*), Mail server (*Postfix*), Hadoop master and two Hadoop slaves[9]. All of the VMs are running Ubuntu 14.04 LTS 64bit and are allocated with 2 virtual CPU, 2 GB RAM, 40 GB disk and 386MB swap space. In addition, the 'challenger' host runs custom scripts to generate attack traffic for a selected attack type and intensity (i.e., the volume of traffic it generates), and client VM generates background traffic using several tools and custom script which continuously request services for certain duration. Figure 6.4 below shows the logical view of experimental setup.

6.3.2 Performance Metrics

Each input entry submitted to the detector implementation in *ADaaS* describes the features of monitored traffic during a given time period (bin), and the detector then computes deviation from normal traffic. Therefore, the performance can be assessed by determining the difference between the class it produces for a given input and the class it should have.The various statistical measures are defined to evaluate the efficacy of anomaly detection technique, which are detailed in Chapter 5 in subsection 6.3.2.

6.3.3 System Metrics Variability Analysis

As stated earlier, the concept of change in data density is used across various metrics (system and network) to identify anomalies. This change in density is independent of normal workload intensity and normal VM configuration changes. Hence, the first set of experiments study is to test the hypothesis, i.e., How can it be ensured that an anomaly detector does not flag a normal

[9]Hadoop cluster is running version 2.6.0 YARN enabled. Hadoop master runs a Python script which continuously put a load to Hadoop Distributed File System (HDFS) for certain duration.

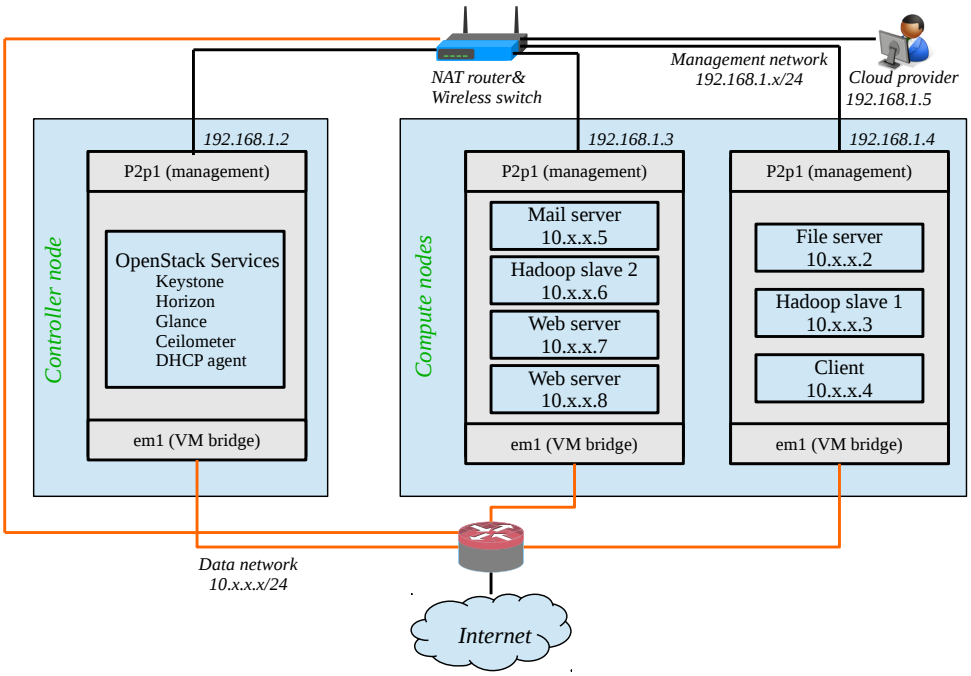


FIGURE 6.3: Topology overview.

event as an anomaly in a complex, dynamic system, such as a cloud?. The correlation is used across various system metrics to statistically define a measure for stability, under the assumption that correlation values remain stable with change in workload and VM configuration parameters. Therefore, the first run of experiment, the configuration of VMs is varied running the workloads by changing their CPU and memory allocations. Basically, changed the CPU and RAM allocations for VM1-VM6 (as shown in Figure. 6.4) from default values (2 virtual CPU, 2GB) to different resource configurations (25%, 50%, 75% and 100%). It is observed that the correlation values remain stable with changes in VM configuration as shown in Figure. 6.5. The results report the average values of CPU and memory utilisation across the VMs and establishes the stability of correlation values with changes in resource configurations.

Similarly, next analyse the variability in workload intensity by measuring pairwise correlation of Hadoop cluster. Figure 6.6 shows the variation in the average pair wise correlation across system metrics (CPU, memory and disk) with changes in workload for Hadoop cluster. The workload intensity refer to running the HDFS write jobs with different file size of 2GB, 5GB, 7GB and 10GB representing 25%, 50%, 75% and 100% workload respectively. It is observed that normal workload show very little variation as represented by pairwise correlation across system metrics. Hence, it can be concluded that normal workload changes will not account for anomalies because anomalies will trigger more deviation in system metrics. Therefore, feature vectors

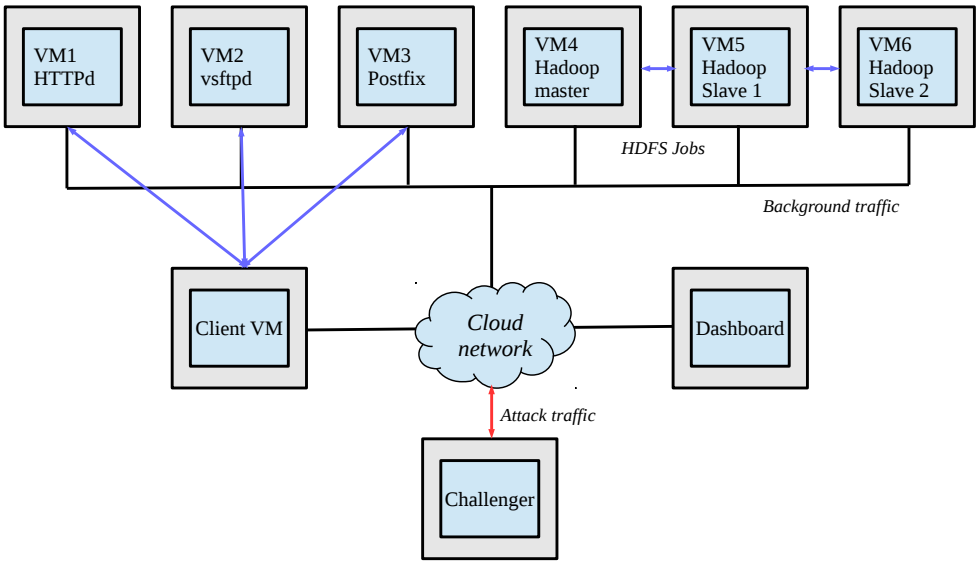


FIGURE 6.4: Logical overview.

obtained using these metrics will be representative of those variations and would influence the computation of data density by indicating a drop in density values, hence, accounting towards anomalous behaviour.

6.3.4 Anomaly Scenarios

To demonstrate the effectiveness of detector, the traces are obtained from the testbed under various anomalous scenarios. These scenarios are categorised into three different types as per their operations as shown in Table. 6.2. The

TABLE 6.2: Description of scenarios

Scenarios	**Type**	**Target Node**
VM Resizing	Management	Compute
API Exhaustion	Management	Controller
Resource Hog (CPU)	Application	Compute
Resource Hog (Memory)	Application	Compute & Storage
Denial-of-Service (DoS)	Network	Controller (network)
Netscan (NS)	Network	Compute
Portscan (PS)	Network	Compute

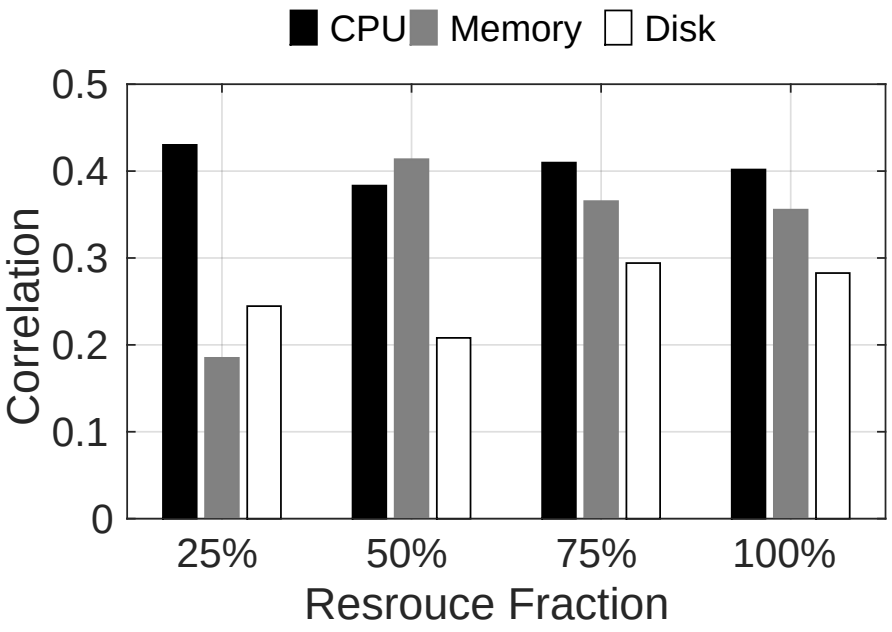


FIGURE 6.5: Configuration variability.

management type include anomalies which are due to operation of hypervisor, VM or the components which provides the management services. The network type involve the internal and external communication network of cloud components. The application type is due to malicious operation of applications ruining in the VM which affected the cloud components.

Malicious VM resizing is the situation where one or more VM attempt to consume more resources then the limit of physical resources available on machine which host the VM. The API exhaustion scenario is realised by overwhelming the authentication API (*Keystone*) of OpenStack with large number of malicious requests using a custom script which exploits *Siege*[10] load testing features to build a scenario. To simulate the performance anomalies in the experiments, the resource hog scenarios are orchestrated by loading *challenger* VM with various custom scripts. The CPU hogging is achieved by performing a floating point computations in an infinite loop, that runs on the target VM. This consumes most of the VM's available processor cycles, leading to poor application performance. Similarly memory hog represents a scenario where there is a memory leak in the application and there is no memory left for application to function, leading to a significant drop in application throughput. The Denial-of-Service attacks are realised using, LOIC[11] which is an open source network stress testing tool, written in C#. It performs DoS attacks on a target VM server by flooding the server with TCP or UDP packets. Intensity of attack was controlled by number of attacking machines and threads involved in generating attack traffic. For network

[10] https://www.joedog.org/siege-home/
[11] https://sourceforge.net/projects/loic/

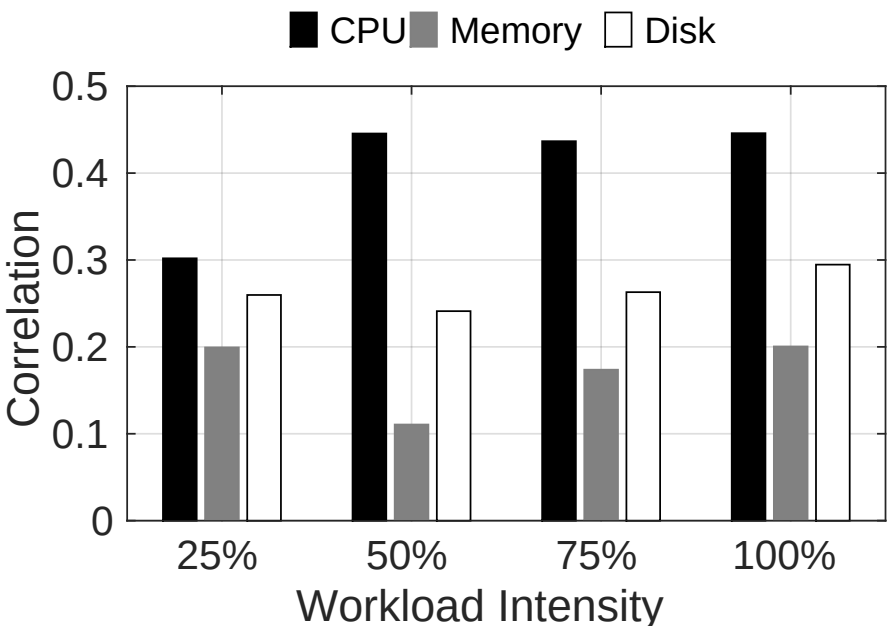


FIGURE 6.6: Workload variability.

and portscan various Python scripts are written which exploit rate limiting features of *Nmap*[12] to control attack intensity and build attack scenarios. These scripts take a configuration file which specifies the timing (such as start and end time of the attack) and networking (such as IP addresses of the VM). These scripts are further re-configurable using different scan, delay and timing modes of the *Nmap* scanner.

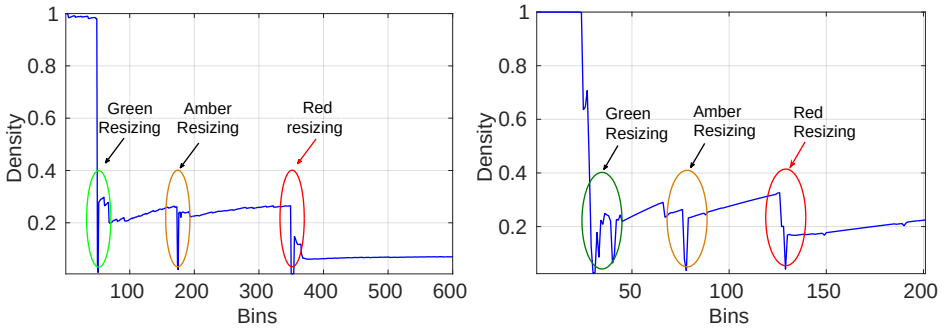
VM Resizing

Three different resizing flavors are defined for VM resizing experiments, which represents different amount of resource allocation. The green, amber and red represents normal, large and malicious resizing respectively. At start of experiments the virtual machines are initiated with a control flavor (1vCPU, 2GB memory and 40GB disk). During the experiments one or more VMs are resized into green (2vCPU, 4GB memory and 60GB disk), amber (3vCPU, 6GB memory and 80GB disk) and red (4vCPU, 10GB memory and 120GB disk). For these scenarios, it is hypothesised that detector should be able to detect resizing while background traffic was running. The VM resizing was achieved using OpenStack dashboard (*Horizon*), under the assumption that attackers have managed to bypass the resizing limit restrictions by exploiting the hypervisor bug or gaining privileged access. During the experiments system and network metrics were collected suing *Sar*[13] and custom scripts respectively.

[12] https://nmap.org/
[13] https://en.wikipedia.org/wiki/Sar_(Unix)

From these experiments the raw data collected is pre-processed and resulting feature vectors are fed into detector. Since, as a result of such management operation the memory commit values of the physical host will increase and also the management network activity would increase. Therefore, three metrics are selected from raw data for pre-processing which include, memory commit, byte count and packet count which is fed into detector for analysis.

Single VM resizing involves resizing of single VM (FTP). At start of experiment (i.e 0th bin), the background traffic scripts are initiated and the data collection begins with binning period of 1sec. The VM is resized to green at 1 minutes. Then at the 3 minute period, the VM was resized again to amber flavor and finally at 6 minute period the, the VM was resized to red flavor. Similarly, for multiple VM resizing experiments two VMs (FTP and Hadoop Slave 2 Server) resized into green, amber and red flavors. The Figure. 6.7a and Figure. 6.7b show the density plot for single and multiple resizing experiments respectively. The density value is dropped when resizing is triggered. The results reveal that background traffic or activity is not being detected as anomalies and therefore it could differentiate between resizing operations. However, the results also indicate that initial resizing process no matter it is green, amber or red, resizing will cause drop in data density values. From detection point of view this is good because cloud provider wants to know of sudden changes in system. However, it seems that in order to differentiate between different types of resizing, the contextual information may be useful which can complement detection process. In this particular case, the *Nova* log can be used to differentiate between malicious and non-malicious resizing. By using the contextual information (e.g., CPU value or VM ID) the detection system can verify the anomalies. This information can be useful to identify the resizing request with the service level agreements, if the VM attempt to claim resources more than allowed threshold the anomaly can be flag as malicious. The drop in density can trigger alarm in real time which can lead to fine grained analysis using this contextual information.



(A) Single VM resizing (B) Multiple VM resizing

FIGURE 6.7: VM resizing results

API Exhaustion

In this scenario, an attack is simulated on *Keystone* API of OpenStack. Keystone is one of the key API and perform authentication and authorisation of various services. During the experiment, 15 hosts are emulated which are sending huge amount of *JSON* requests to API services using the *Siege*. The experiment run for duration of 10 minutes with background traffic. To make sure the detection able to differentiate the anomaly created by an attack from normal authentication operations, two VM operations were performed during this experiments i.e., restart and deletion of a VM. Both operations require the *Nova* API to use the *Keystone* for authentication and authorisation process. At 1 min restart FTP server and and 3 minutes into the run delete Mail server and at 6 minutes the 'challenger' runs script to invoke API exhaustion attack. The system and network metrics are collected form the controller node because *Keystone* components which provides the authentication are installed on controller node. The feature extraction and selection process is similar to previous run of resizing experiment. Both system and network features were fed into detector after normalisation. The results are shown in the Figure. 6.8. The plot shows significant drop in density. The first drop at 20th bin, when VM restart operation is invoked, the density value drop around 30%. The density then start recovery at 21st bin. Then the second operation cause density to drop around 40%. The first and second drop are cause by the same VM restart operation, the first anomaly happen when the VM is turned off and the second anomaly happens when the VM is turned back on again, thus completing the restart process. The third anomaly is at time 60 which is exactly around the time VM deletion operation is invoked. During the deletion process density drops around 38%. This time, the density begin to recover at 64th bin indicating the completion of the deletion process. The fourth drop is detected at time interval 120 when the API exhaustion attack begin. At this time the density value drops significantly 80%. The pattern for such an attack is distinctive and one can apply thresholds on the level of drop to isolate real anomalies. Similar to resizing, the results shows that there is clear distinction between the anomaly of normal VM operation and anomaly of API exhaustion attack. To reduce the alarm, the threshold of 0.5 will automatically flags malicious API request.

Resource Hogging

Generally, the hogging of physical host is due to mis-configuration errors, software bugs or malware activity [124]. Therefore, the aim of this experiment is simulate and investigate the anomalous pattern as a result of hogging of physical resources (i.e., CPU and Memory). Similar to previous run, the system metric data was collected every 3 seconds for duration of 10 minute

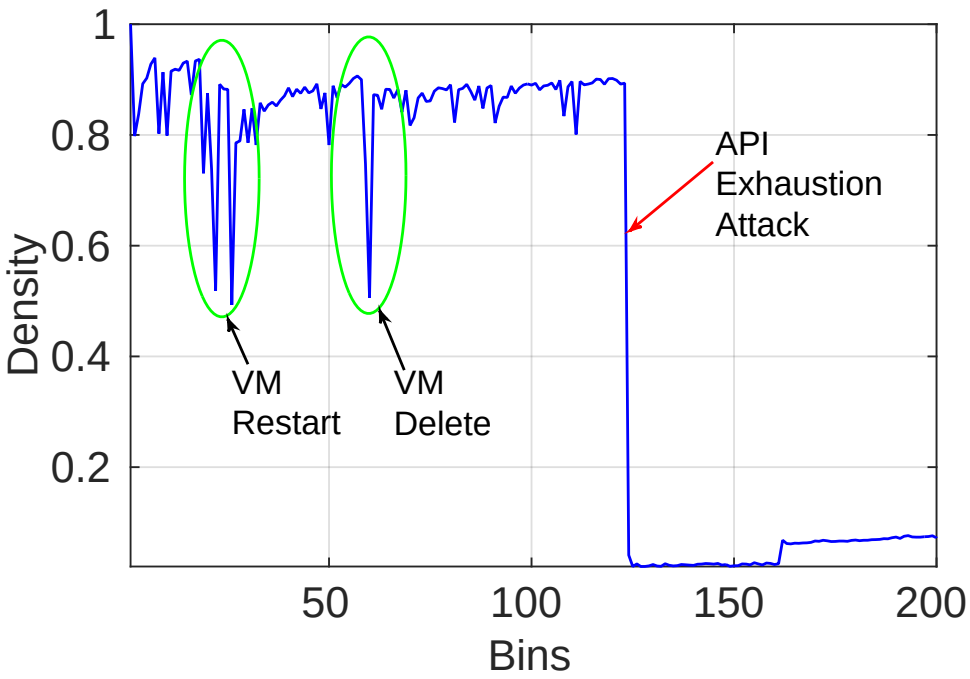


FIGURE 6.8: API Exhaustion

run for each resource hogging scenario. The representative metrics collected in the scenarios are listed in Appendix B. The Figure. 6.9 and Figure. 6.10 visually represents the output of detector (density plot) under CPU and Memory hog respectively. The anomaly start at half way through (i.e 100th bin). There is also drop in density around 21st and 40th bin which is due to background traffic and operations. However, the density drop for actual hogging process is significantly high around (80%), resulting in significantly different anomalous pattern for resource hogging. Similar to previous observations, the results show that anomaly creates more significant drop. Hence, an alarm can be created by applying an arbitrary threshold value of 0.6 on density metric to flag such anomalies.

Network Attacks

To evaluate a detection technique under network attacks, three experimental runs were performed, where each yields a packet trace representing Denial-of-Service (DoS), Netscan (NS) and Portscan (PS) attacks. Each 10 minute run, the background traffic occurs continuously at a fixed rate, and exactly at 5 minute, an attack script starts, hence its traffic appears in each trace from the midpoint. Each packet trace is filtered to eliminate the related management traffic between the VM host nodes. It is subsequently divided in 3 second bins, and each bin is converted into feature vector. Similar to previous run the feature vector is then submitted to detector, yielding density plot against each run. The density plot for DoS, NS and PS is shown in Figures. 6.11, 6.12 and 6.13 respectively and it is evident that the volume based attacks cause

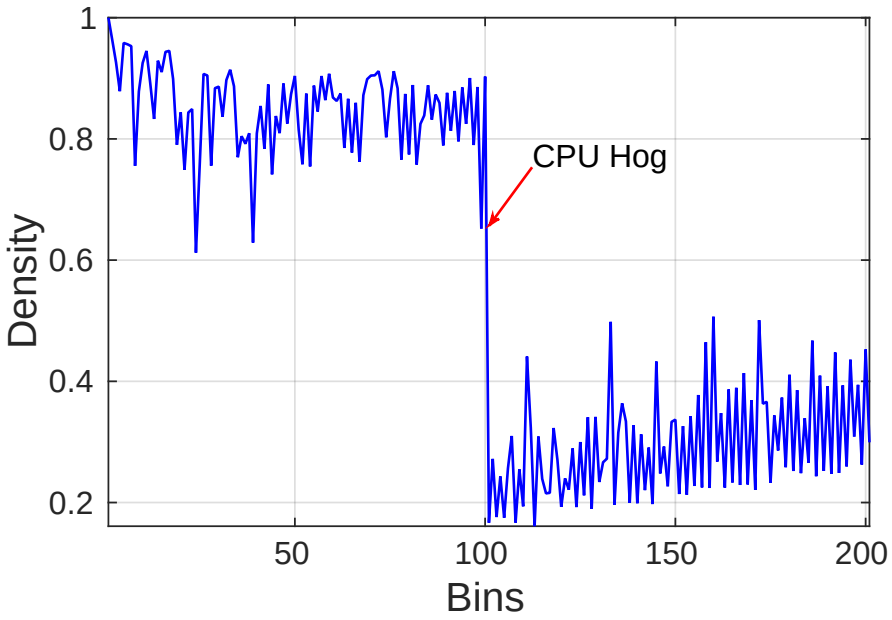


FIGURE 6.9: CPU hog results.

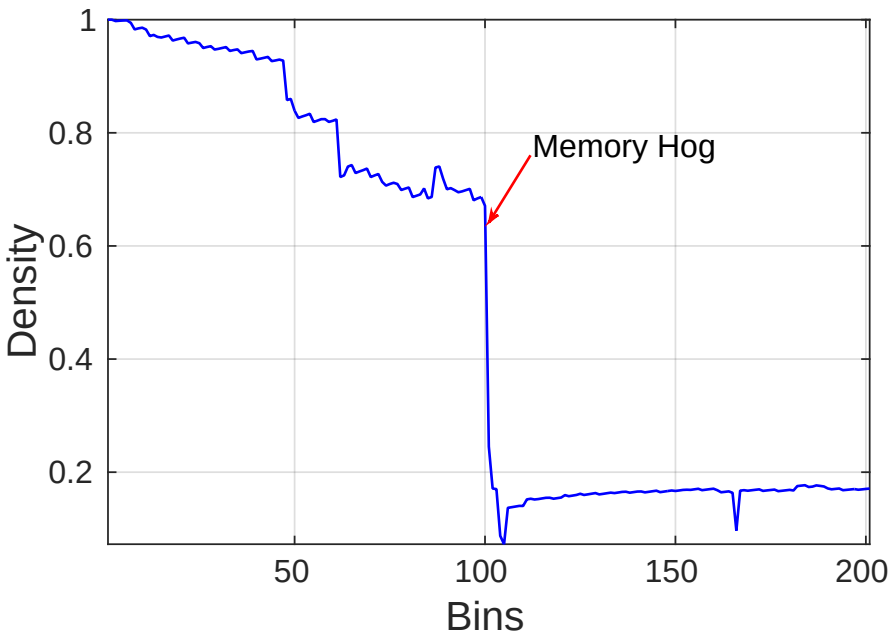


FIGURE 6.10: Memory hog results.

significant drop in density. Therefore, the detector is able to detect such anomalies under high and low intensities with high accuracy and precision as reported in Table. 6.3.

6.3.5 Detection with Migration

Clouds are also characterised by the elasticity that virtualisation enables. As such it is also important to test detection in scenarios that utilise the elastic

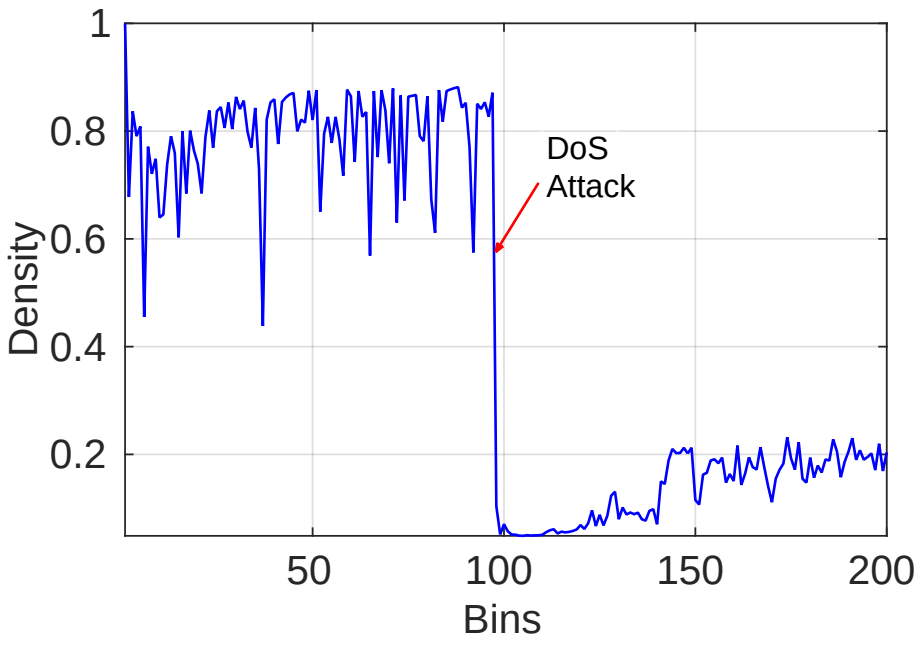


FIGURE 6.11: Denial-of-Service.

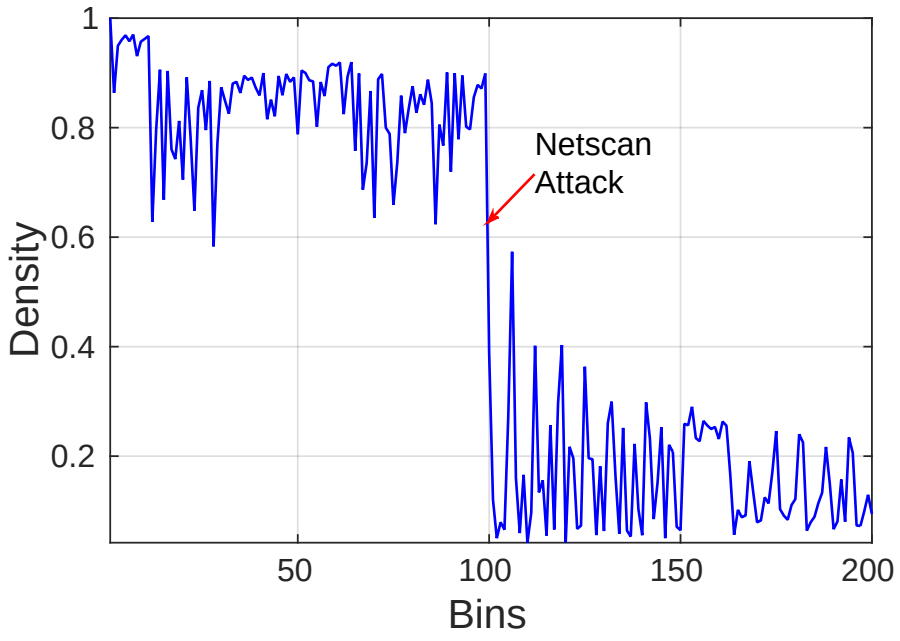


FIGURE 6.12: Netscan.

nature of the cloud. One such elasticity measure is migration, which allows load balancing and fail-over. To validate this, detector is applied to a Denial-of-Service (DoS) traces previously obtained testbed and discussed in earlier work [125]. Using a feature set that is capable of encapsulating changes to the volumetric properties of traffic on the network, DoS attacks were detected on VMs using the detector with a high degree of accuracy during migration.

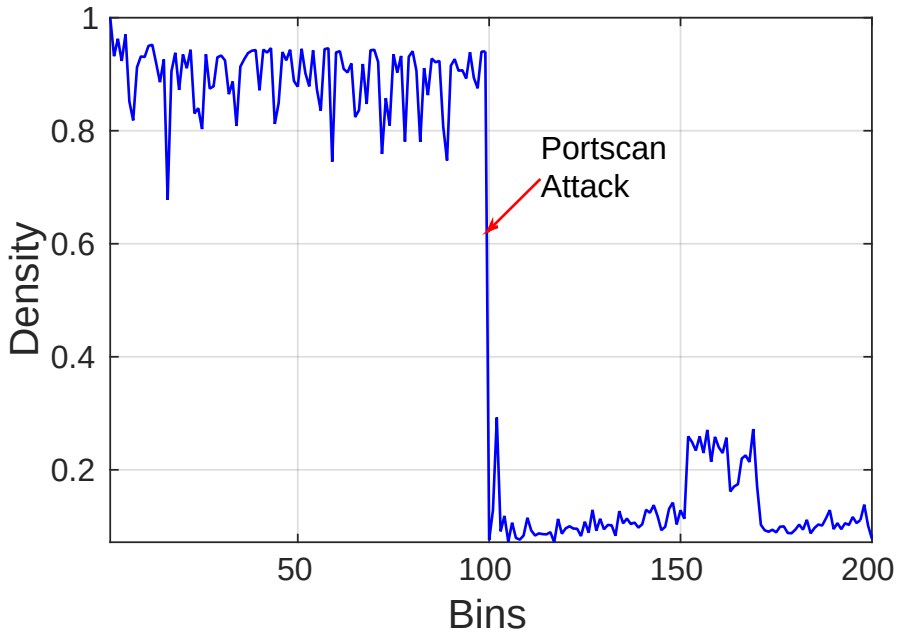
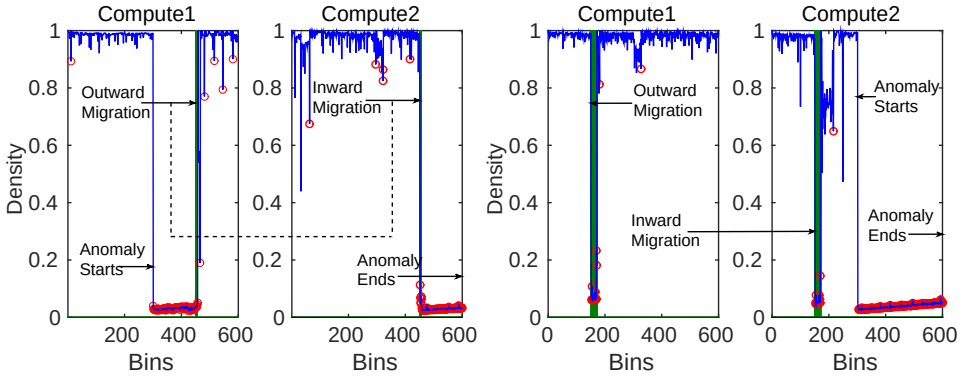


FIGURE 6.13: Portscan.

Figures 6.14a and 6.14b visually represent detection performance under high-intensity DoS attack, when the VM experiencing the attack is migrating, and where migration happens during anomalous and normal periods respectively. The results show that the anomalous region precisely detected (marked by red circles) with very few false-positives[14].



(A) Migration during anomalous period (B) Migration during normal period

FIGURE 6.14: Detection results under migration

The output of the detector was used to produce evaluation metrics according to the formulae in Section 6.3.2. The results in Table 6.3 show that the choice of features (system and network) is appropriate and sufficient for detecting various type of anomalies. The results of detection are promising,

[14]The overall results are the best measure of the performance of the detector and were calculated by combining the results of both components as if they had been produced by a single detector.

with excellent detection observed for different type of anomalies with varying intensities. In addition, the feature set could be expanded to include statistics from *Monasca* agents. There is trade off since more computation resources will be required, but this could be useful in the detection of unseen anomalies. The metrics monitored to build this and all previously described scenarios are listed in Appendix B.

TABLE 6.3: Detection results of anomaly scenarios

Scenario	**Recall**	**Precision**	**Accuracy**	**F-score**	**G-score**
Type: Management					
API Exhaustion	0.955224	0.908046	0.987500	0.946108	0.946940
Type: Application					
CPU Hog	0.751244	0.776596	0.715686	0.744898	0.745519
Memory Hog	0.457711	0.480874	0.862745	0.617544	0.644106
Type: Network **High Intensity (HI) and Low Intensity (LI)**					
Netscan (HI)	0.963272	1.0	0.926421	0.961806	0.962508
Netscan (LI)	0.870000	0.844037	0.910891	0.876190	0.876827
Portscan (HI)	0.835000	0.814815	0.871287	0.842105	0.842578
Portscan (LI)	0.863333	0.791444	0.986667	0.878338	0.883681
Type: Network anomalies under migration **AM: Migration happening during anomalous period** **NM: Migration happening during normal period**					
DoS (HI)	0.996667	0.993377	1.0000	0.996678	0.996683
DoS (LI)	0.978333	0.967427	0.990000	0.978583	0.978648
DoS (HI-AM-MT1)[15])	0.991667	0967532	1.0	0.983498	0.983632
DoS (HI-NM-MT1)	0.96500	0.0	NaN[16]	0.0	NaN

[15]MT1:Represents a scenario characterisation where attack targets the VM that migrates to another host (live migration). AM represents whether the m migration occurs during the attack (AM) or during the normal period (NM) (i.e., 'migration overlap')

[16]The missing values are simply a matter of the detector not producing any true positives or false positives, therefore these metrics could not be calculated

6.3.6 Evaluation of Test Cases

A set of technical test cases have been defined to evaluate the integration of the *ADaaS* in use case defined in *SECCRIT* project [26], called "hosting critical urban mobility services in the cloud". This set of test cases reflects usual needs identified from operating installations of the use case, and demonstrate how the *ADaaS* deal with these needs and fulfil the tenant requirements. The test cases and their descriptions are presented in the Table. 6.4. The OpenStack platform is hosted in one of the Europe leading

TABLE 6.4: Description of Test Cases

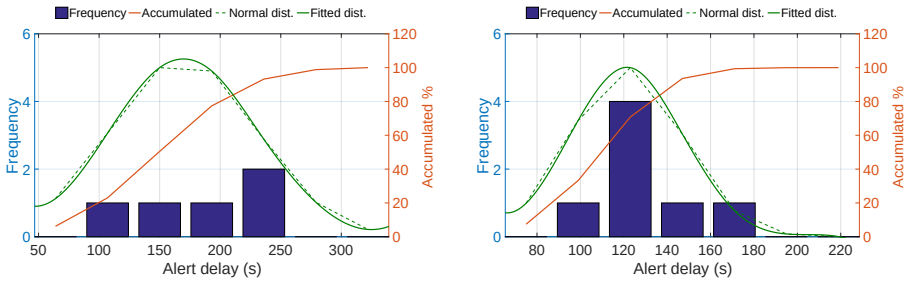
ID	**Test case**	**Description**
TC1	**Unexpected Database Growth**	In this scenario, the database grows unexpectedly in a certain period, and if growth continues, then the database server could run out of disk space, thus affecting the entire system. In this case *ADaaS* sends a notification to the operator in real time to trigger further remediation actions.
TC2	**Network Pattern Change**	Network traffic pattern between communications server and devices on public areas changes unexpectedly. This may indicate that "something is going wrong" and a warning must be sent to the operator.

telecommunication provider's research labs, who act as Cloud Provider (CP) and *ADaaS* is integrated to the cloud environment. Another Europe's critical service provider, who is tenant, installed and configured the replica of their crital services related to Traffic Management Systems (TMS) which consist of three virtual machines. These virtual machines include: *CommsMain* which runs the communication kernel, an application in charge of communicating with the devices on the street and providing necessary information to other applications; *UrbanMain* runs the actual traffic management system applications; and the *Database* VM runs a Microsoft SQL Server 2012 hosting all databases necessary for the correct operation of the system.

In order to build Test Case 1 (TC1) scenario, the customized stored procedure is installed in the SQL server, on the database. The script basically inserts new sensor data for a set of sensors at a given rate and parameters of the stored procedure control the growth rate. Similarly, for Test Case 2 (TC2), in order to simulate the on-street devices (mainly Traffic Controllers

and Variable Message Signs), simulators are provided that are running outside the cloud environment and provide background traffic to build the scenario. These simulators communicate via Internet with the *CommsMain*, and simulate around 90% of the existing traffic controllers in the city and are capable of responding to traffic control actions and sending sensor data. To inject anomalies, the traffic controller simulators running is manipulated, so anomalous traffic (random 256 bytes messages every 100ms) can be injected on demand. The traffic has been collected for 15 minutes on the virtual machine that communicates directly to the traffic controllers *CommsMain*. The trace is fed live to *ADaaS* and represents regular operation, system working normally for first five minutes interval (0-5). For the next five minutes interval (5-10) one of the traffic controllers injects additional traffic, consisting on random 256-byte long bursts every 100ms coming and for last five minutes interval (10-15) the anomaly stops and system returns to regular operation.

From system operator's point of view the alert about detected anomalies must be available in timely and interpretable manner, therefore, the notification engine of *Monsasca* is extended to send alerts back to tenant management system by providing a JSON webhook so that further subsequent remediation actions can be taken. Ten experimental run were performed and then computed the distribution of time taken from start of the event to be detected until the message appears in tenant management system hosted in service provider premises.



(A) Unexpected database growth (B) Network pattern change

FIGURE 6.15: TC1 and TC2 delay results

The Figures 6.15a and 6.15b represents the delay for TC1 and TC2 respectively. The use of several metrics for calculating the overall anomaly score allows the framework to minimize the probability of triggering false positives, at the cost of increasing the detection time. In addition, tests revealed an incremental tendency on the measured delays. This can be explained by the fact that the framework continuously adapts its definition of normality. Therefore, the first measurements taken (lower than 110 seconds) are the more significant timings representing the detection of specific problems in the systems. From the tenant point of view, this delay in getting the

alert is acceptable, since it provides useful information and allows an early intervention of the operators to check alerts coming from other processes and take proper actions in the system.

The test case demonstrates how the *ADaaS* framework monitors relevant metrics for detection of anomalies. The results from captured screen shots of running systems are presented in Appendix C. Figure C.1 and Figure C.2 show that the choice of feature is appropriate and sufficient for detecting changes in normal patterns. It can be observed that anomaly score going high when anomalies are happening and then going down when anomaly stops. The framework also allow us to monitor single metrics as oppose to collection of metrics (as shown in top row of Figure C.1). The anomaly score shown in figure is based on collection of metrics that are collected for each scenario.

6.4 Summary

In summary, and in contrast to the reported literature on cloud anomaly detection, the main contributions of the chapter are as follows:

1. The anomaly detection-as-a-service model is presented for Cloud infrastructures called *ADaaS*.

2. *ADaaS*uses an unsupervised, lightweight and memory-less detector based on a technique presented in Chapter 5.

3. The detector is evaluated using emulated data captured from a cloud testbed with respect to various attack types and intensities.

4. An elastic framework is developed for providing *ADaaS* to meet functional requirements in cloud operations to provide more secure environments to its consumers. The model is further tested in a real world cloud environment hosting critical services of European provider with high efficiency. The later deployment shows that the solution can capture real world applications interactions and can yield good performance ability.

   Expanding on 2) above, results show that proposed approach is effective in detection high and low intensity attacks with high precision and accuracy.

The cloud-specific data set labelled with ground truth of migration and anomalies, on which this work is based, is available on request. The implementation of *ADaaS* in Python is available online[17].

[17] https://forge.comp.lancs.ac.uk/svn-repos/seccrit-internal/public/Anomaly-detection-installation-notes.html

Chapter 7

Conclusion and Future Work

The new and intrinsic capabilities of cloud environments pose a number of security concerns that have not yet been fully assessed with respect to their implications on the performance of traditional off-the-shelf anomaly detection solutions. In particular, methods that empower the aspects of elasticity, resource management and service transparency such as live migration involve a range of system and network-wide activities. These activities are difficult to monitor, and interfere with the detection of anomalous incidents. In this work the impact of intra-cloud live migration is assessed over the *de facto* anomaly detection techniques within a controlled experimental cloud testbed. Through the results it is demonstrated that live migration has a measurable impact on underlying anomaly detection techniques since their performances vary under different intensities of attack scenarios. This work has provided initial insight towards identifying the correct requirements within the development process of cloud resilience management framework. The work also presents a novel real time anomaly-detection technique and the design of *ADaaS* as a model for operationally applying this technique in cloud environments.

7.1 Overview of Thesis

Chapter 2 covers relevant background information for this research and provides a detailed literature study on anomaly detection in cloud environments. Chapter 3 describes the experimental evaluation of state-of-the-art anomaly detection techniques in the context of live virtual machine migration, and under various attack types and intensities. The purpose of this evaluation was to gain an understanding of how migration affects the anomaly detection accuracy in a cloud environment. Informed by the issues identified in the experimental evaluation (Chapter 3), Chapter 4 describes the design and evaluation of a novel real time data density based anomaly detection technique that addresses many of the shortcomings of state-of-the-art anomaly detection methods in the cloud context. Chapter 5 presents the design and qualitative evaluation of a cloud resilience management framework that uses

proposed anomaly detection technique as its detection component. The design of the resilience management framework is also informed by the challenges identified through the experimental evaluation. Chapter 6 presents the design, implementation, and experimental results of the *ADaaS* model that operationalises anomaly detection techniques and enables a cloud provider to offer detection services based on security and resilience requirements of its customers and tenants.

7.2 Major Contributions

In the course of the cohesive research, the following technical contributions is made:

Cloud Resilience Management Framework (CRMF): The main focus of the evaluation presented in Chapter 3 has been to gain an understanding of how migration affects the anomaly detection accuracy in a cloud environment. The Cloud Resilience Management Framework (CRMF) (See Chapter 4) models and then applies an existing resilience strategy in a cloud operating context to diagnose anomalies. The framework uses end-to-end feedback loop that allows remediation to be integrated with existing cloud management systems. The key aspects of the framework are demonstrated in the prototype, namely: anomaly detection for network and system level analysis, and remediation strategies with the aid of a policy engine. An experimental infrastructure is then built to include a real cloud infrastructure resembling two cloud data centres. This is further extended with implementations of resilience management systems to realise resilience management under different attack scenarios.

One possible future research direction is to further refine the framework as a result of lessons learned from the development of the prototype. This mainly includes analytical capabilities to different outside applications. Moreover, additions to the framework will be made to cover for extra aspects to be identified in due course – which may include the need for inter-provider resilience management and support for more resilience strategies to combat challenges specific to service performance.

Management and resilience of the cloud environments are closely linked. Resilience in cloud environments need to be managed to disseminate relevant policies to the cloud provider that will implement them. Policies specifies actions which are needed to deal with challenges and due to varying nature of challenges these policies need to be further refined to be adaptive in response to challenges. The recent work is presented on refinement process for policies that has iterated phases of decompositions at its core. There are many avenues for future work in this discipline but particularly interesting is in policy based resilience management in the cloud.

It would also be interesting to explore how example polices e.g described in templates can be optimised with refinement process. The work thesis presented on refinement procedures can be merged with the policy analysis framework in future to see whether this can be used to guide the resilience against challenges. There are a number of different conflicts that can arise from policies. For example, some policies will trigger complex management procedures which require the execution of actions that may be specified as part of different policies. Determining the existence of conflicting configurations for cloud environments is of critical importance. Under the assumption that the system may be loaded with a number of policies to address many different challenges simultaneously. Multiple policies that need to coexist may specify conflicting actions on the same virtual resources, or may trigger the activation of incompatible mechanisms, thereby rendering a particular resilience strategy ineffective. Further, to investigate approaches for the automatic identification and resolution of policy conflicts. One possible approach around solution of this problem is the use of meta-policies which resolve conflicting situations during run-time.

Novel Anomaly Detection Algorithm: The thesis presents a modified anomaly detector based on the density of observed feature vectors (see Chapter 5). The modifications made have achieved improved detection for dynamically evolving workload patterns without having pre-defined anomaly models. This encourages further work concerning monitoring scalability in terms of efficiency and accuracy with cloud specific workloads. The density computation is expressed recursively, i.e., based only on the signal at the previous step together with the latest vector. This makes the algorithm memoryless, i.e., it does not need to store historical data. The lightweight nature of this approach makes it particularly suitable for deployment in cloud environments. In particular, a detector associated with a specific VM is sufficiently lightweight to travel with the migrating VM. This is especially valuable, as a detector that does not migrate with a VM may interpret the sudden departure or arrival of the VM as an anomaly.

Anomaly Detection as a Service (*ADaaS*): The thesis proposed an Anomaly Detection as a Service Model for cloud infrastructures called *ADaaS* (see Chapter 6). The *ADaaS* uses an unsupervised, lightweight and memoryless detector. The embodied technique for detection is based on density of observed feature vectors. *ADaaS* monitors a range of metrics (VM, host and network) using *Monasca* API to diagnose anomalies. The extensive evaluation of *ADaaS* is conducted offline, using representative cloud workloads, as well as online, running in European critical infrastructure provider's setup for real time anomaly detection. The results show that *ADaaS* achieves high accuracy with low false alarm rate in detection anomalies. To the best of the knowledge, *ADaaS* is the first system that offer anomaly detection

as a service for cloud infrastructure. Cloud infrastructure will benefit from this proposed approach since it is designed to be flexible and distributed as well as being able to detect anomalies in real time. The future work will include identification of anomalies to zoom in detection to focus on remediation actions in response to detected anomalies using service model. Further work will look into integrating the *ADaaS* with Software Defined Networking (SDN) solutions because these new evolving technologies will create new vulnerabilities to the cloud computing environments; therefore an anomaly detection system would be needed to reduce the overall risk.

7.3 Future Work

The future work will look into integrating *ADaaS* with Software Defined Networking (SDN) solutions because these new evolving technologies will create new vulnerabilities to the cloud computing environments; therefore an anomaly detection system would be needed to reduce the overall risk. It is also plan to further refine the framework using lessons learned from the development of the prototype. This mainly includes specific analytical capabilities for different outside applications. Moreover, new capabilities to the framework may be introduced, which may include the need for inter-provider resilience management and support for more resilience strategies to combat challenges specific to service performance.

The future work will also explore how example polices e.g. described in templates can be optimised with refinement process. The main aim would be to merge the refinement procedures develop in course of this work with the policy analysis framework to see whether this can be used to guide the resilience against challenges. There are a number of different conflicts that can arise from policies. For example, some policies will trigger complex management procedures which require the execution of actions that may be specified as part of different policies. Determining the existence of conflicting configurations for cloud environments is of critical importance. There is an assumption that the system may be loaded with a number of policies to address many different challenges simultaneously. Multiple policies that need to coexist may specify conflicting actions on the same virtual resources, or may trigger the activation of incompatible mechanisms, thereby rendering a particular resilience strategy ineffective. Further, to investigate approaches for the automatic identification and resolution of policy conflicts. One possible approach around solution of this problem is the use of meta-policies which resolve conflicting situations during at run time.

Management and resilience of the cloud environments are closely linked. Resilience in cloud environments need to be managed to disseminate relevant

policies to the cloud provider that will implement them. Policies specifies actions which are needed to deal with challenges and due to varying nature of challenges these policies need to be further refined to be adaptive in response to challenges.The recent work on refinement process is presented for policies that has iterated phases of decompositions at its core. There are many avenues for future work in this discipline but in particular interesting one in policy based resilience management in the cloud. As cloud environments grow, the cost and performance of management also increase. Controlling and optimizing resources for resilience management purpose can only become more critical due to scale and novel challenges. In this research it is shown that different resilience mechanisms can be used with the aid of policies to change the behaviour of how the system should respond to challenges. However, each policy can yield different cost and performance. The focus is needed on the design of a management plane which can realize appropriate resilience strategies by deploying the most cost effective and optimized mechanisms that are needed in response to challenges.

Appendix A

Results of Migration on Anomaly Detection

A.1 Combined dataset BC0-PS-AH

A.1.1 Migration effect BC0-PS-AH-PCA

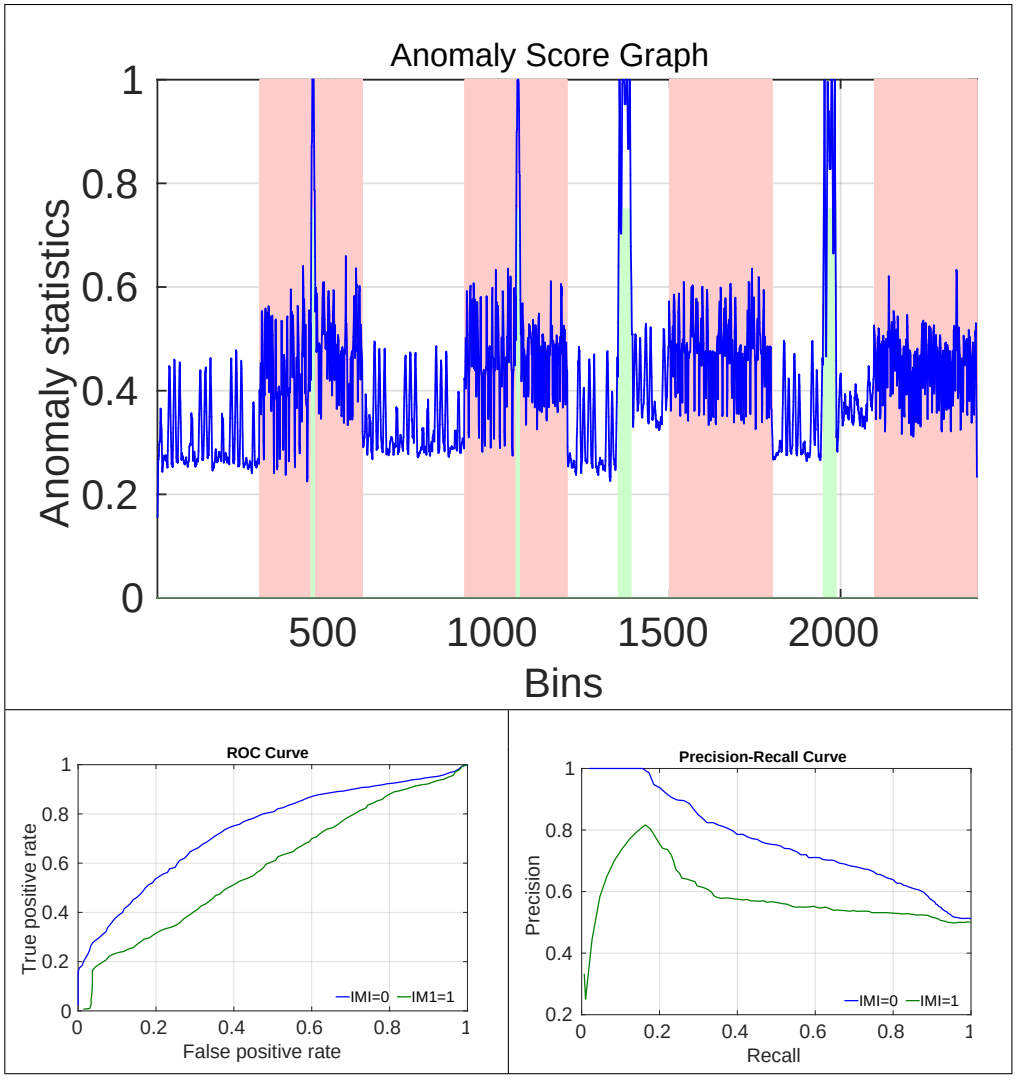


TABLE A.1: Anomaly Score Graph, Effect of migration on ROC/PRC for PCA with combined dataset BC0-PS-AH

PCA with BC0-PS-AH	Without migration	With migration
Error rate	0.1651	0.2080
Detection rate	83.49%	79.2%

TABLE A.2: Effect of migration on PCA with BC0-PS-AH

Table A.1 summarizes anomaly detector PCA when applied to dataset BC0-PS-AH. The top half shows the Anomaly Score Graph (ASG). In the bottom-left, two ROCs are generated by partitioning the ASG labelled with anomaly and migration ground truths into those with $m = 0$ vs. $m = 1$. In the bottom-right, two PRCs are similarly generated. Table A.2 summarizes the error/detection rates with/without migration present.

A.1.2 Migration effect BC0-PS-AH-KM

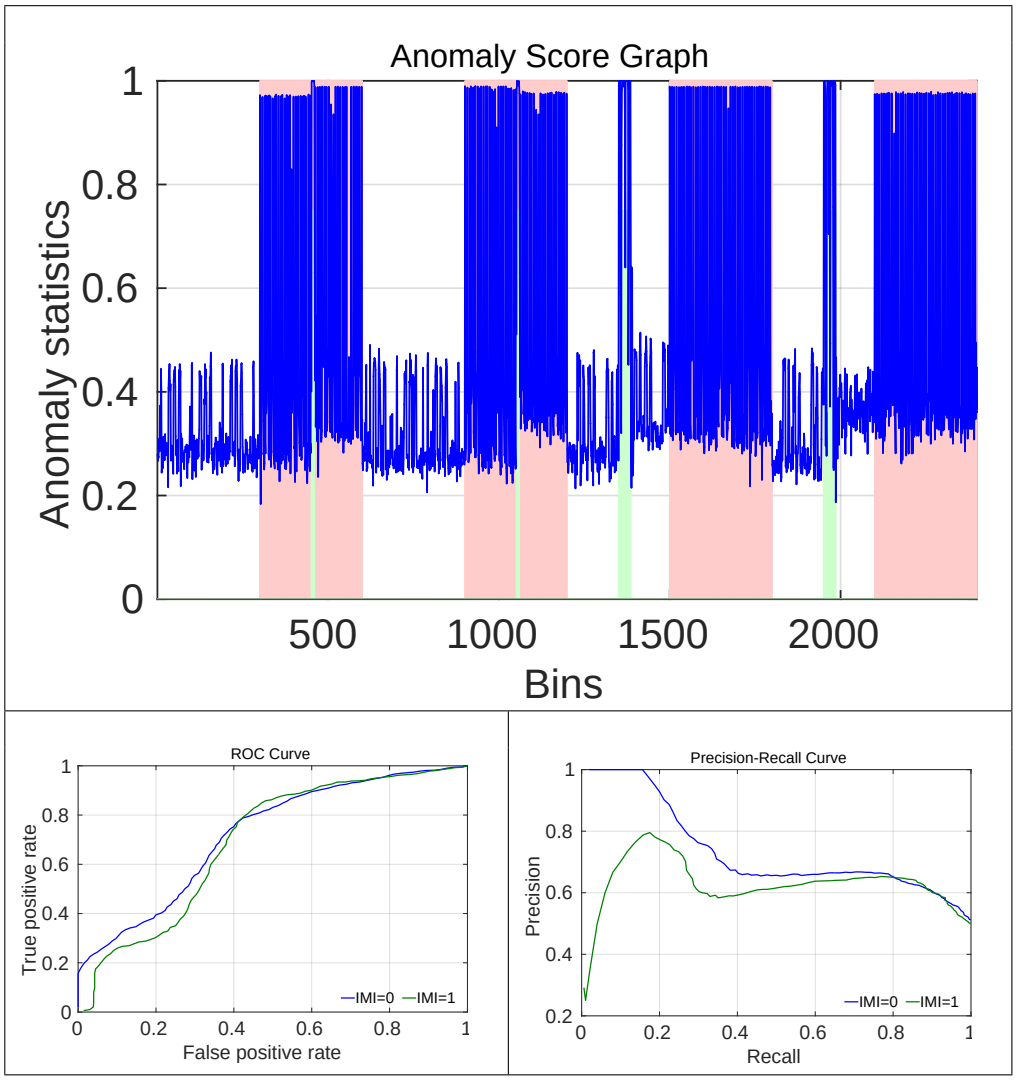


TABLE A.3: Anomaly Score Graph, Effect of migration on ROC/PRC for KM with combined dataset BC0-PS-AH

KM with BC0-PS-AH	Without migration	With migration
Error rate	0.4072	0.4318
Detection rate	59.28%	56.82%

TABLE A.4: Effect of migration on KM with BC0-PS-AH

Table A.3 summarizes anomaly detector KM when applied to dataset BC0-PS-AH. The top half shows the Anomaly Score Graph (ASG). In the bottom-left, two ROCs are generated by partitioning the ASG labelled with anomaly and migration ground truths into those with $m = 0$ vs. $m = 1$. In the bottom-right, two PRCs are similarly generated. Table A.4 summarizes the error/detection rates with/without migration present.

A.1.3 Migration effect BC0-PS-AH-NB

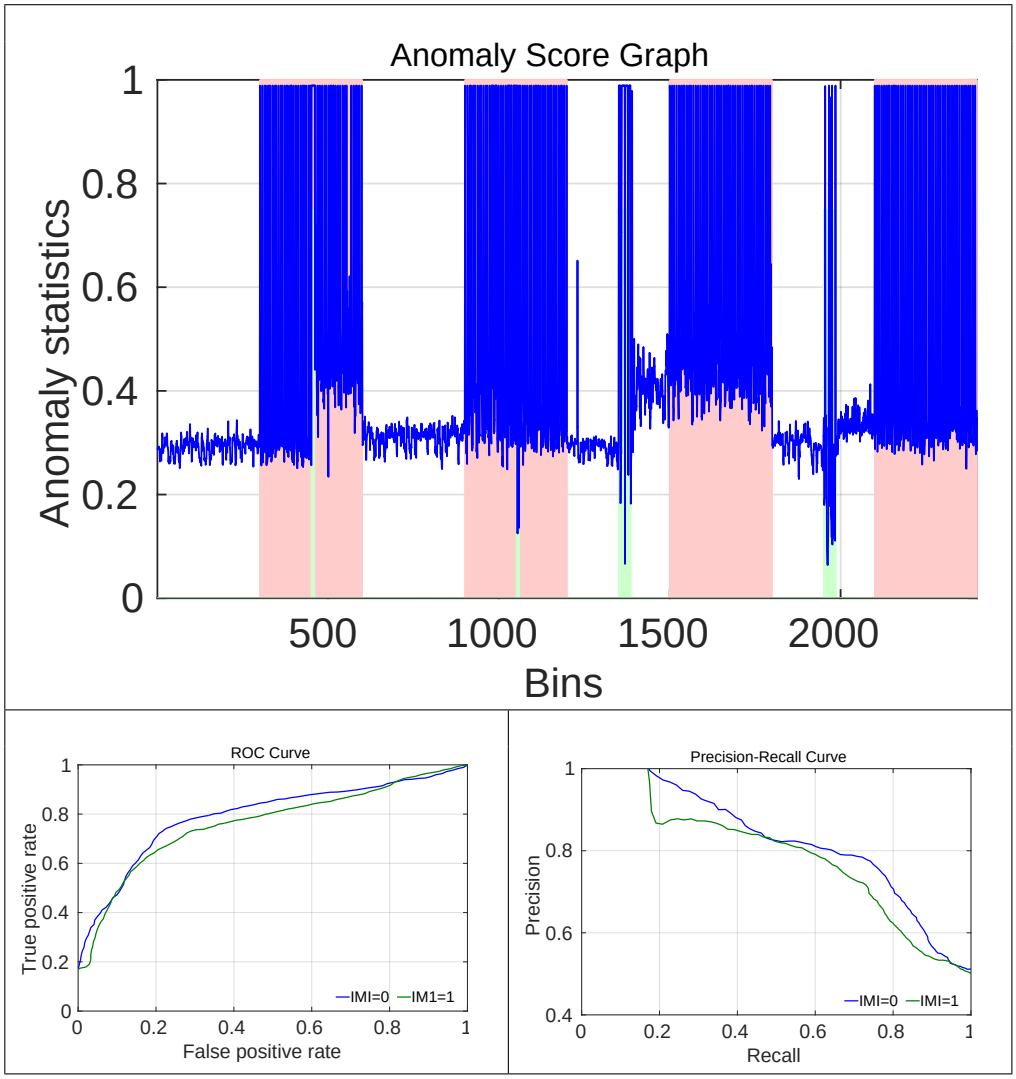


TABLE A.5: Anomaly Score Graph, Effect of migration on ROC/PRC for NB with combined dataset BC0-PS-AH

NB with BC0-PS-AH	Without migration	With migration
Error rate	0.2851	0.3143
Detection rate	71.49%	68.57%

TABLE A.6: Effect of migration on NB with BC0-PS-AH

Table A.5 summarizes anomaly detector NB when applied to dataset BC0-PS-AH. The top half shows the Anomaly Score Graph (ASG). In the bottom-left, two ROCs are generated by partitioning the ASG labelled with anomaly and migration ground truths into those with $m = 0$ vs. $m = 1$. In the bottom-right, two PRCs are similarly generated. Table A.6 summarizes the error/detection rates with/without migration present.

A.1.4 Migration effect BC0-PS-AH-EMGM

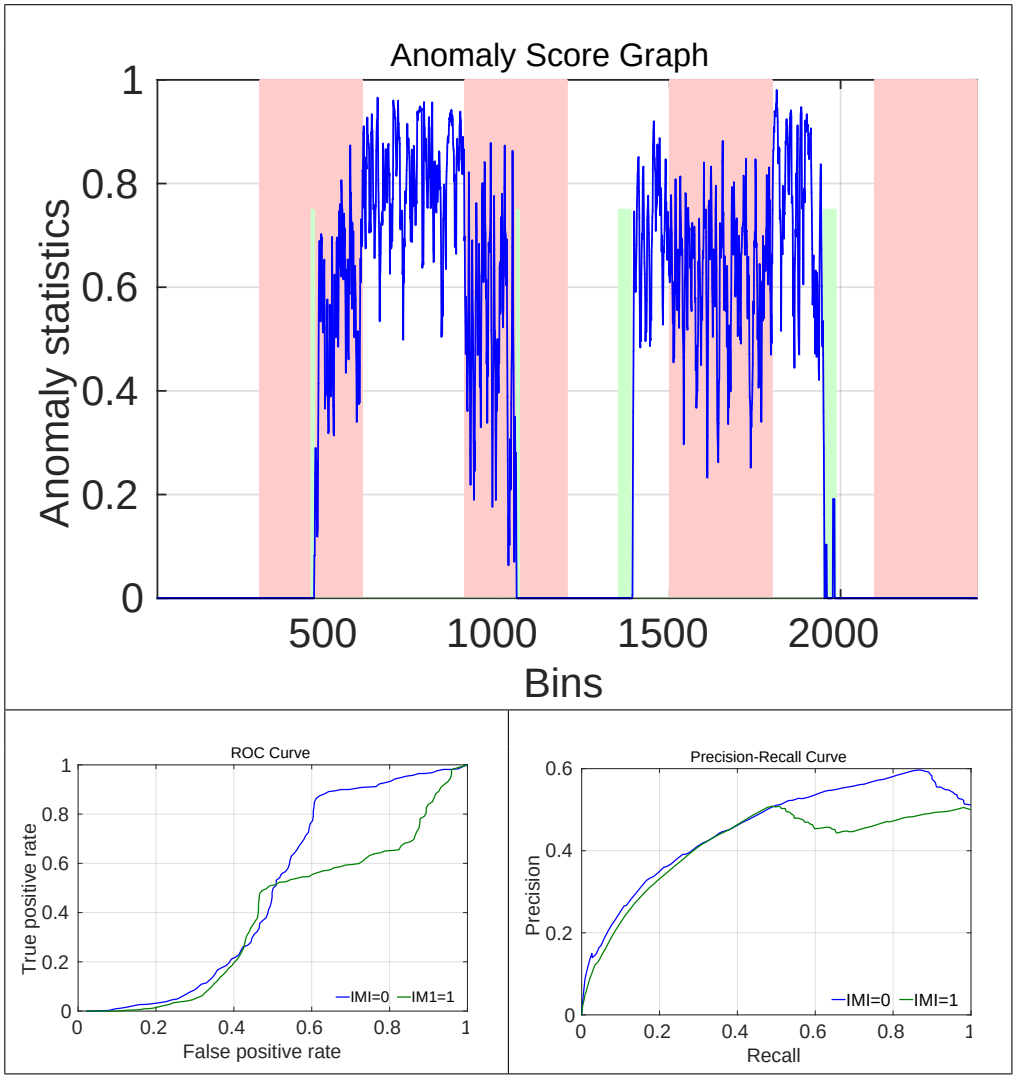


TABLE A.7: Anomaly Score Graph, Effect of migration on ROC/PRC for EMGM with combined dataset BC0-PS-AH

EMGM with BC0-PS-AH	Without migration	With migration
Error rate	0.5589	0.5506
Detection rate	44.11%	44.94%

TABLE A.8: Effect of migration on EMGM with BC0-PS-AH

Table A.7 summarizes anomaly detector EMGM when applied to dataset BC0-PS-AH. The top half shows the Anomaly Score Graph (ASG). In the bottom-left, two ROCs are generated by partitioning the ASG labelled with anomaly and migration ground truths into those with $m = 0$ vs. $m = 1$. In the bottom-right, two PRCs are similarly generated. Table A.8 summarizes the error/detection rates with/without migration present.

A.2 Combined dataset BC0-PS-AL

A.2.1 Migration effect BC0-PS-AL-PCA

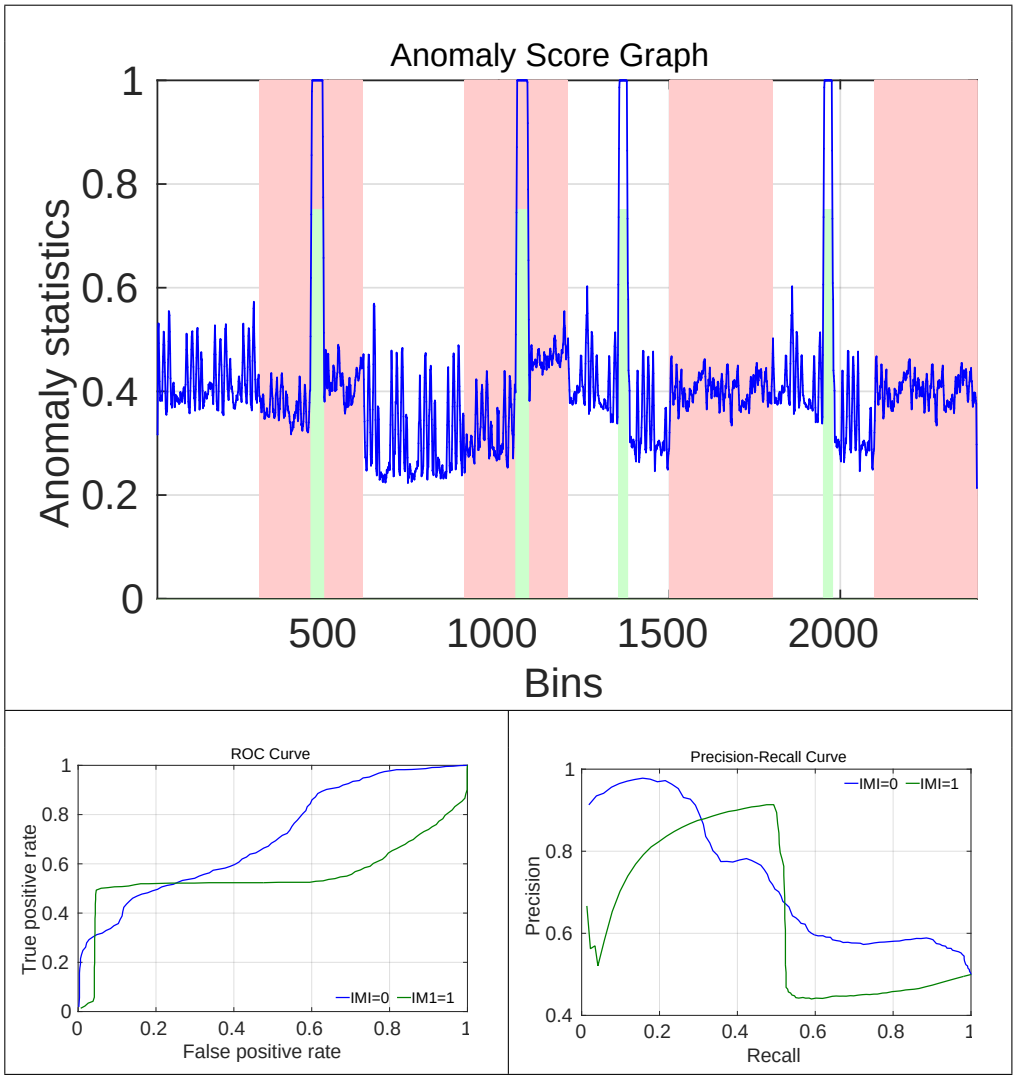


TABLE A.9: Anomaly Score Graph, Effect of migration on ROC/PRC for PCA with combined dataset BC0-PS-AL

PCA with BC0-PS-AL	Without migration	With migration
Error rate	0.3762	0.4879
Detection rate	62.38%	51.21%

TABLE A.10: Effect of migration on PCA with BC0-PS-AL

Table A.9 summarizes anomaly detector PCA when applied to dataset BC0-PS-AL. The top half shows the Anomaly Score Graph (ASG). In the bottom-left, two ROCs are generated by partitioning the ASG labelled with anomaly and migration ground truths into those with $m = 0$ vs. $m = 1$. In the bottom-right, two PRCs are similarly generated. Table A.10 summarizes the error/detection rates with/without migration present.

A.2.2 Migration effect BC0-PS-AL-KM

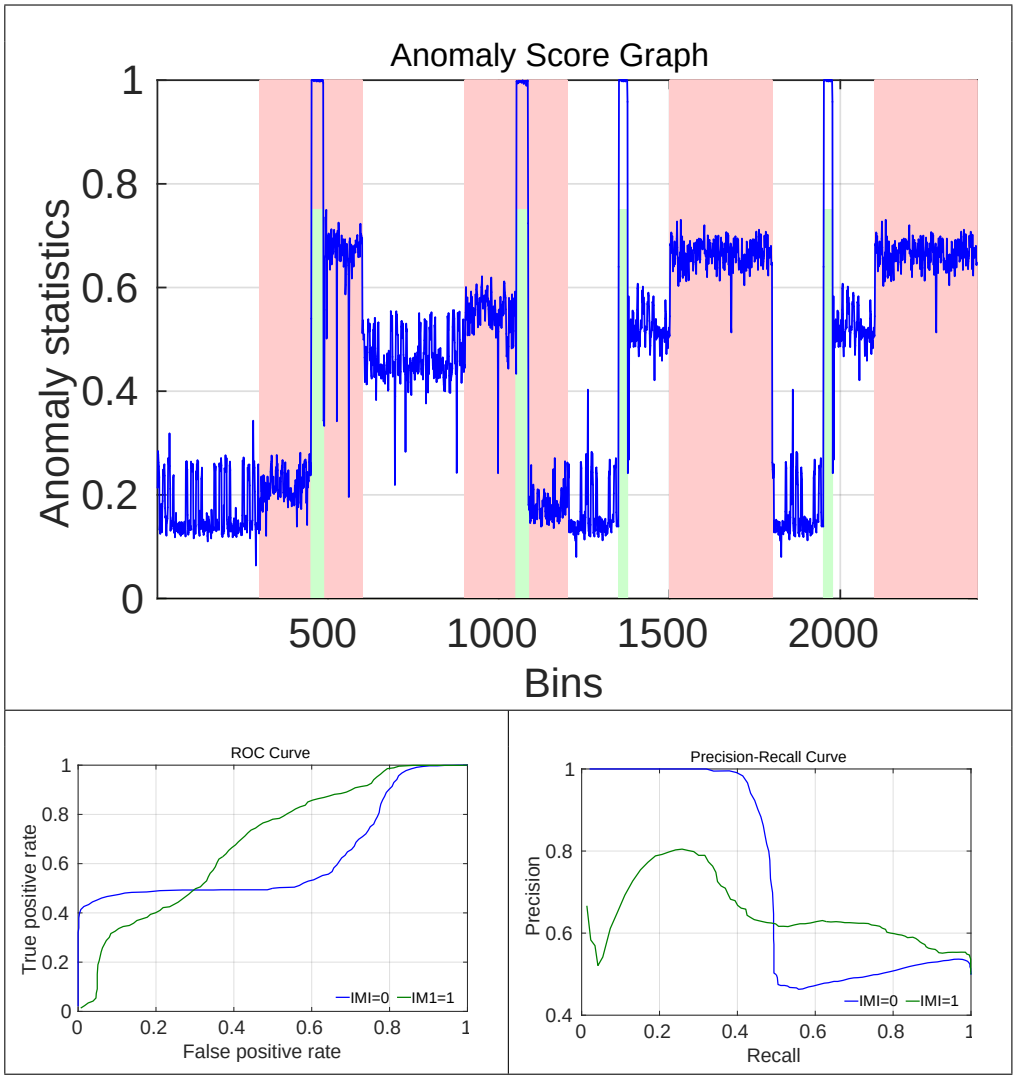


TABLE A.11: Anomaly Score Graph, Effect of migration on ROC/PRC for KM with combined dataset BC0-PS-AL

KM with BC0-PS-AL	Without migration	With migration
Error rate	0.5018	0.3
Detection rate	49.82%	51.21%

TABLE A.12: Effect of migration on KM with BC0-PS-AL

Table A.11 summarizes anomaly detector KM when applied to dataset BC0-PS-AL. The top half shows the Anomaly Score Graph (ASG). In the bottom-left, two ROCs are generated by partitioning the ASG labelled with anomaly and migration ground truths into those with $m = 0$ vs. $m = 1$. In the bottom-right, two PRCs are similarly generated. Table A.12 summarizes the error/detection rates with/without migration present.

A.2.3 Migration effect BC0-PS-AL-NB

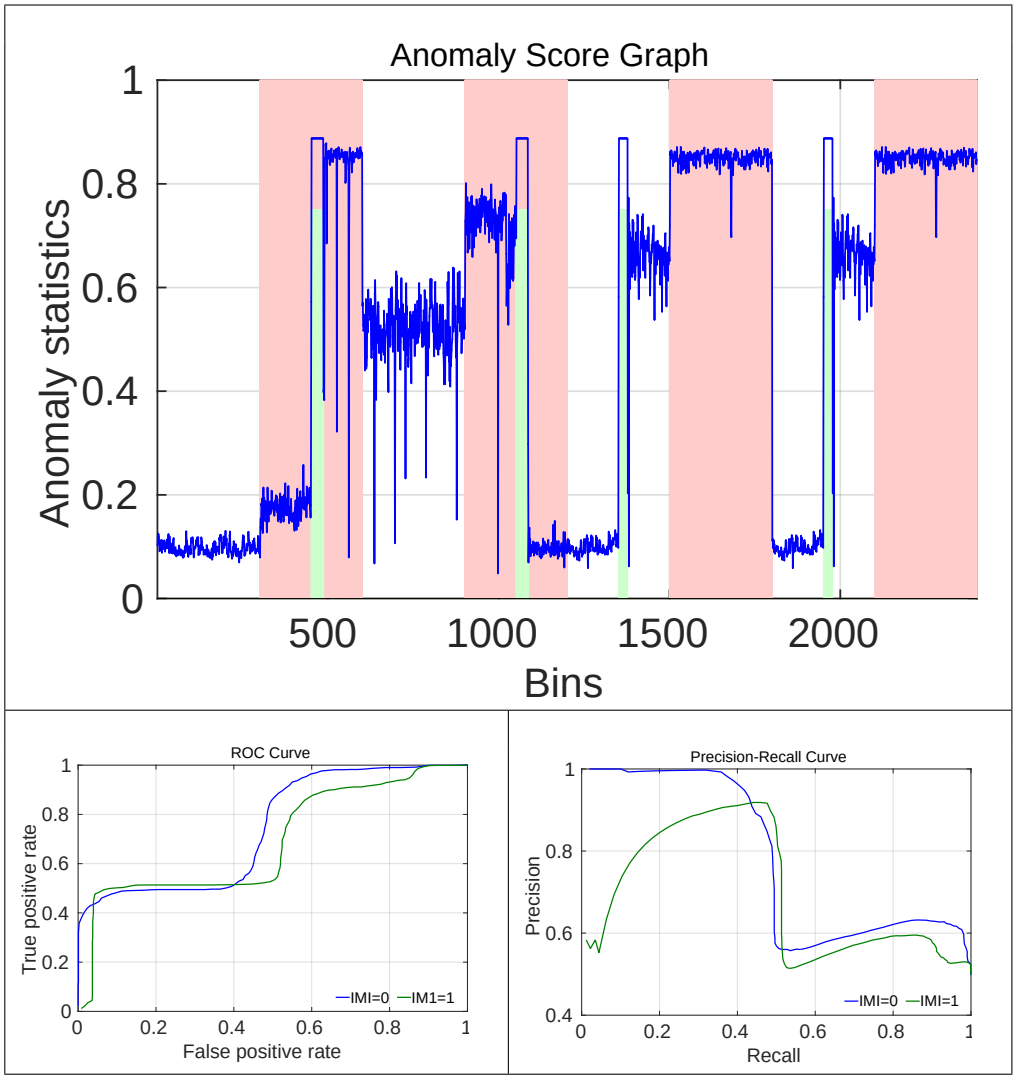


TABLE A.13: Anomaly Score Graph, Effect of migration on ROC/PRC for NB with combined dataset BC0-PS-AL

NB with BC0-PS-AL	Without migration	With migration
Error rate	0.4126	0.4362
Detection rate	58.74%	56.38%

TABLE A.14: Effect of migration on NB with BC0-PS-AL

Table A.13 summarizes anomaly detector NB when applied to dataset BC0-PS-AL. The top half shows the Anomaly Score Graph (ASG). In the bottom-left, two ROCs are generated by partitioning the ASG labelled with anomaly and migration ground truths into those with $m = 0$ vs. $m = 1$. In the bottom-right, two PRCs are similarly generated. Table A.14 summarizes the error/detection rates with/without migration present.

A.2.4 Migration effect BC0-PS-AL-EMGM

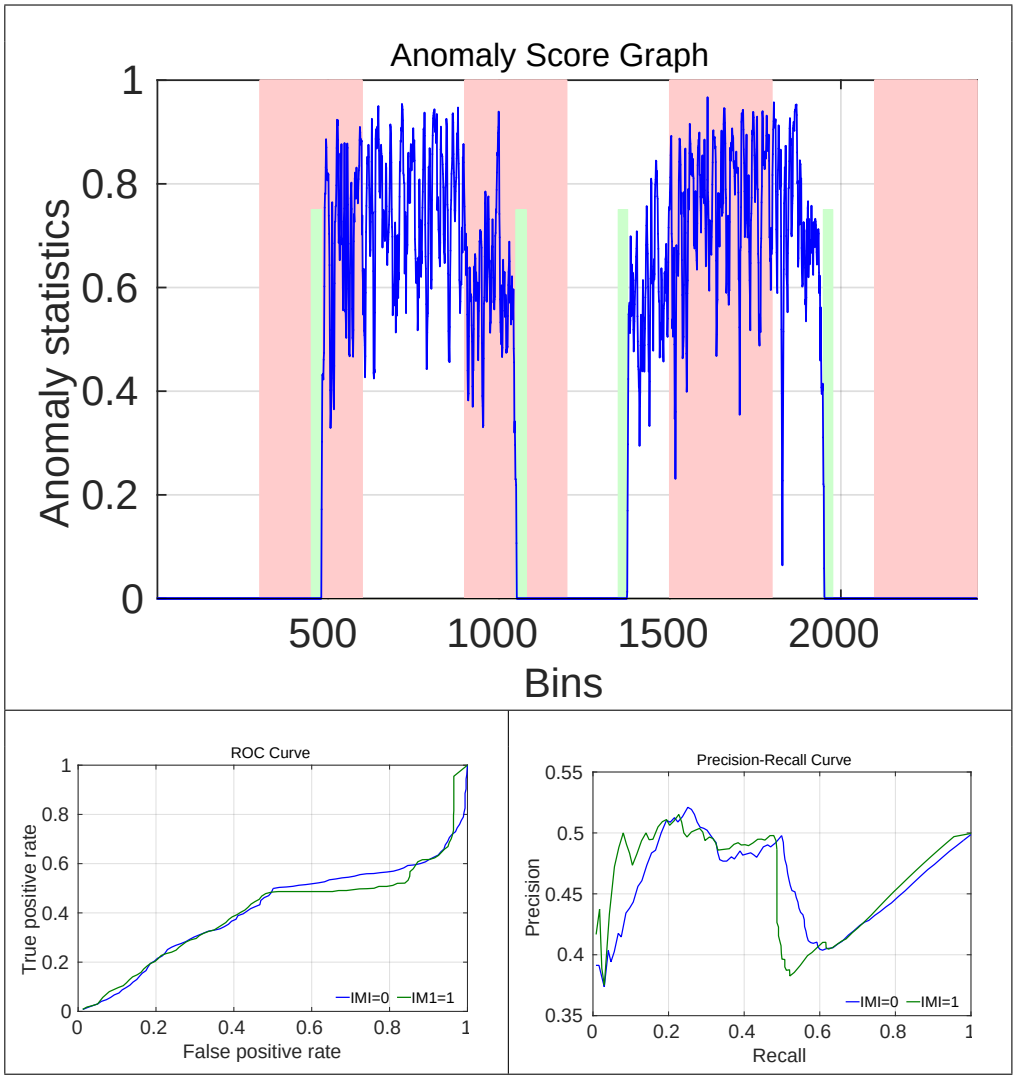


TABLE A.15: Anomaly Score Graph, Effect of migration on ROC/PRC for EMGM with combined dataset BC0-PS-AL

EMGM with BC0-PS-AL	Without migration	With migration
Error rate	0.5013	0.5033
Detection rate	49.87%	49.67%

TABLE A.16: Effect of migration on EMGM with BC0-PS-AL

Table A.15 summarizes anomaly detector EMGM when applied to dataset BC0-PS-AL. The top half shows the Anomaly Score Graph (ASG). In the bottom-left, two ROCs are generated by partitioning the ASG labelled with anomaly and migration ground truths into those with $m = 0$ vs. $m = 1$. In the bottom-right, two PRCs are similarly generated. Table A.16 summarizes the error/detection rates with/without migration present.

A.3 Combined dataset BC0-DoS-AL-MT1

A.3.1 Migration effect BC0-DoS-AL-MT1-PCA

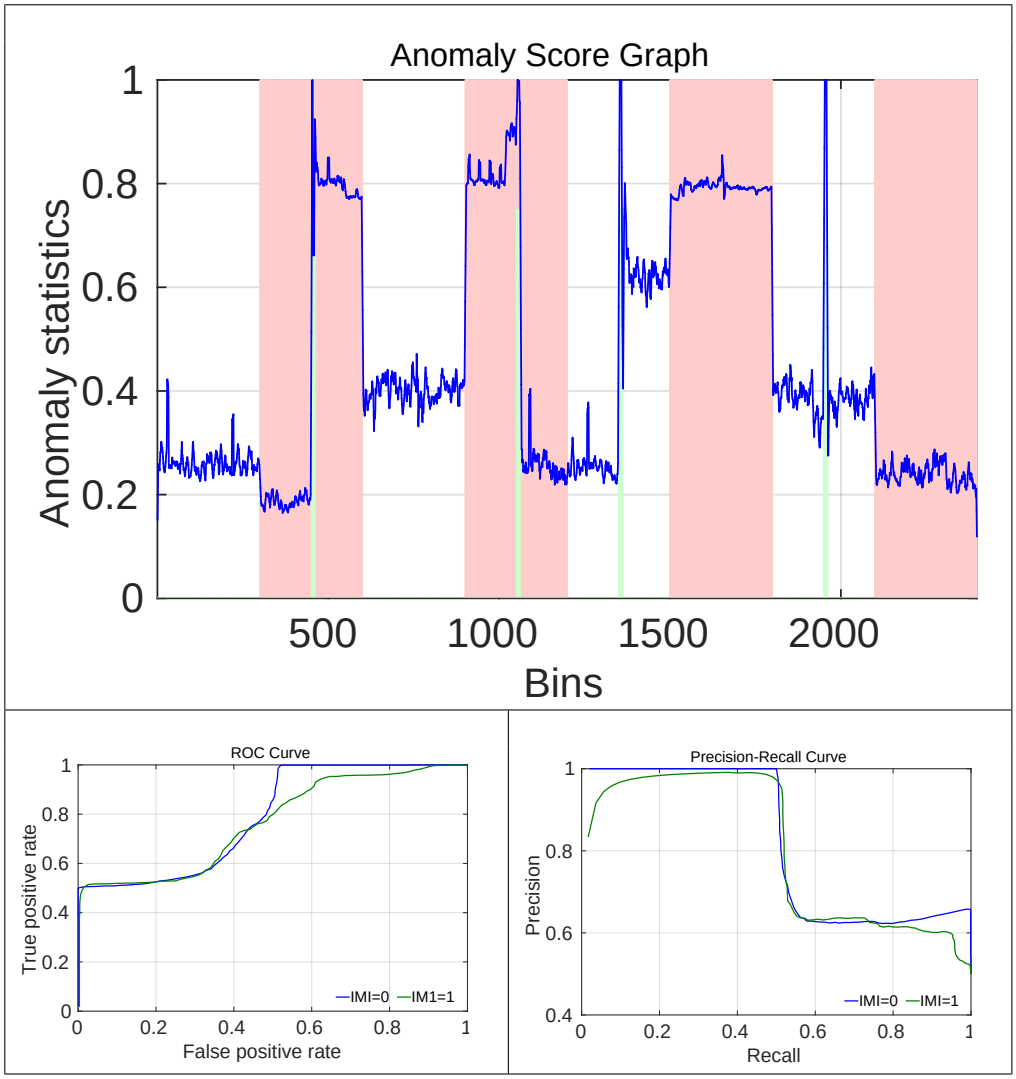


TABLE A.17: Anomaly Score Graph, Effect of migration on ROC/PRC for PCA with combined dataset BC0-DoS-AL-MT1

Table A.17 summarizes anomaly detector PCA when applied to dataset BC0-DoS-AL-MT1. The top half shows the Anomaly Score Graph (ASG). In the bottom-left, two ROCs are generated by partitioning the ASG labelled with anomaly and migration ground truths into those with $m = 0$ vs. $m = 1$. In the bottom-right, two PRCs are similarly generated. Table A.18 summarizes the error/detection rates with/without migration present.

A.3.2 Migration effect BC0-DoS-AL-MT1-KM

Table A.19 summarizes anomaly detector KM when applied to dataset BC0-DoS-AL-MT1. The top half shows the Anomaly Score Graph (ASG). In the

PCA with BC0-DoS-AL-MT1	Without migration	With migration
Error rate	0.3799	0.2606
Detection rate	62.01%	73.94%

TABLE A.18: Effect of migration on PCA with BC0-DoS-AL-MT1

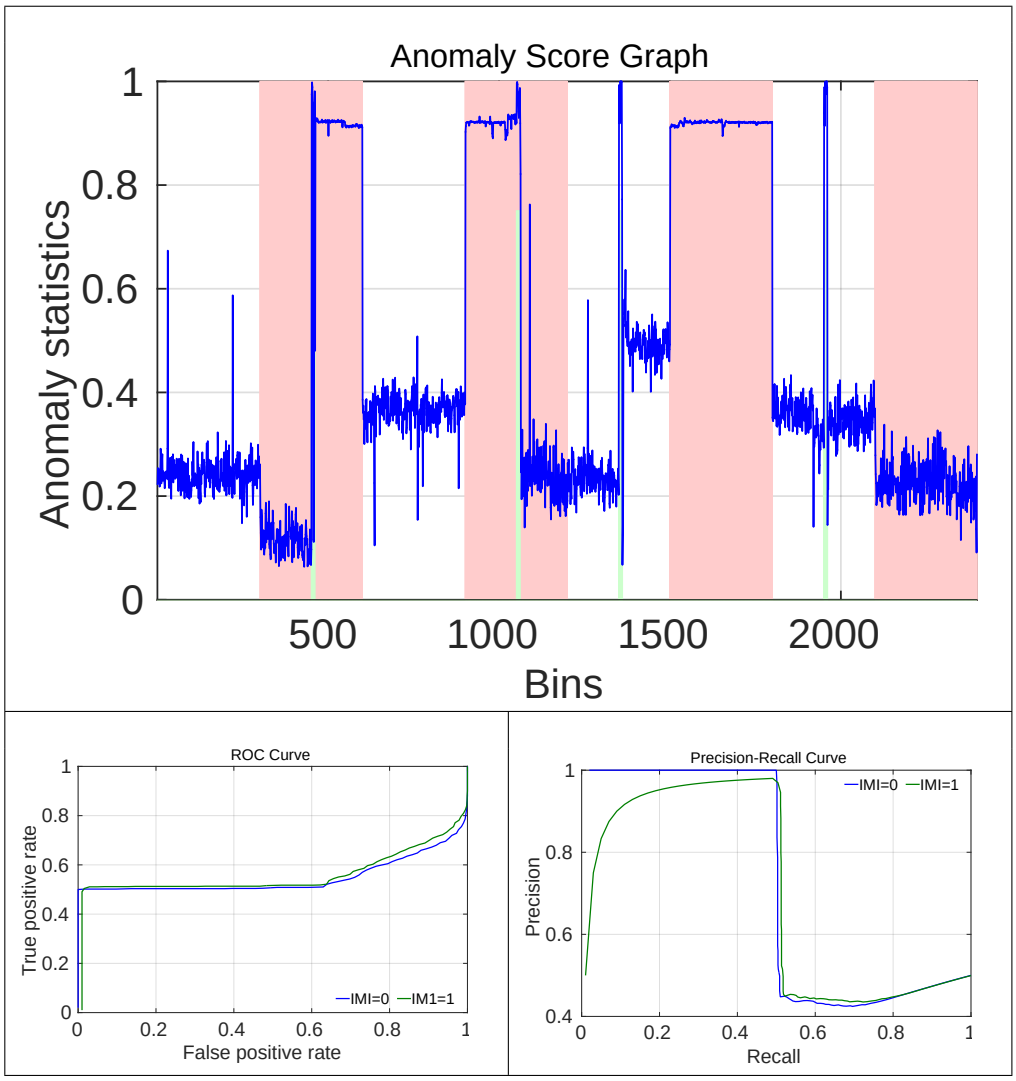


TABLE A.19: Anomaly Score Graph, Effect of migration on ROC/PRC for KM with combined dataset BC0-DoS-AL-MT1

KM with BC0-DoS-AL-MT1	Without migration	With migration
Error rate	0.3633	0.3119
Detection rate	63.67%	68.81%

TABLE A.20: Effect of migration on KM with BC0-DoS-AL-MT1

bottom-left, two ROCs are generated by partitioning the ASG labelled with anomaly and migration ground truths into those with $m = 0$ vs. $m = 1$. In the bottom-right, two PRCs are similarly generated. Table A.20 summarizes the error/detection rates with/without migration present.

A.3.3 Migration effect BC0-DoS-AL-MT1-NB

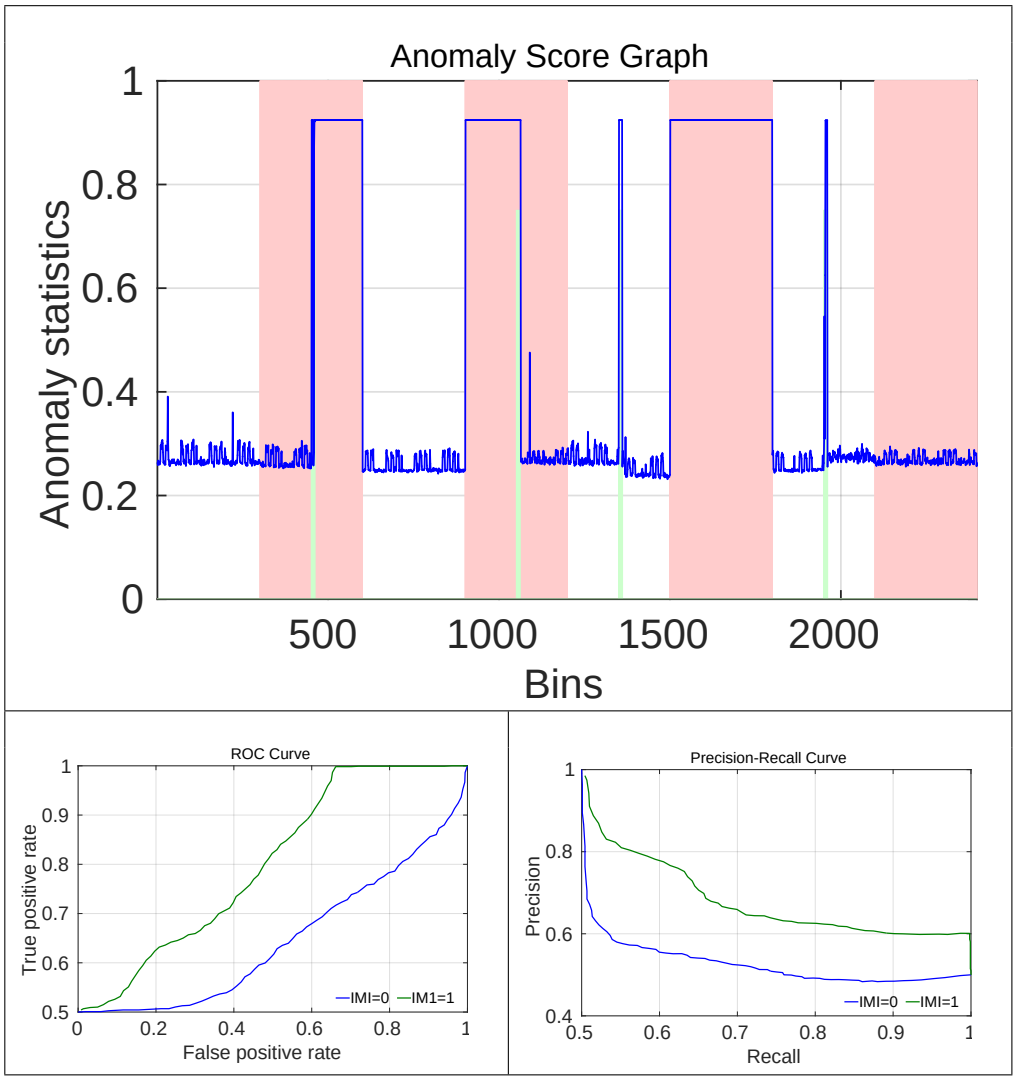


TABLE A.21: Anomaly Score Graph, Effect of migration on ROC/PRC for NB with combined dataset BC0-DoS-AL-MT1

Table A.21 summarizes anomaly detector NB when applied to dataset BC0-DoS-AL-MT1. The top half shows the Anomaly Score Graph (ASG). In the bottom-left, two ROCs are generated by partitioning the ASG labelled with anomaly and migration ground truths into those with $m = 0$ vs. $m = 1$. In the bottom-right, two PRCs are similarly generated. Table A.22 summarizes the error/detection rates with/without migration present.

NB with BC0-DoS-AL-MT1	Without migration	With migration
Error rate	0.2538	0.2531
Detection rate	74.62%	74.69%

TABLE A.22: Effect of migration on NB with BC0-DoS-AL-MT1

A.3.4 Migration effect BC0-DoS-AL-MT1-EMGM

Table A.23 summarizes anomaly detector EMGM when applied to dataset BC0-DoS-AL-MT1. The top half shows the Anomaly Score Graph (ASG). In the bottom-left, two ROCs are generated by partitioning the ASG labelled with anomaly and migration ground truths into those with $m = 0$ vs. $m = 1$. In the bottom-right, two PRCs are similarly generated. Table A.24 summarizes the error/detection rates with/without migration present.

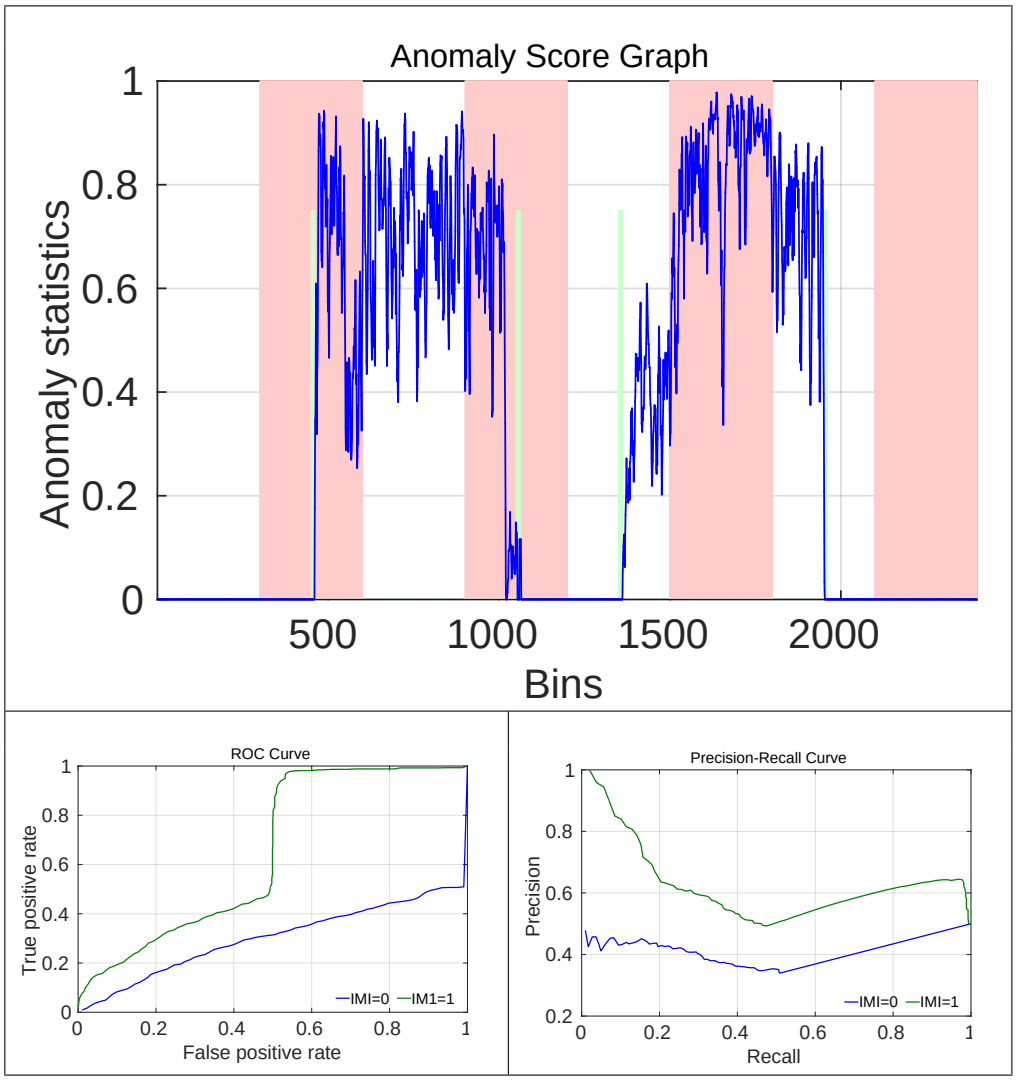


TABLE A.23: Anomaly Score Graph, Effect of migration on ROC/PRC for EMGM with combined dataset BC0-DoS-AL-MT1

EMGM with BC0-DoS-AL-MT1	Without migration	With migration
Error rate	0.6942	0.4887
Detection rate	30.58%	51.13%

TABLE A.24: Effect of migration on EMGM with BC0-DoS-AL-MT1

A.4 Combined dataset BC0-DoS-AL-MT0

A.4.1 Migration effect BC0-DoS-AL-MT0-PCA

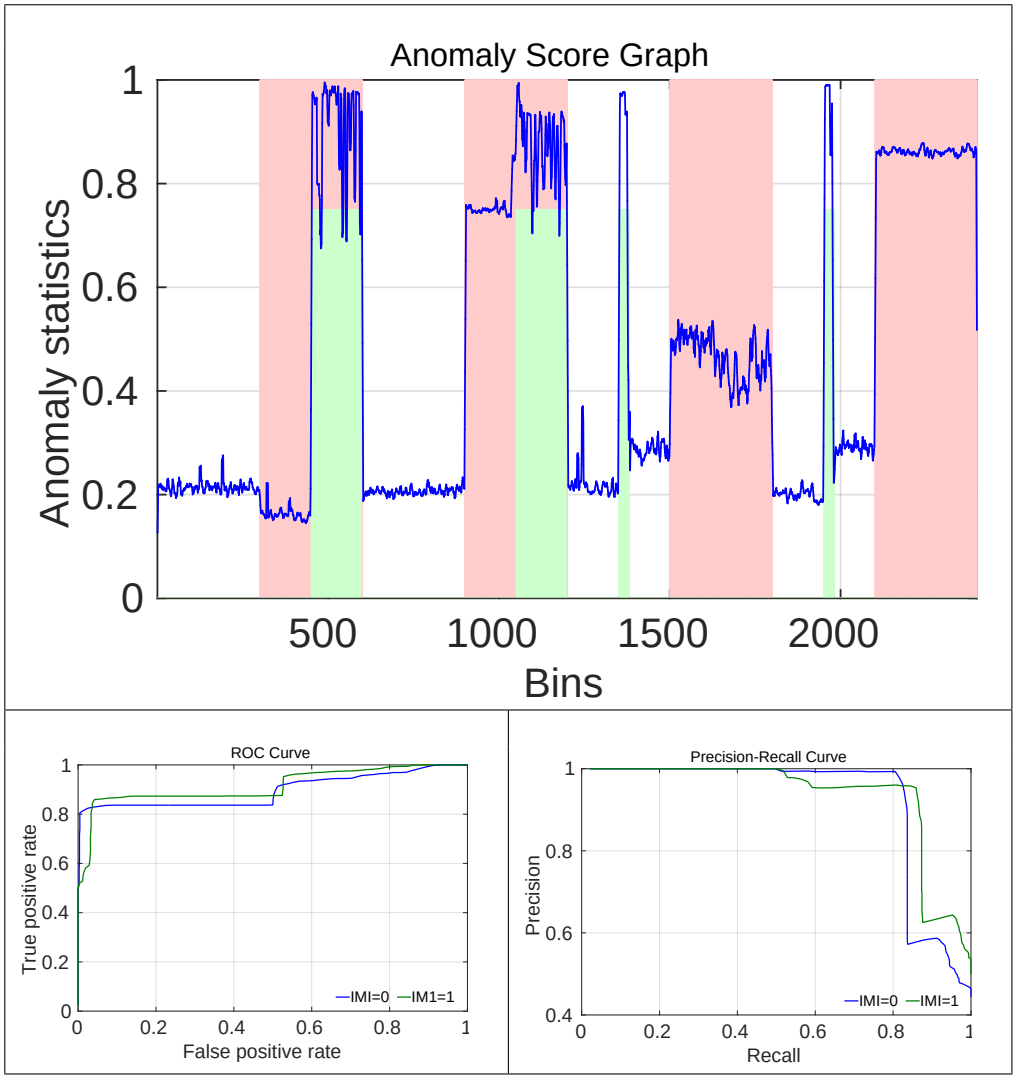


TABLE A.25: Anomaly Score Graph, Effect of migration on ROC/PRC for PCA with combined dataset BC0-DoS-AL-MT0

Table A.25 summarizes anomaly detector PCA when applied to dataset BC0-DoS-AL-MT0. The top half shows the Anomaly Score Graph (ASG). In the bottom-left, two ROCs are generated by partitioning the ASG labelled with anomaly and migration ground truths into those with $m = 0$ vs. $m = 1$. In the bottom-right, two PRCs are similarly generated. Table A.26 summarizes the error/detection rates with/without migration present.

A.4.2 Migration effect BC0-DoS-AL-MT0-KM

Table A.27 summarizes anomaly detector KM when applied to dataset BC0-DoS-AL-MT0. The top half shows the Anomaly Score Graph (ASG). In the

PCA with BC0-DoS-AL-MT0	Without migration	With migration
Error rate	0.1177	0.1317
Detection rate	88.23%	86.83%

TABLE A.26: Effect of migration on PCA with BC0-DoS-AL-MT0

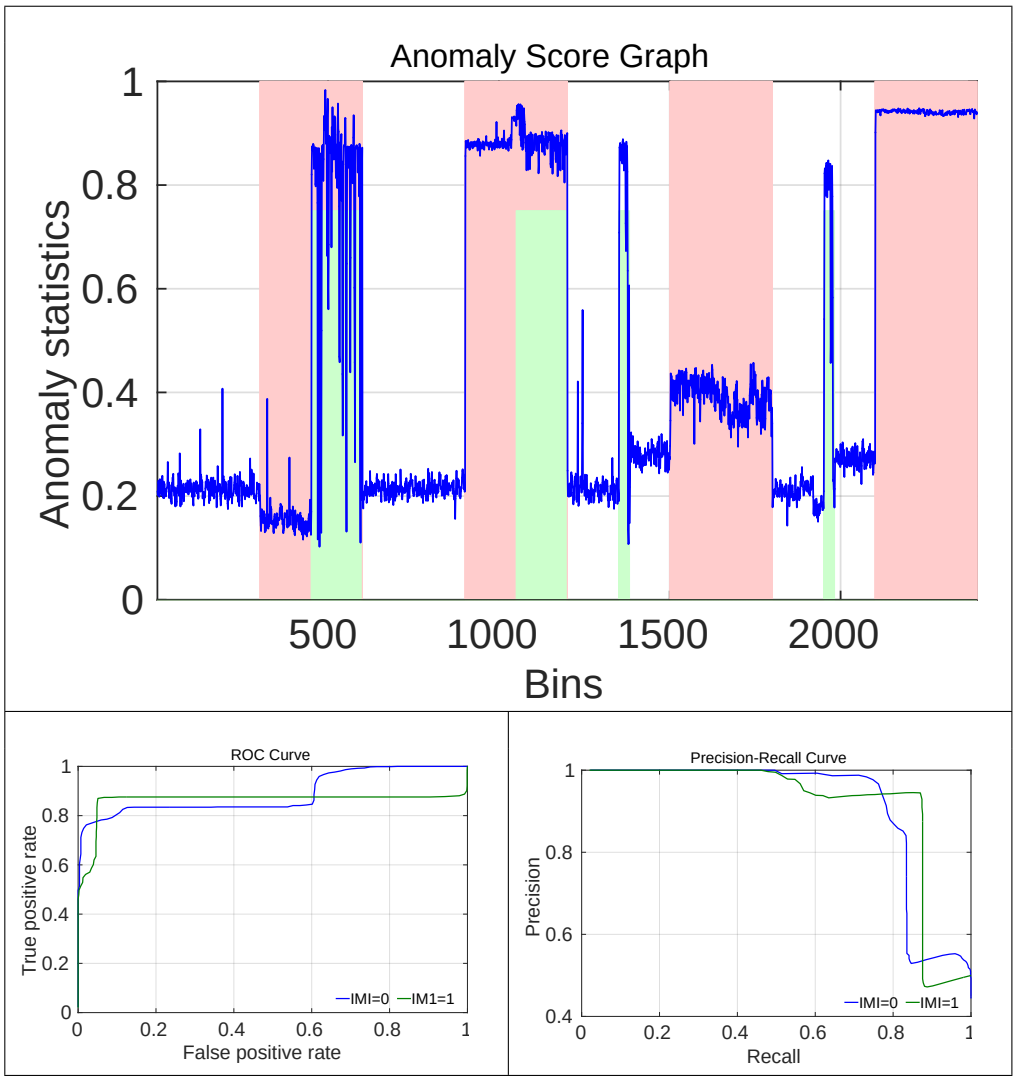


TABLE A.27: Anomaly Score Graph, Effect of migration on ROC/PRC for KM with combined dataset BC0-DoS-AL-MT0

KM with BC0-DoS-AL-MT0	Without migration	With migration
Error rate	0.1157	0.2101
Detection rate	88.43%	78.99%

TABLE A.28: Effect of migration on KM with BC0-DoS-AL-MT0

bottom-left, two ROCs are generated by partitioning the ASG labelled with anomaly and migration ground truths into those with $m = 0$ vs. $m = 1$. In the bottom-right, two PRCs are similarly generated. Table A.28 summarizes the error/detection rates with/without migration present.

A.4.3 Migration effect BC0-DoS-AL-MT0-NB

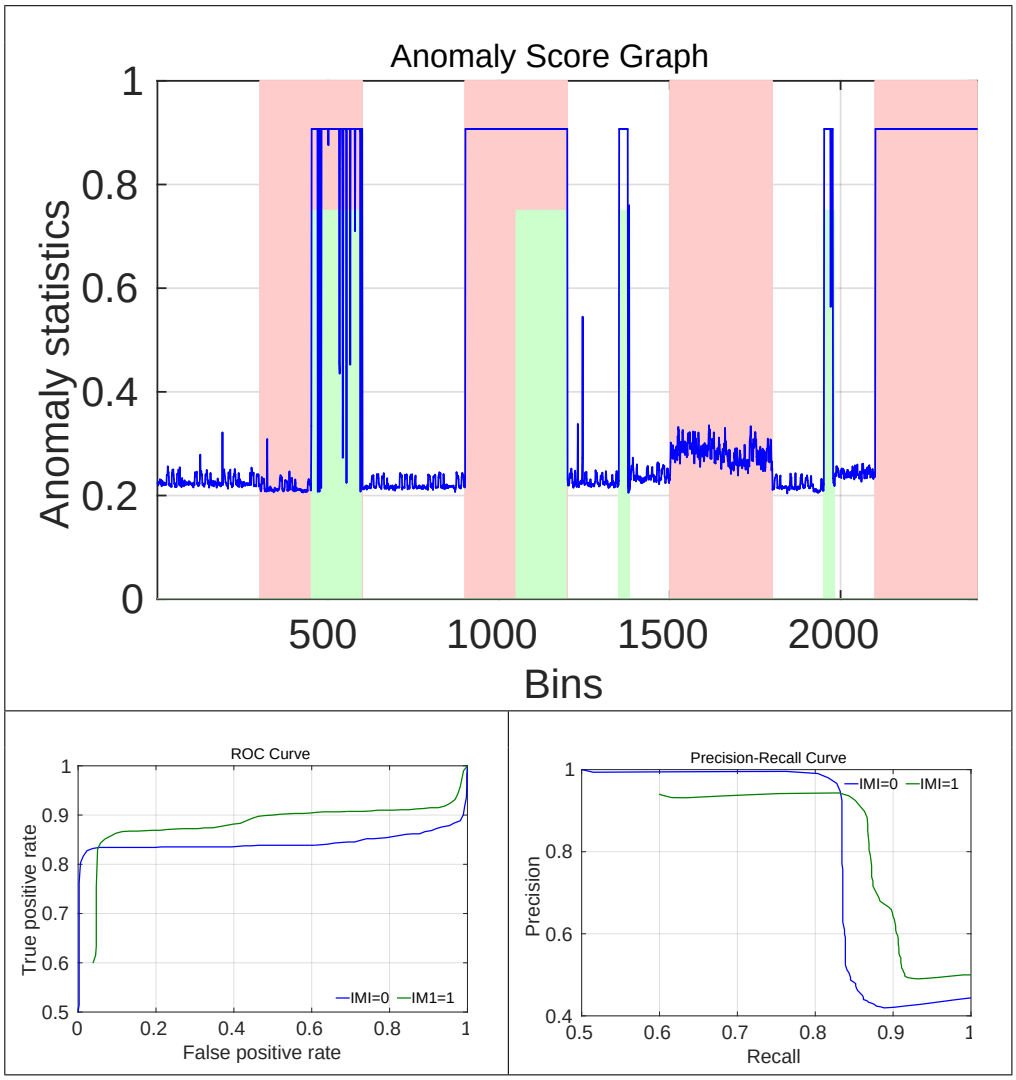


TABLE A.29: Anomaly Score Graph, Effect of migration on ROC/PRC for NB with combined dataset BC0-DoS-AL-MT0

Table A.29 summarizes anomaly detector NB when applied to dataset BC0-DoS-AL-MT0. The top half shows the Anomaly Score Graph (ASG). In the bottom-left, two ROCs are generated by partitioning the ASG labelled with anomaly and migration ground truths into those with $m = 0$ vs. $m = 1$. In the bottom-right, two PRCs are similarly generated. Table A.30 summarizes the error/detection rates with/without migration present.

NB with BC0-DoS-AL-MT0	Without migration	With migration
Error rate	0.0.0907	0.2143
Detection rate	90.93%	78.57%

TABLE A.30: Effect of migration on NB with BC0-DoS-AL-MT0

A.4.4 Migration effect BC0-DoS-AL-MT0-EMGM

Table A.31 summarizes anomaly detector EMGM when applied to dataset BC0-DoS-AL-MT0. The top half shows the Anomaly Score Graph (ASG). In the bottom-left, two ROCs are generated by partitioning the ASG labelled with anomaly and migration ground truths into those with $m = 0$ vs. $m = 1$. In the bottom-right, two PRCs are similarly generated. Table A.32 summarizes the error/detection rates with/without migration present.

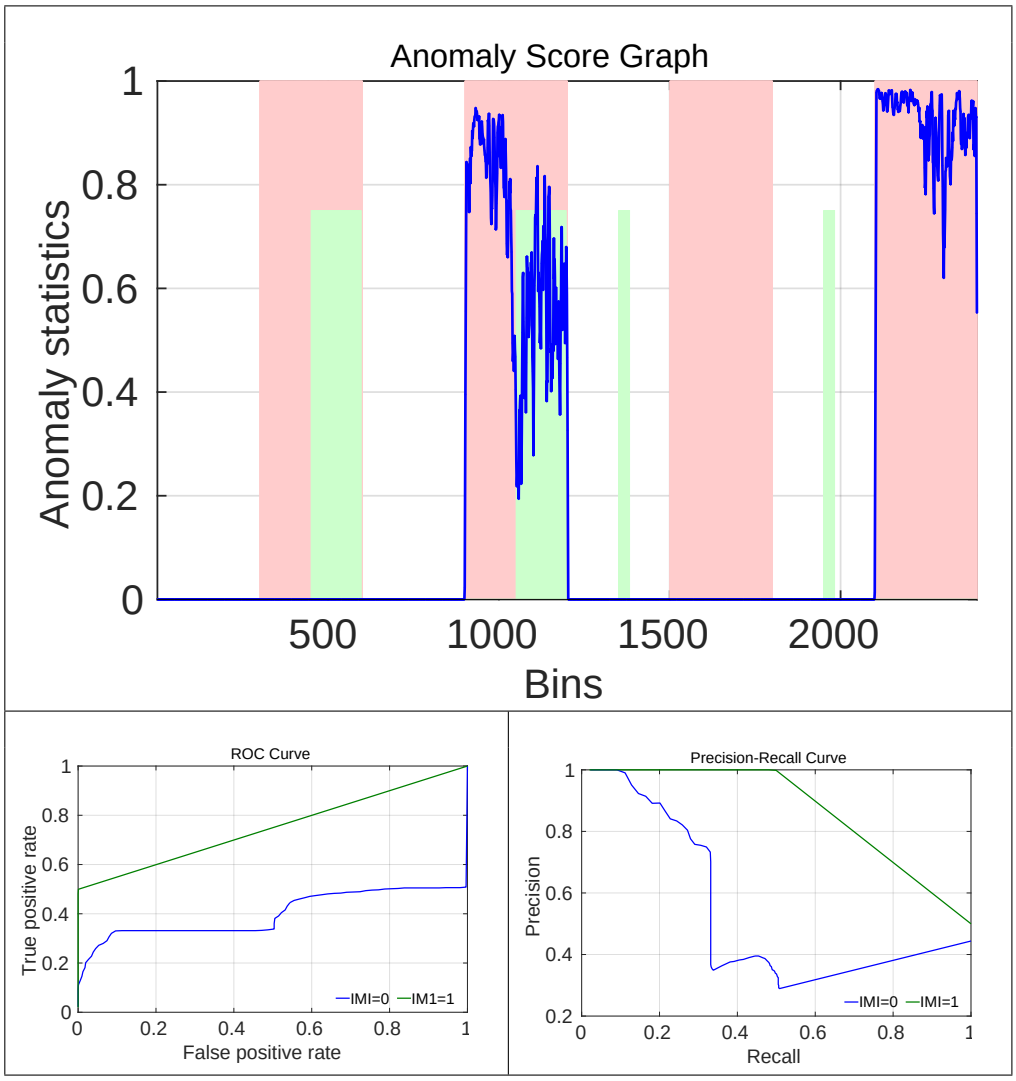


TABLE A.31: Anomaly Score Graph, Effect of migration on ROC/PRC for EMGM with combined dataset BC0-DoS-AL-MT0

EMGM with BC0-DoS-AL-MT0	Without migration	With migration
Error rate	0.3541	0.2676
Detection rate	64.59%	73.24%

TABLE A.32: Effect of migration on EMGM with BC0-DoS-AL-MT0

A.4.5 Combined dataset BC0-DoS-AH-MT1

A.4.6 Migration effect BC0-DoS-AH-MT1-PCA

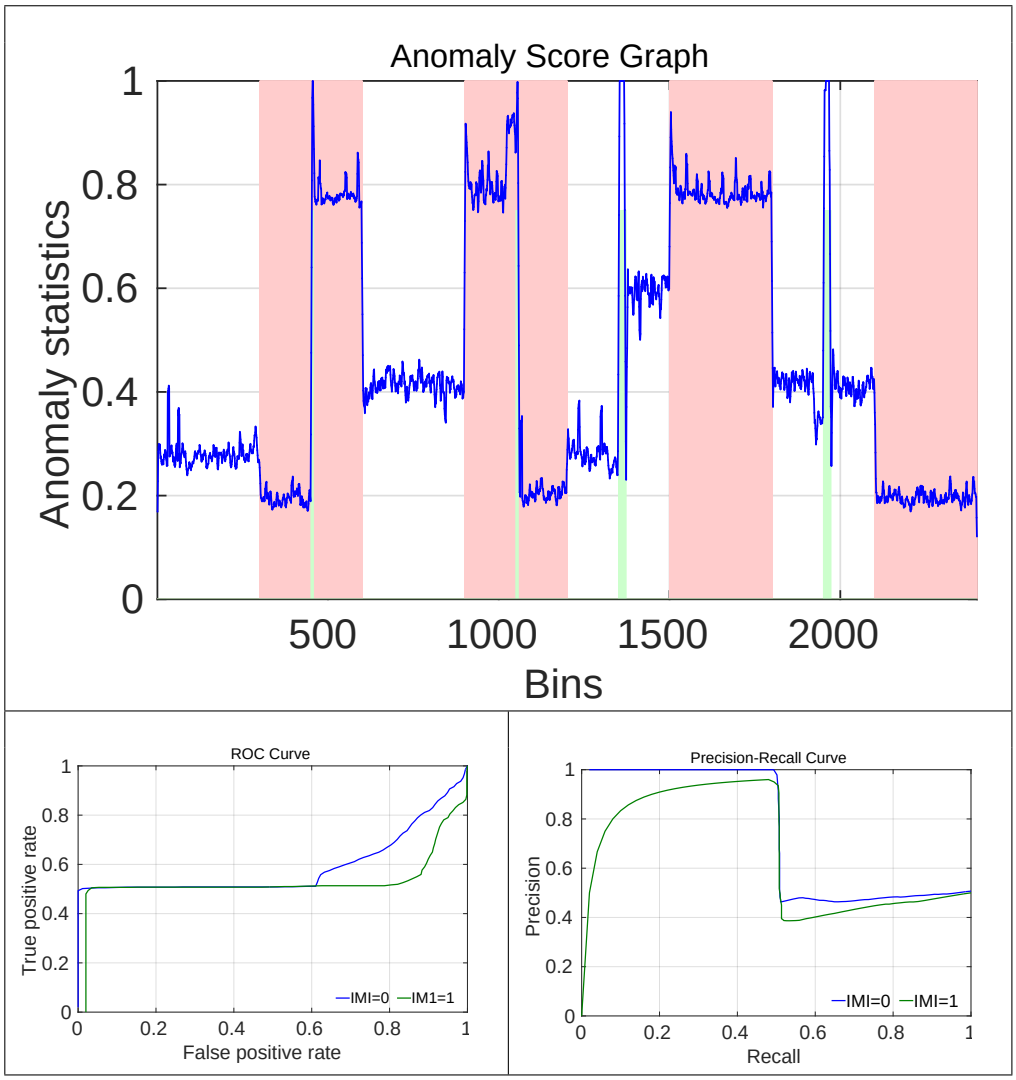


TABLE A.33: Anomaly Score Graph, Effect of migration on ROC/PRC for PCA with combined dataset BC0-DoS-AH-MT1

Table A.33 summarizes anomaly detector PCA when applied to dataset BC0-DoS-AH-MT1. The top half shows the Anomaly Score Graph (ASG). In the bottom-left, two ROCs are generated by partitioning the ASG labelled with anomaly and migration ground truths into those with $m = 0$ vs. $m = 1$. In the bottom-right, two PRCs are similarly generated. Table A.34 summarizes the error/detection rates with/without migration present.

A.4.7 Migration effect BC0-DoS-AH-MT1-KM

Table A.35 summarizes anomaly detector KM when applied to dataset BC0-DoS-AH-MT1. The top half shows the Anomaly Score Graph (ASG). In the

PCA with BC0-DoS-AH-MT1	Without migration	With migration
Error rate	0.2744	0.3179
Detection rate	72.56%	68.21%

TABLE A.34: Effect of migration on PCA with BC0-DoS-AH-MT1

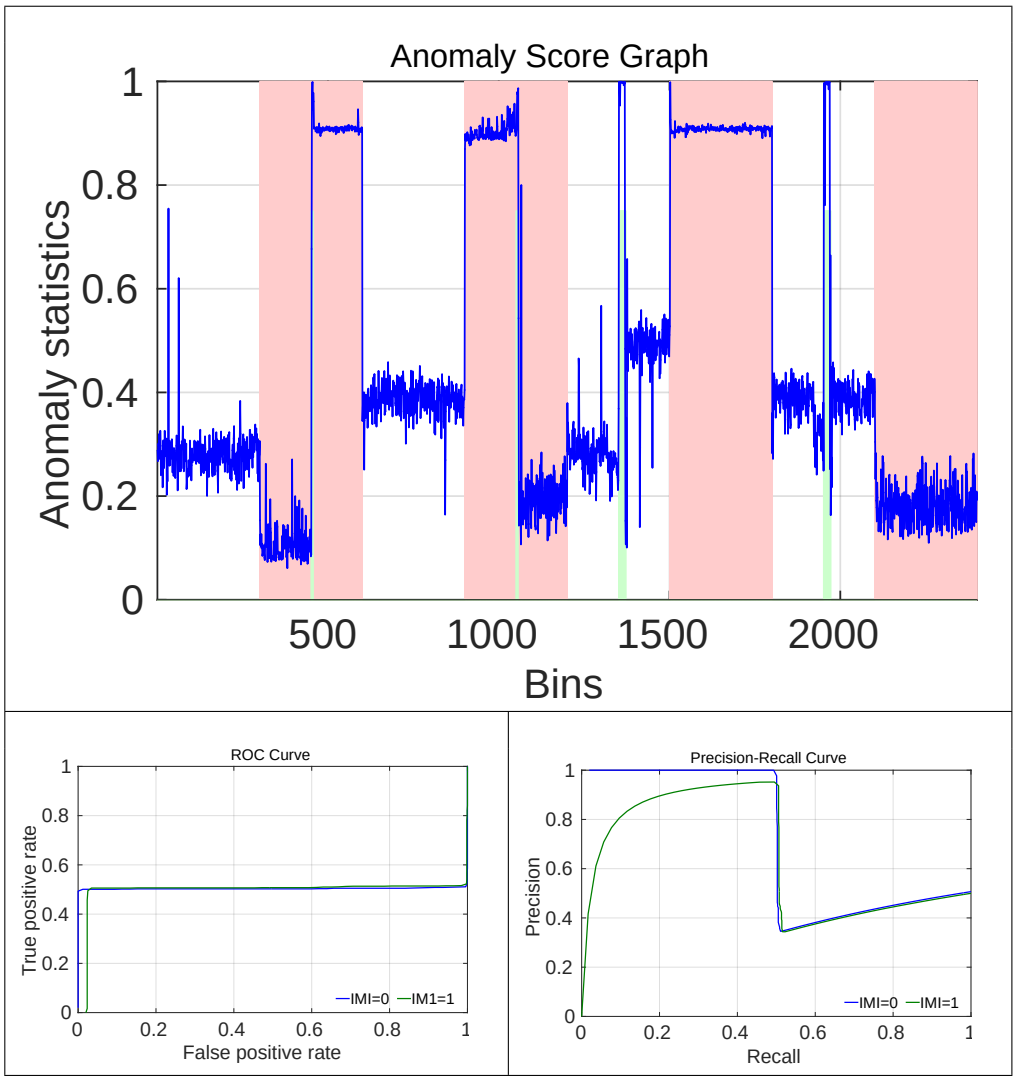


TABLE A.35: Anomaly Score Graph, Effect of migration on ROC/PRC for KM with combined dataset BC0-DoS-AH-MT1

KM with BC0-DoS-AH-MT1	Without migration	With migration
Error rate	0.5090	0.3150
Detection rate	49.1%	68.50%

TABLE A.36: Effect of migration on KM with BC0-DoS-AH-MT1

bottom-left, two ROCs are generated by partitioning the ASG labelled with anomaly and migration ground truths into those with $m = 0$ vs. $m = 1$. In the bottom-right, two PRCs are similarly generated. Table A.36 summarizes the error/detection rates with/without migration present.

A.4.8 Migration effect BC0-DoS-AH-MT1-NB

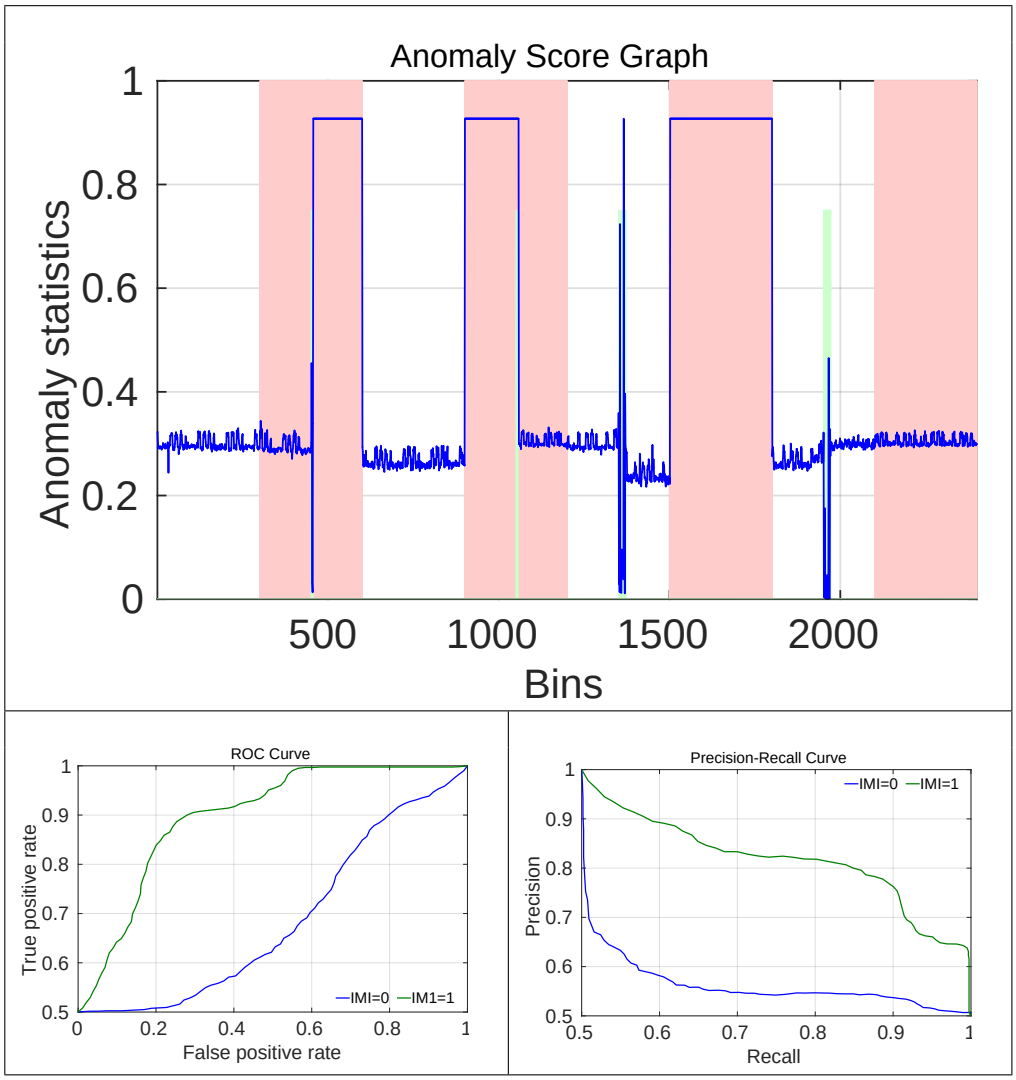


TABLE A.37: Anomaly Score Graph, Effect of migration on ROC/PRC for NB with combined dataset BC0-DoS-AH-MT1

Table A.37 summarizes anomaly detector NB when applied to dataset BC0-DoS-AH-MT1. The top half shows the Anomaly Score Graph (ASG). In the bottom-left, two ROCs are generated by partitioning the ASG labelled with anomaly and migration ground truths into those with $m = 0$ vs. $m = 1$. In the bottom-right, two PRCs are similarly generated. Table A.38 summarizes the error/detection rates with/without migration present.

NB with BC0-DoS-AH-MT1	Without migration	With migration
Error rate	0.2560	0.2504
Detection rate	74.4%	74.96%

TABLE A.38: Effect of migration on NB with BC0-DoS-AH-MT1

A.4.9 Migration effect BC0-DoS-AH-MT1-EMGM

Table A.39 summarizes anomaly detector EMGM when applied to dataset BC0-DoS-AH-MT1. The top half shows the Anomaly Score Graph (ASG). In the bottom-left, two ROCs are generated by partitioning the ASG labelled with anomaly and migration ground truths into those with $m = 0$ vs. $m = 1$. In the bottom-right, two PRCs are similarly generated. Table A.40 summarizes the error/detection rates with/without migration present.

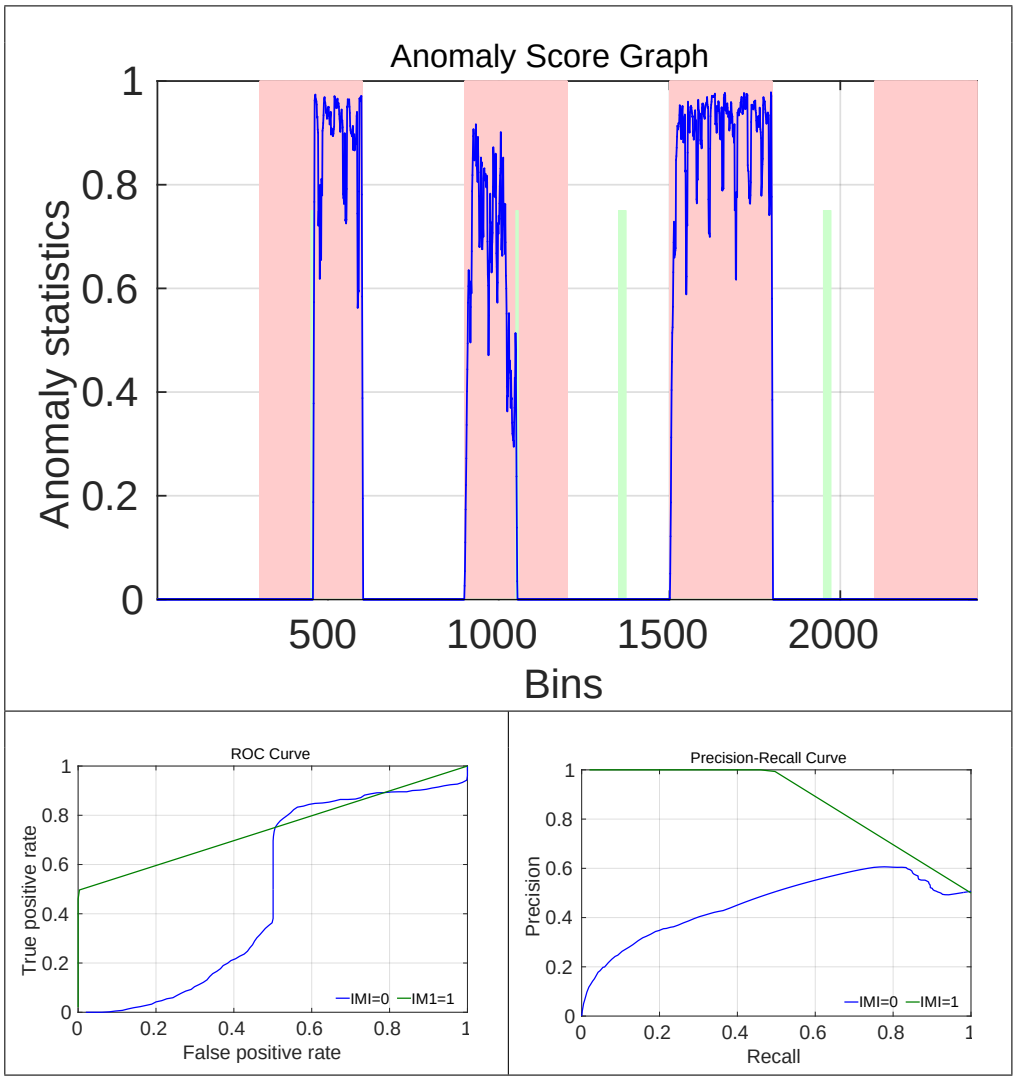


TABLE A.39: Anomaly Score Graph, Effect of migration on ROC/PRC for EMGM with combined dataset BC0-DoS-AH-MT1

EMGM with BC0-DoS-AH-MT1	Without migration	With migration
Error rate	0.5726	0.2679
Detection rate	42.74%	73.21%

TABLE A.40: Effect of migration on EMGM with BC0-DoS-AH-MT1

A.4.10 Combined dataset BC0-DoS-AH-MT0

A.4.11 Migration effect BC0-DoS-AH-MT0-PCA

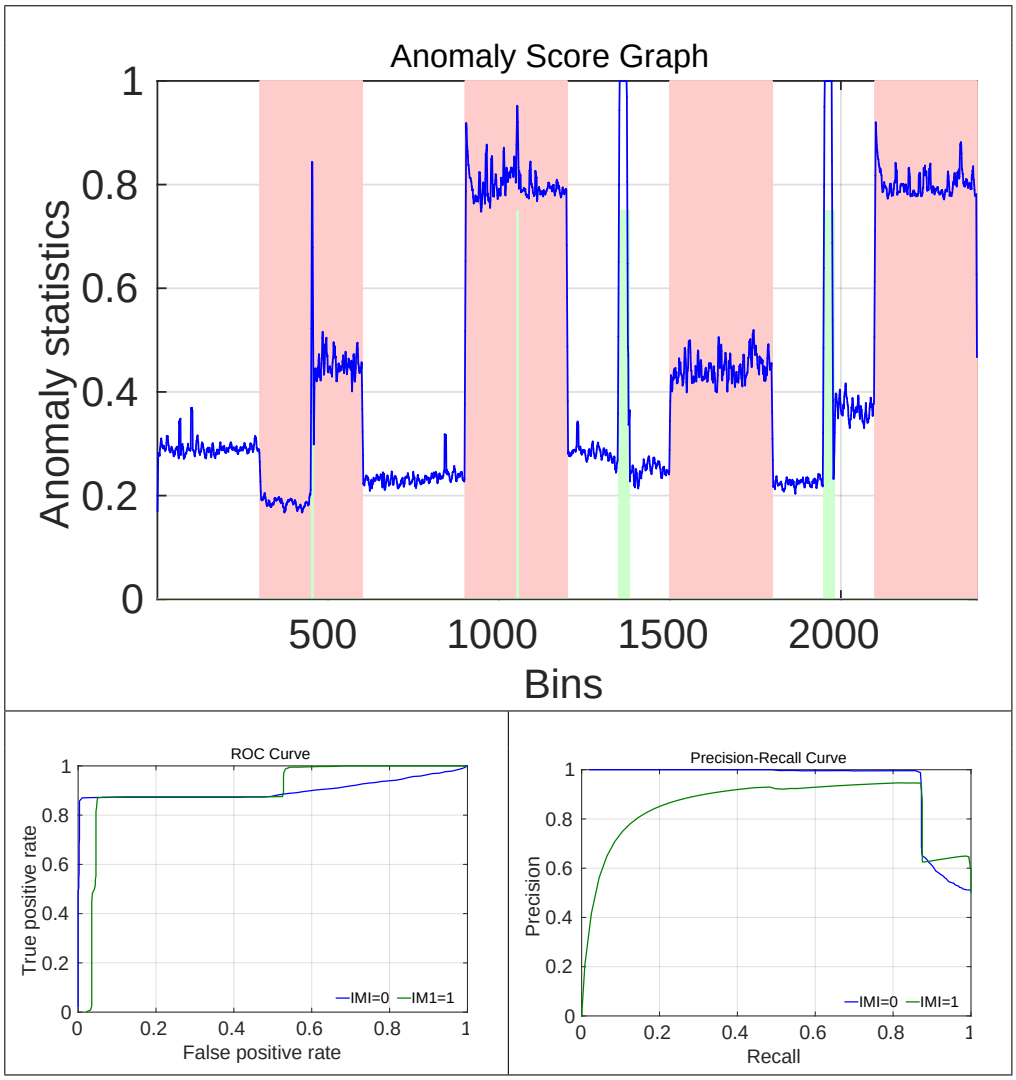


TABLE A.41: Anomaly Score Graph, Effect of migration on ROC/PRC for PCA with combined dataset BC0-DoS-AH-MT0

Table A.41 summarizes anomaly detector PCA when applied to dataset BC0-DoS-AH-MT0. The top half shows the Anomaly Score Graph (ASG). In the bottom-left, two ROCs are generated by partitioning the ASG labelled with anomaly and migration ground truths into those with $m = 0$ vs. $m = 1$. In the bottom-right, two PRCs are similarly generated. Table A.42 summarizes the error/detection rates with/without migration present.

A.4.12 Migration effect BC0-DoS-AH-MT0-KM

Table A.43 summarizes anomaly detector KM when applied to dataset BC0-DoS-AH-MT0. The top half shows the Anomaly Score Graph (ASG). In the

PCA with BC0-DoS-AH-MT0	Without migration	With migration
Error rate	0.0897	0.1247
Detection rate	91.03%	87.53%

TABLE A.42: Effect of migration on PCA with BC0-DoS-AH-MT0

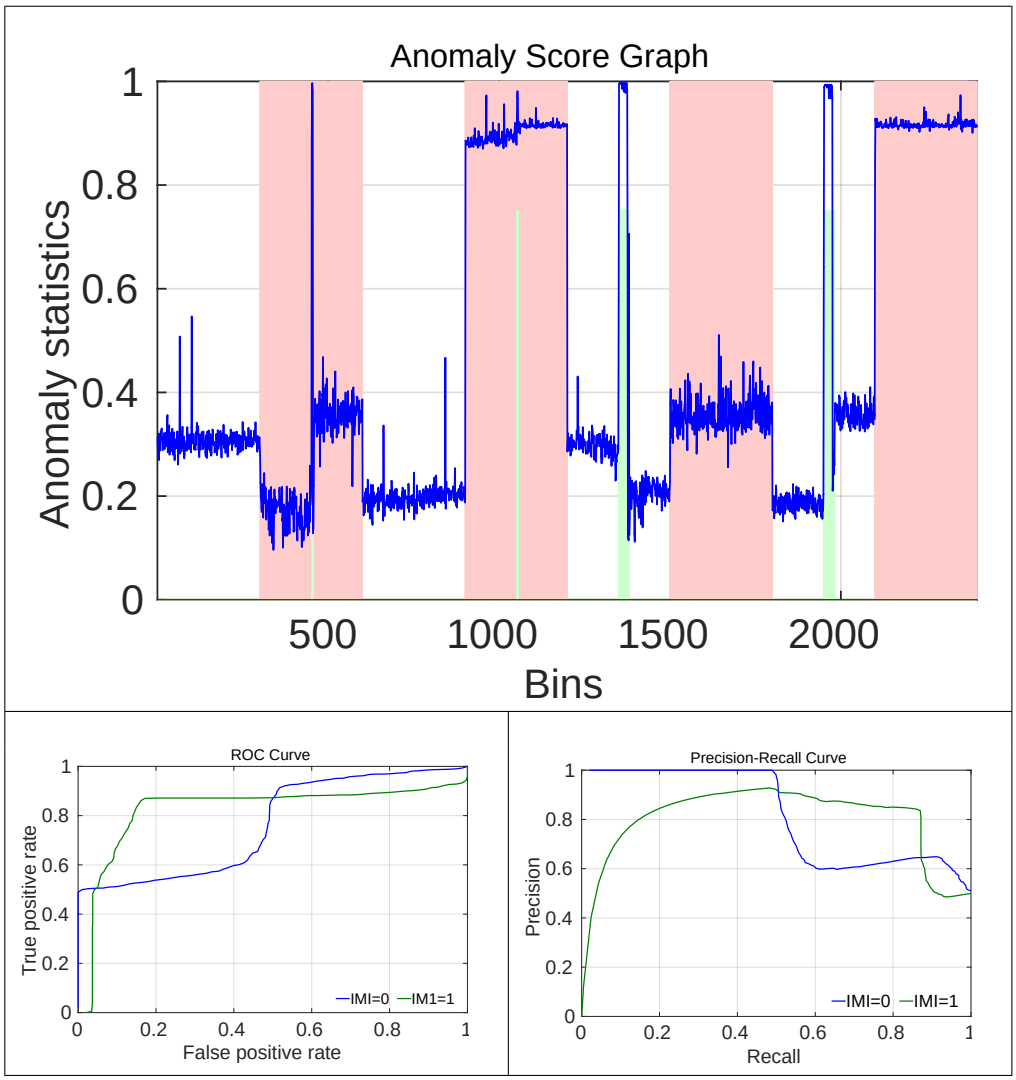


TABLE A.43: Anomaly Score Graph, Effect of migration on ROC/PRC for KM with combined dataset BC0-DoS-AH-MT0

KM with BC0-DoS-AH-MT0	Without migration	With migration
Error rate	0.2918	0.2719
Detection rate	70.82%	72.81%

TABLE A.44: Effect of migration on KM with BC0-DoS-AH-MT0

bottom-left, two ROCs are generated by partitioning the ASG labelled with anomaly and migration ground truths into those with $m = 0$ vs. $m = 1$. In the bottom-right, two PRCs are similarly generated. Table A.44 summarizes the error/detection rates with/without migration present.

A.4.13 Migration effect BC0-DoS-AH-MT0-NB

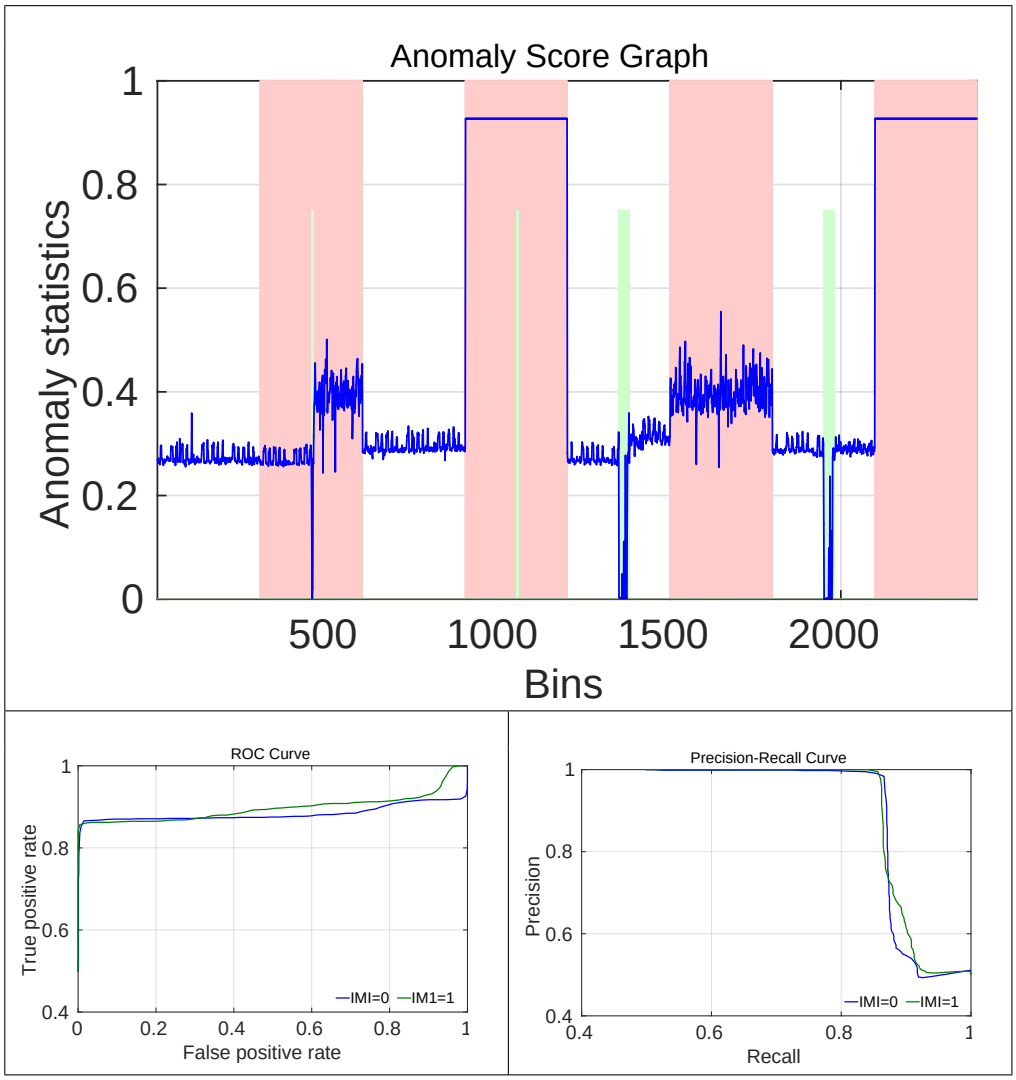


TABLE A.45: Anomaly Score Graph, Effect of migration on ROC/PRC for NB with combined dataset BC0-DoS-AH-MT0

Table A.45 summarizes anomaly detector NB when applied to dataset BC0-DoS-AH-MT0. The top half shows the Anomaly Score Graph (ASG). In the bottom-left, two ROCs are generated by partitioning the ASG labelled with anomaly and migration ground truths into those with $m = 0$ vs. $m = 1$. In the bottom-right, two PRCs are similarly generated. Table A.46 summarizes the error/detection rates with/without migration present.

NB with BC0-DoS-AH-MT0	Without migration	With migration
Error rate	0.0793	0.2389
Detection rate	92.07%	76.11%

TABLE A.46: Effect of migration on NB with BC0-DoS-AH-MT0

A.4.14 Migration effect BC0-DoS-AH-MT0-EMGM

Table A.47 summarizes anomaly detector EMGM when applied to dataset BC0-DoS-AH-MT0. The top half shows the Anomaly Score Graph (ASG). In the bottom-left, two ROCs are generated by partitioning the ASG labelled with anomaly and migration ground truths into those with $m = 0$ vs. $m = 1$. In the bottom-right, two PRCs are similarly generated. Table A.48 summarizes the error/detection rates with/without migration present.

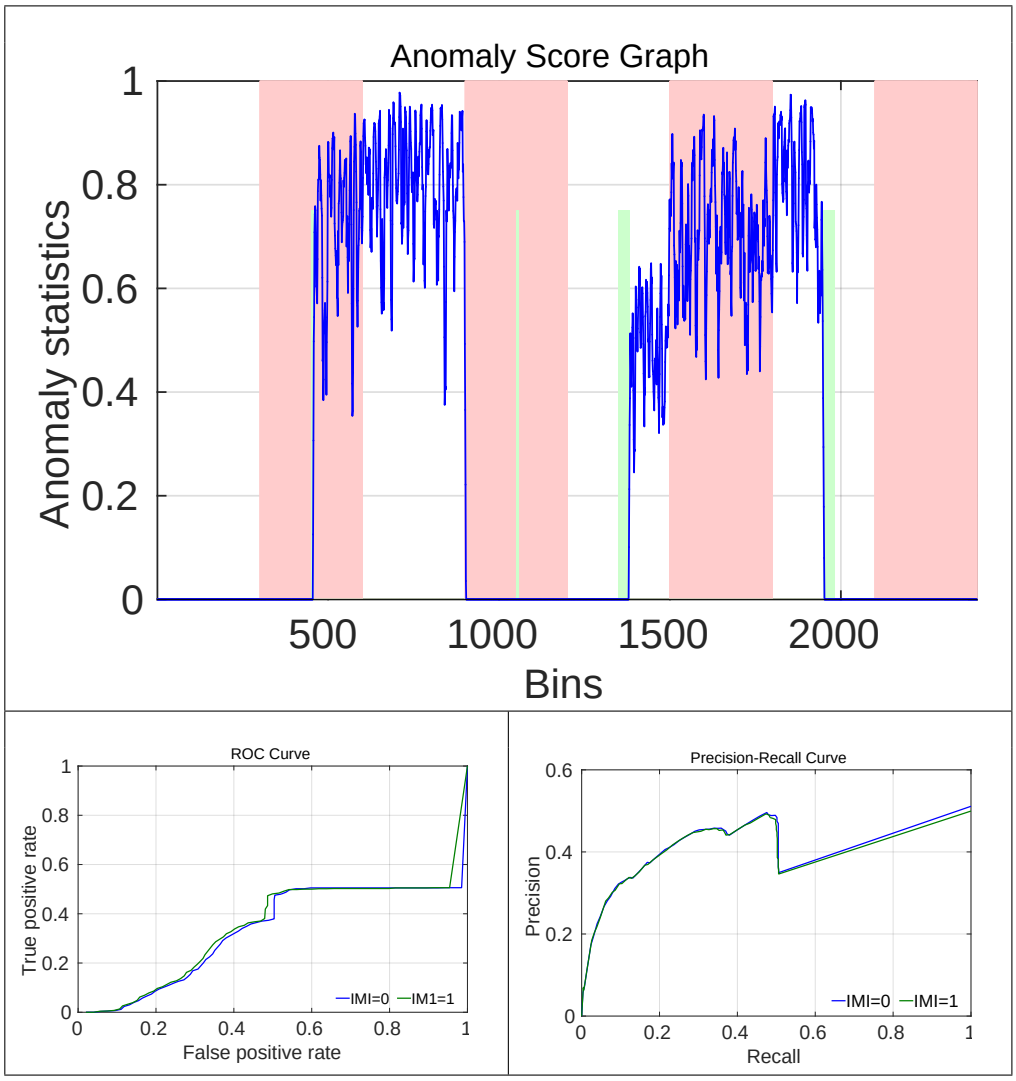


TABLE A.47: Anomaly Score Graph, Effect of migration on ROC/PRC for EMGM with combined dataset BC0-DoS-AH-MT0

EMGM with BC0-DoS-AH-MT0	Without migration	With migration
Error rate	0.5556	0.5371
Detection rate	44.44%	46.29%

TABLE A.48: Effect of migration on EMGM with BC0-DoS-AH-MT0

A.4.15 Combined dataset BC0-NS-AH

A.4.16 Migration effect BC0-NS-AH-PCA

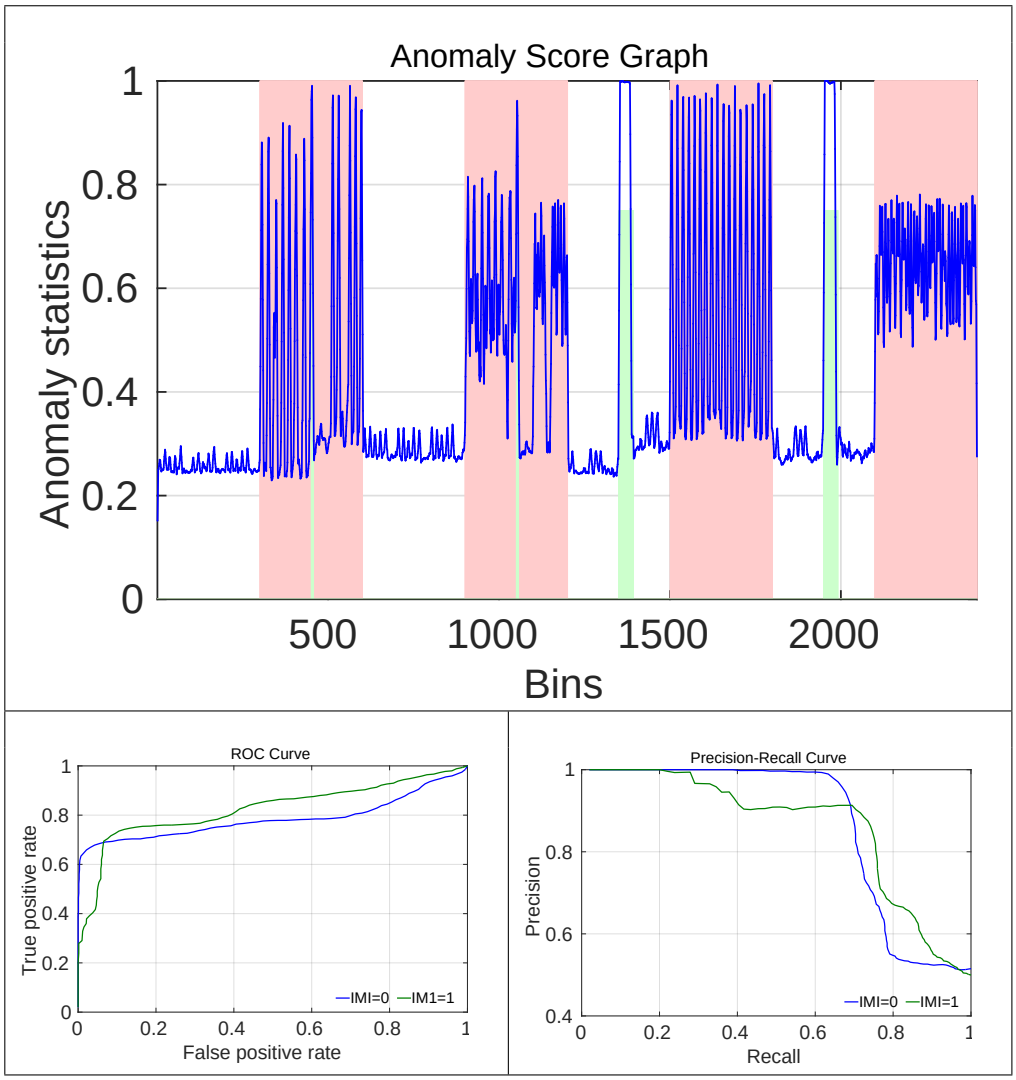


TABLE A.49: Anomaly Score Graph, Effect of migration on ROC/PRC for PCA with combined dataset BC0-NS-AH

PCA with BC0-NS-AH	Without migration	With migration
Error rate	0.3129	0.2840
Detection rate	68.71%	71.6%

TABLE A.50: Effect of migration on PCA with BC0-NS-AH

Table A.49 summarizes anomaly detector PCA when applied to dataset BC0-NS-AH. The top half shows the Anomaly Score Graph (ASG). In the bottom-left, two ROCs are generated by partitioning the ASG labelled with anomaly and migration ground truths into those with $m = 0$ vs. $m = 1$. In the bottom-right, two PRCs are similarly generated. Table A.50 summarizes the error/detection rates with/without migration present.

A.4.17 Migration effect BC0-NS-AH-KM

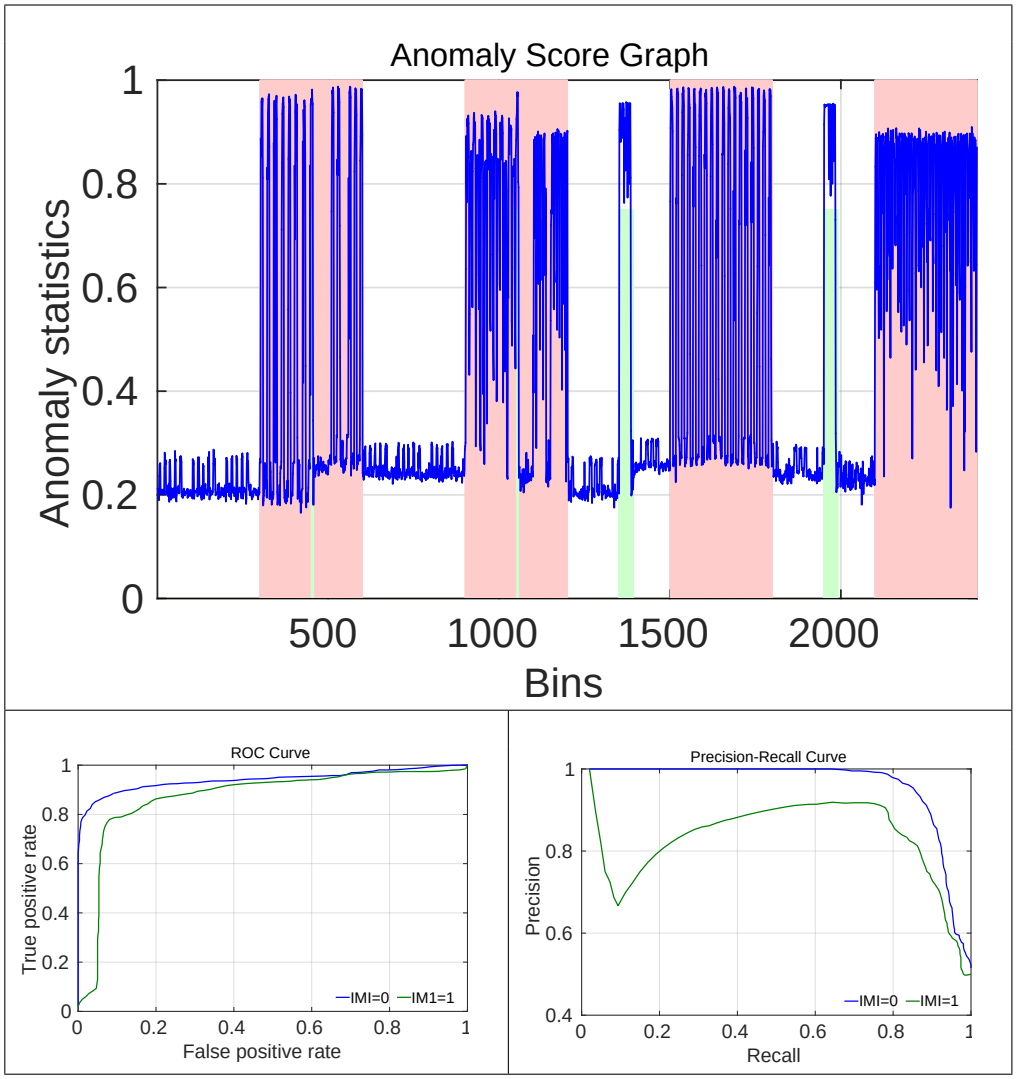


TABLE A.51: Anomaly Score Graph, Effect of migration on ROC/PRC for KM with combined dataset BC0-NS-AH

KM with BC0-NS-AH	Without migration	With migration
Error rate	0.1384	0.1843
Detection rate	86.16%	81.57%

TABLE A.52: Effect of migration on KM with BC0-NS-AH

Table A.51 summarizes anomaly detector KM when applied to dataset BC0-NS-AH. The top half shows the Anomaly Score Graph (ASG). In the bottom-left, two ROCs are generated by partitioning the ASG labelled with anomaly and migration ground truths into those with $m = 0$ vs. $m = 1$. In the bottom-right, two PRCs are similarly generated. Table A.52 summarizes the error/detection rates with/without migration present.

A.4.18 Migration effect BC0-NS-AH-NB

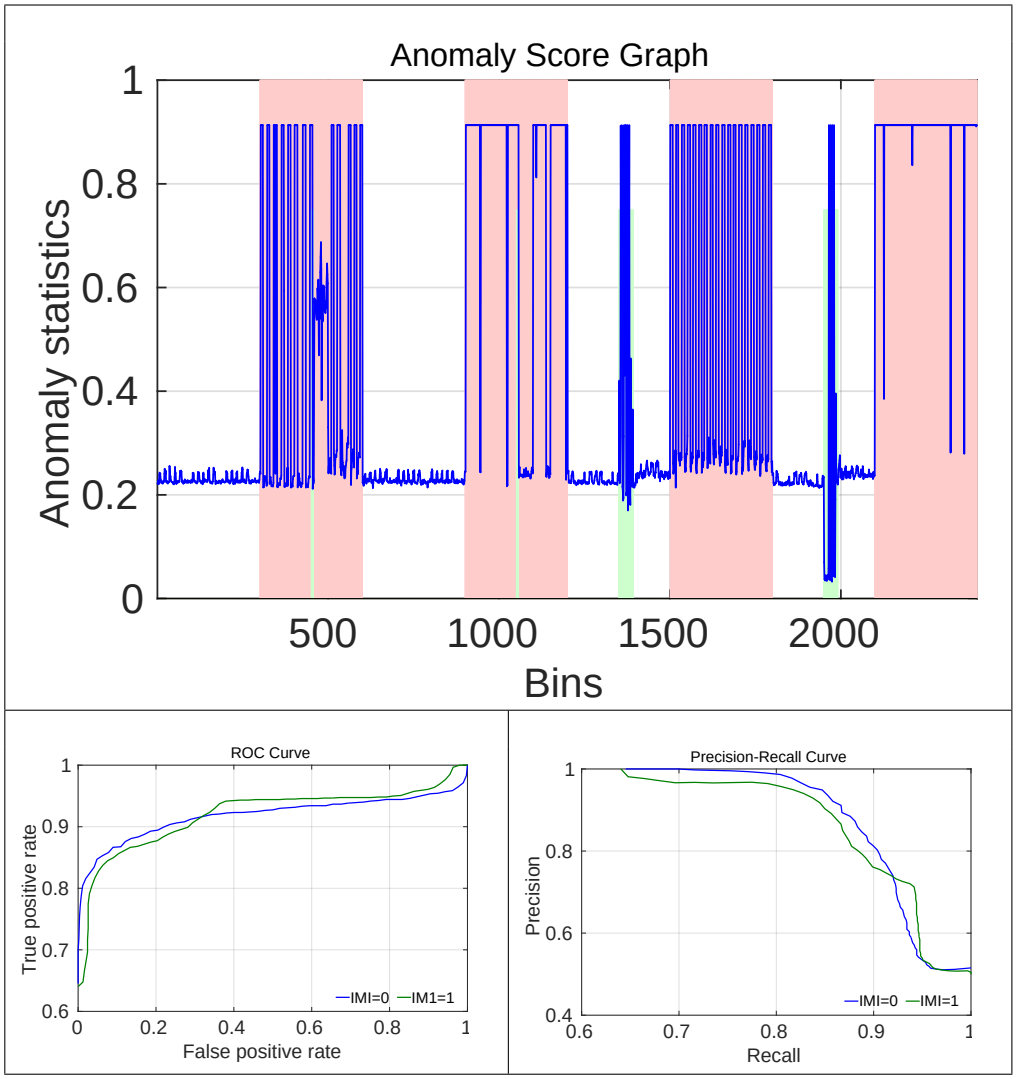


TABLE A.53: Anomaly Score Graph, Effect of migration on ROC/PRC for NB with combined dataset BC0-NS-AH

NB with BC0-NS-AH	Without migration	With migration
Error rate	0.1484	0.1689
Detection rate	85.16%	83.11%

TABLE A.54: Effect of migration on NB with BC0-NS-AH

Table A.53 summarizes anomaly detector NB when applied to dataset BC0-NS-AH. The top half shows the Anomaly Score Graph (ASG). In the bottom-left, two ROCs are generated by partitioning the ASG labelled with anomaly and migration ground truths into those with $m = 0$ vs. $m = 1$. In the bottom-right, two PRCs are similarly generated. Table A.54 summarizes the error/detection rates with/without migration present.

A.4.19 Migration effect BC0-NS-AH-EMGM

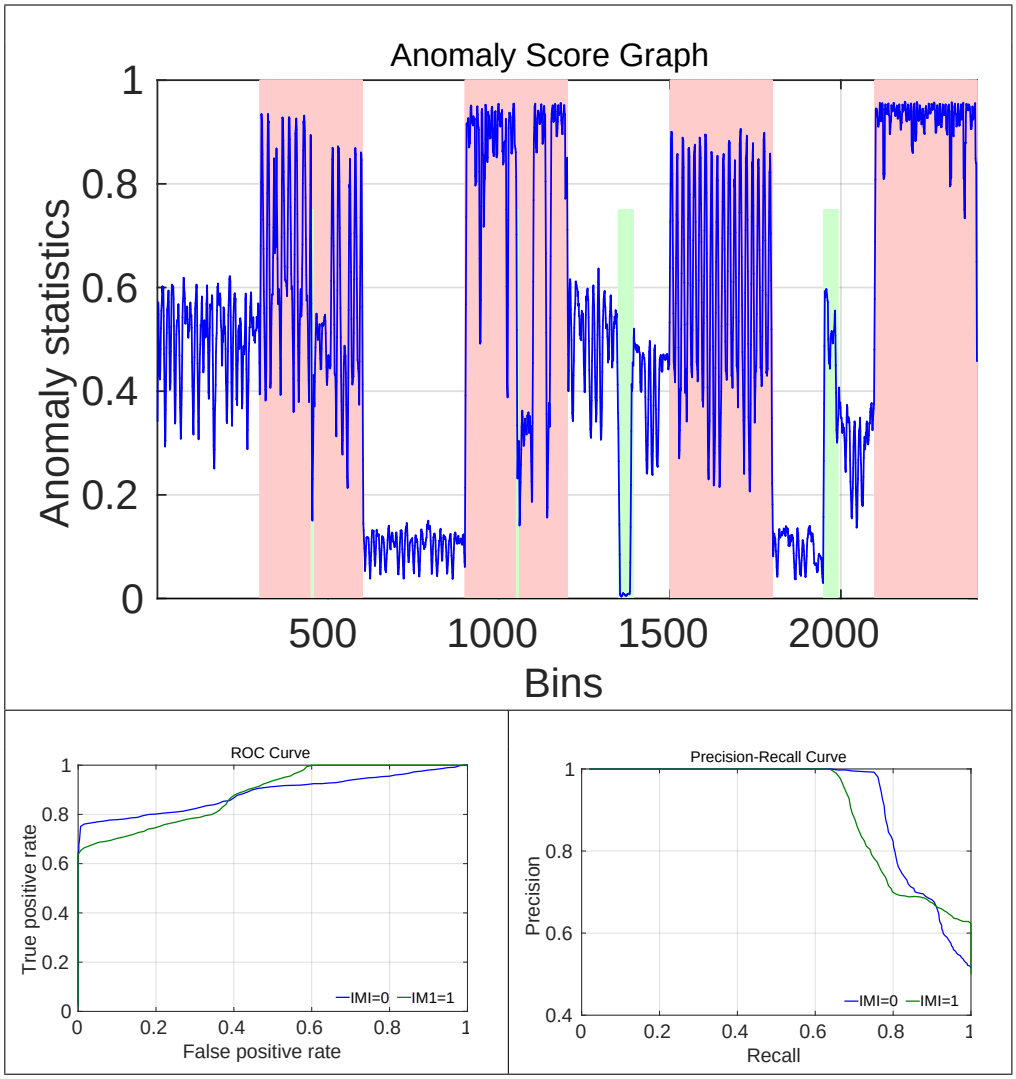


TABLE A.55: Anomaly Score Graph, Effect of migration on ROC/PRC for EMGM with combined dataset BC0-NS-AH

EMGM with BC0-NS-AH	Without migration	With migration
Error rate	0.1340	0.2744
Detection rate	86.6%	72.56%

TABLE A.56: Effect of migration on EMGM with BC0-NS-AH

Table A.55 summarizes anomaly detector EMGM when applied to dataset BC0-NS-AH. The top half shows the Anomaly Score Graph (ASG). In the bottom-left, two ROCs are generated by partitioning the ASG labelled with anomaly and migration ground truths into those with $m = 0$ vs. $m = 1$. In the bottom-right, two PRCs are similarly generated. Table A.56 summarizes the error/detection rates with/without migration present.

A.4.20 Combined dataset BC0-NS-AL

A.4.21 Migration effect BC0-NS-AL-PCA

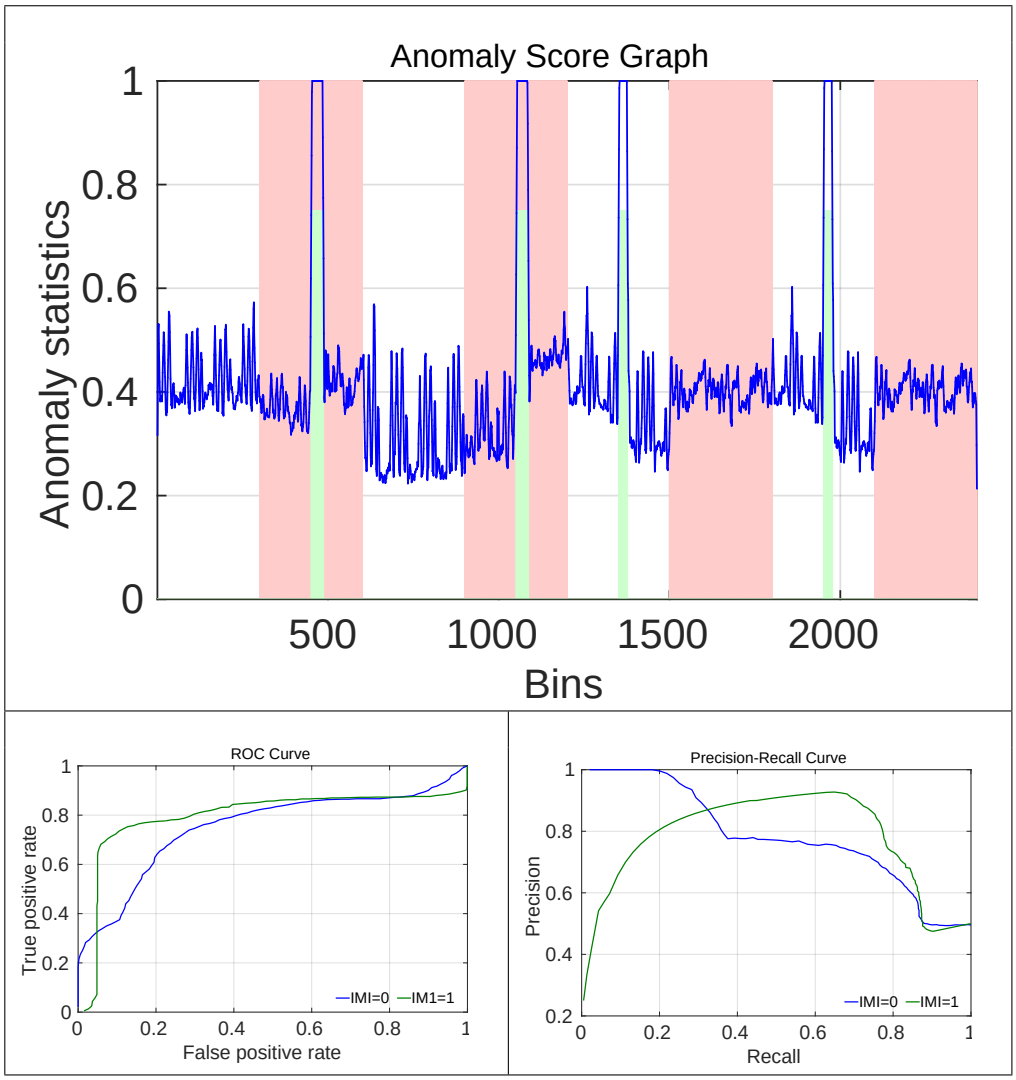


TABLE A.57: Anomaly Score Graph, Effect of migration on ROC/PRC for PCA with combined dataset BC0-NS-AL

PCA with BC0-NS-AL	Without migration	With migration
Error rate	0.3708	0.2292
Detection rate	62.92%	77.08%

TABLE A.58: Effect of migration on PCA with BC0-NS-AL

Table A.57 summarizes anomaly detector PCA when applied to dataset BC0-NS-AL. The top half shows the Anomaly Score Graph (ASG). In the bottom-left, two ROCs are generated by partitioning the ASG labelled with anomaly and migration ground truths into those with $m = 0$ vs. $m = 1$. In the bottom-right, two PRCs are similarly generated. Table A.58 summarizes the error/detection rates with/without migration present.

A.4.22 Migration effect BC0-NS-AL-KM

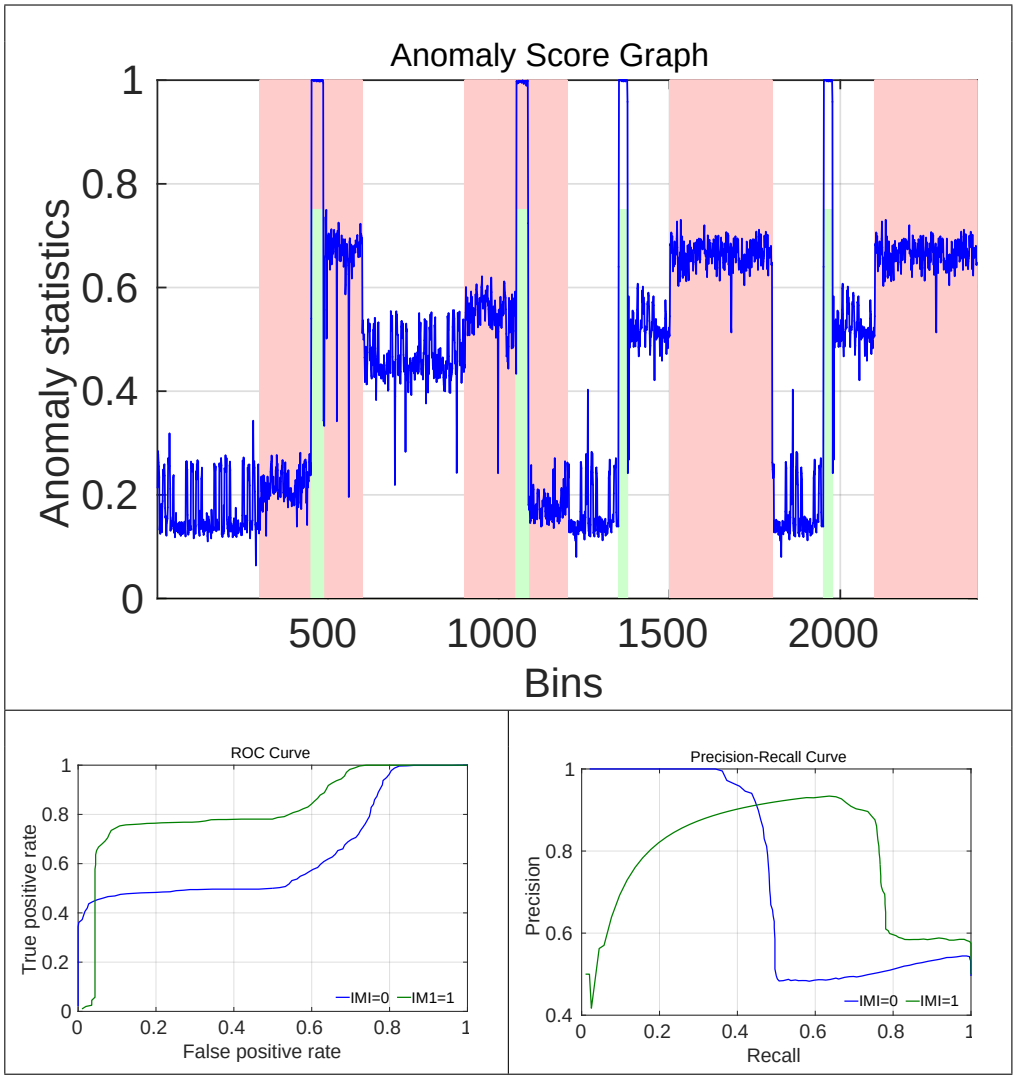


TABLE A.59: Anomaly Score Graph, Effect of migration on ROC/PRC for KM with combined dataset BC0-NS-AL

KM with BC0-NS-AL	Without migration	With migration
Error rate	0.5018	0.3079
Detection rate	49.82%	69.21%

TABLE A.60: Effect of migration on KM with BC0-NS-AL

Table A.59 summarizes anomaly detector KM when applied to dataset BC0-NS-AL. The top half shows the Anomaly Score Graph (ASG). In the bottom-left, two ROCs are generated by partitioning the ASG labelled with anomaly and migration ground truths into those with $m = 0$ vs. $m = 1$. In the bottom-right, two PRCs are similarly generated. Table A.60 summarizes the error/detection rates with/without migration present.

A.4.23 Migration effect BC0-NS-AL-NB

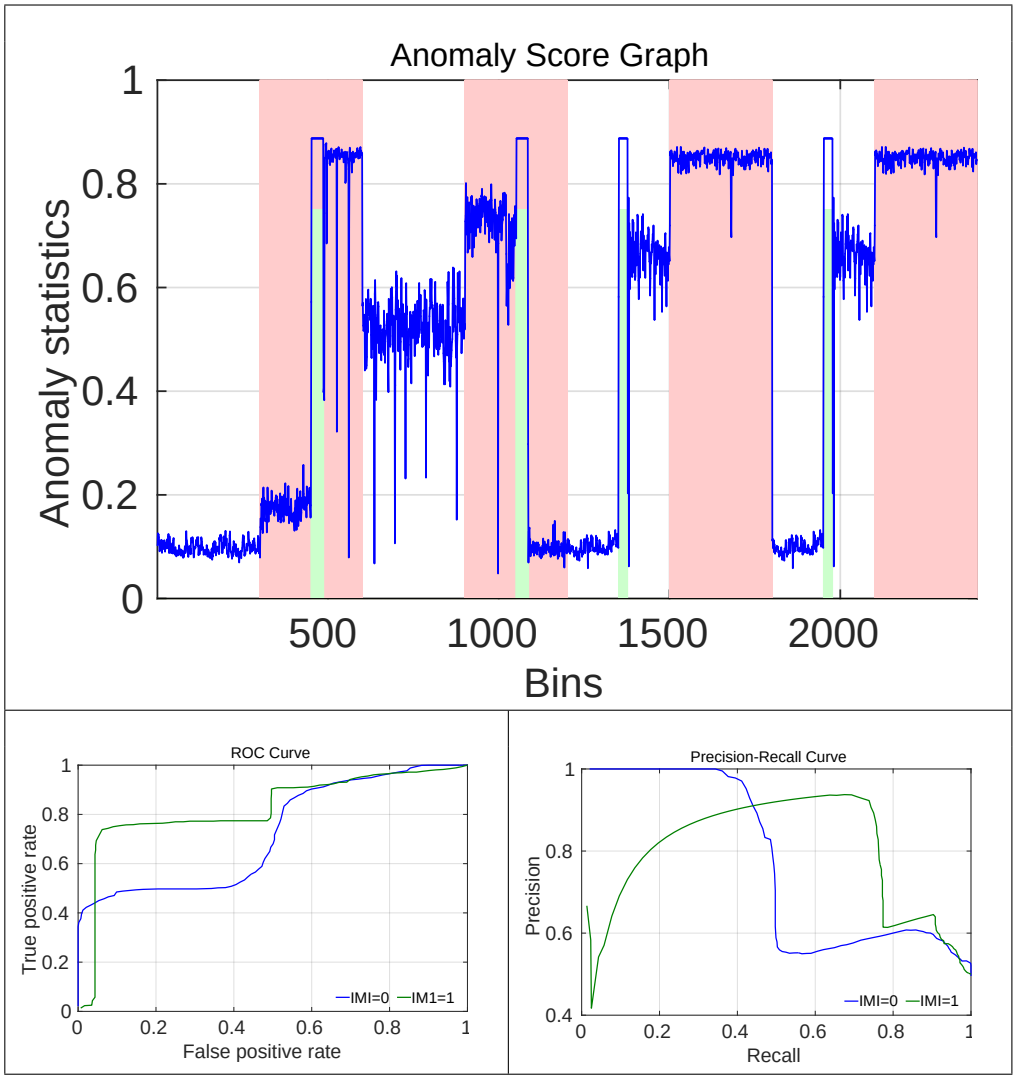


TABLE A.61: Anomaly Score Graph, Effect of migration on ROC/PRC for NB with combined dataset BC0-NS-AL

NB with BC0-NS-AL	Without migration	With migration
Error rate	0.4453	0.3525
Detection rate	55.47%	64.75%

TABLE A.62: Effect of migration on NB with BC0-NS-AL

Table A.61 summarizes anomaly detector NB when applied to dataset BC0-NS-AL. The top half shows the Anomaly Score Graph (ASG). In the bottom-left, two ROCs are generated by partitioning the ASG labelled with anomaly and migration ground truths into those with $m = 0$ vs. $m = 1$. In the bottom-right, two PRCs are similarly generated. Table A.62 summarizes the error/detection rates with/without migration present.

A.4.24 Migration effect BC0-NS-AL-EMGM

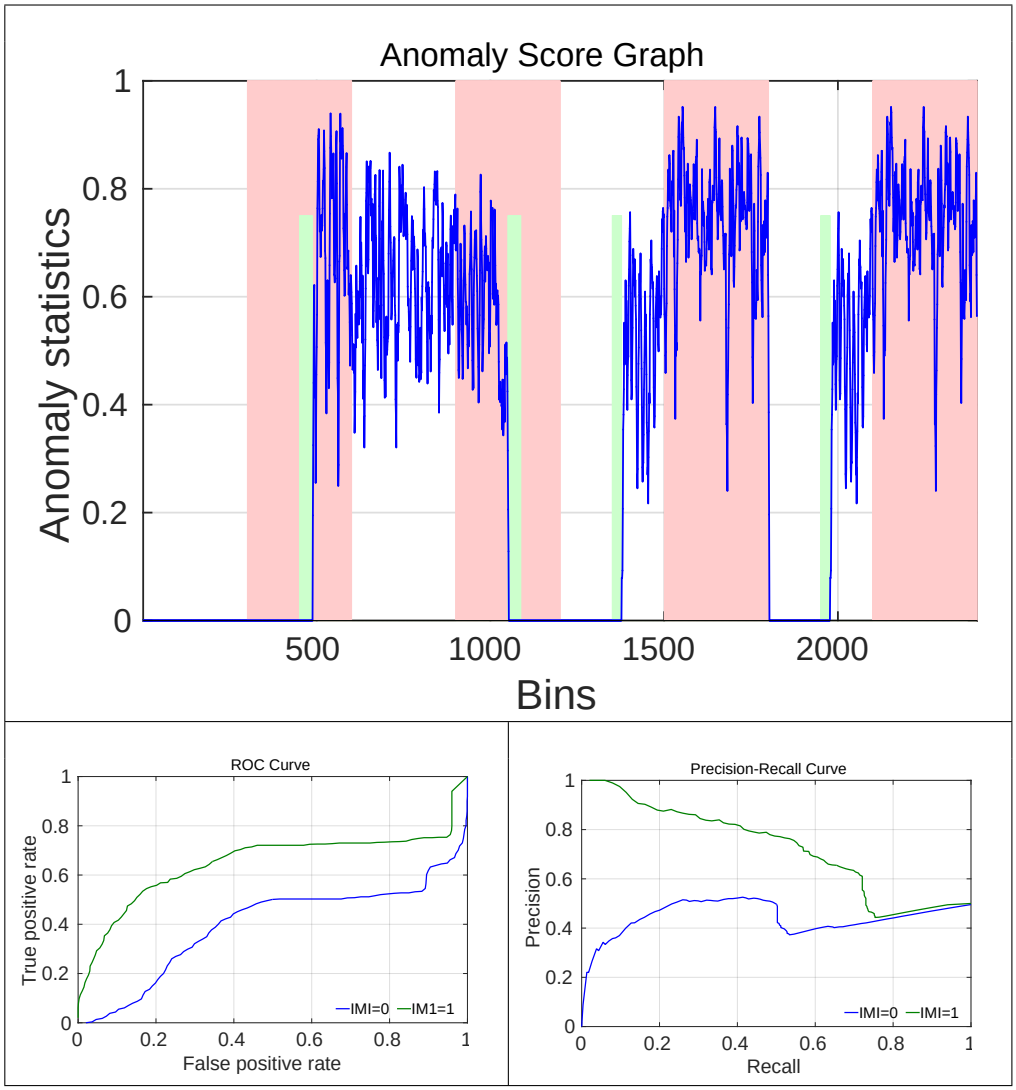


TABLE A.63: Anomaly Score Graph, Effect of migration on ROC/PRC for EMGM with combined dataset BC0-NS-AL

Table A.63 summarizes anomaly detector EMGM when applied to dataset BC0-NS-AL. The top half shows the Anomaly Score Graph (ASG). In the bottom-left, two ROCs are generated by partitioning the ASG labelled with anomaly and migration ground truths into those with $m = 0$ vs. $m = 1$. In the bottom-right, two PRCs are similarly generated. Table A.64 summarizes the error/detection rates with/without migration present.

EMGM with BC0-NS-AL	Without migration	With migration
Error rate	0.3504	0.3504
Detection rate	51.19%	64.96%

TABLE A.64: Effect of migration on EMGM with BC0-NS-AL

Appendix B

Monitoring metrics

ADaaS collects various type of metrics for anomaly detection. These metrics have different measurement levels. Table B.1 list the set of system, application and network level metrics monitored by *ADaaS* using agents. For the experimental evaluation Linux *Sar* utility is used to collect the VM level system metrics.

TABLE B.1: Monitoring metrics

Metrics Type	**Description**	**Measurement Level**	**Scenario**
dwbr	Total amount of data written to disk in blocks per second	VM,Host	Resource Hog
drbr	Total amount of data read from disk in blocks per second	VM,Host	Resource Hog
dwr	Number of sectors written to the disk (each sector 512 bytes)	VM,Host	Resource Hog
drr	Number of sectors read to the disk (each sector 512 bytes)	VM,Host	Resource Hog
memutil	% of memory used at particular time	VM, Host	Resource Hog, API Exhaustion
memcommit	% of physical host memory needed for current workload in relation to total amount of memory	VM, Host	Resource Hog/API Exhaustion/VM Resizing
mbfused	Total amount of space used in megabytes	VM, Host	Resource Hog / API Exhaustion
cpuuser	CPU time in user space	VM, Host	Resource Hog/API Exhaustion
cpusystem	CPU time in kernel space	VM, Host	Resource Hog/API Exhaustion
mngtbyteCnt	Total number of packets (bytes) for management network	Network	API Exhaustion/VM Resizing
mngtpktsCnt	Total number of packet traversing the management network at particular time	Network	API Exhaustion/VM Resizing
mngtfr	Flow rate of management network	Network	API Exhaustion/VM Resizing
nwIntRXr	Network utilisation rate of inbound network interface of the controller node	Network	API Exhaustion
nwIntTXr	Network utilisation rate of outbound interface of the controller node	network	API Exhaustion
bytecnt	Total number of packets in byte for data network for each bin	Network	VM Resizing/network anomalies
pktcnt	Total number of packets for the data network for each bin	Network	VM Resizing/network anomalies
activeflows	Number of active flows in each bin	Network	Network anomalies
srcIPaddrdist	Entropy of source IP address distribution	Network	Network anomalies
dstIPadddist	Entropy of destination IP address distribution	Network	Network anomalies
srcPrtdist	Entropy of source port distribution	Network	Network anomalies
dstPrtdist	Entropy of destination port distribution	Network	Network anomalies
pktSizedist	Entropy of packet size distribution	Network	Network anomalies
dbread	Number of reads issued on file	Application	Database growth
dbwrite	Number of writes issued on file	Application	Database growth
iostallread	Total time (ms),that users waited for reads issued on file	Application	Database growth
numwrites	Number of writes made on file	Application	Database growth
numbytes	Number of bytes written on file	Application	Database growth
iostallwrite	Total time (ms), that user waited for writes to be completed	Application	Database growth
iostall	Total time (ms), that users waited for I/O to be completed	Application	Database growth
sizeondisk	Number of bytes used on the disk for this file	Application	Database growth

Appendix C

Screenshots of running *ADaaS*

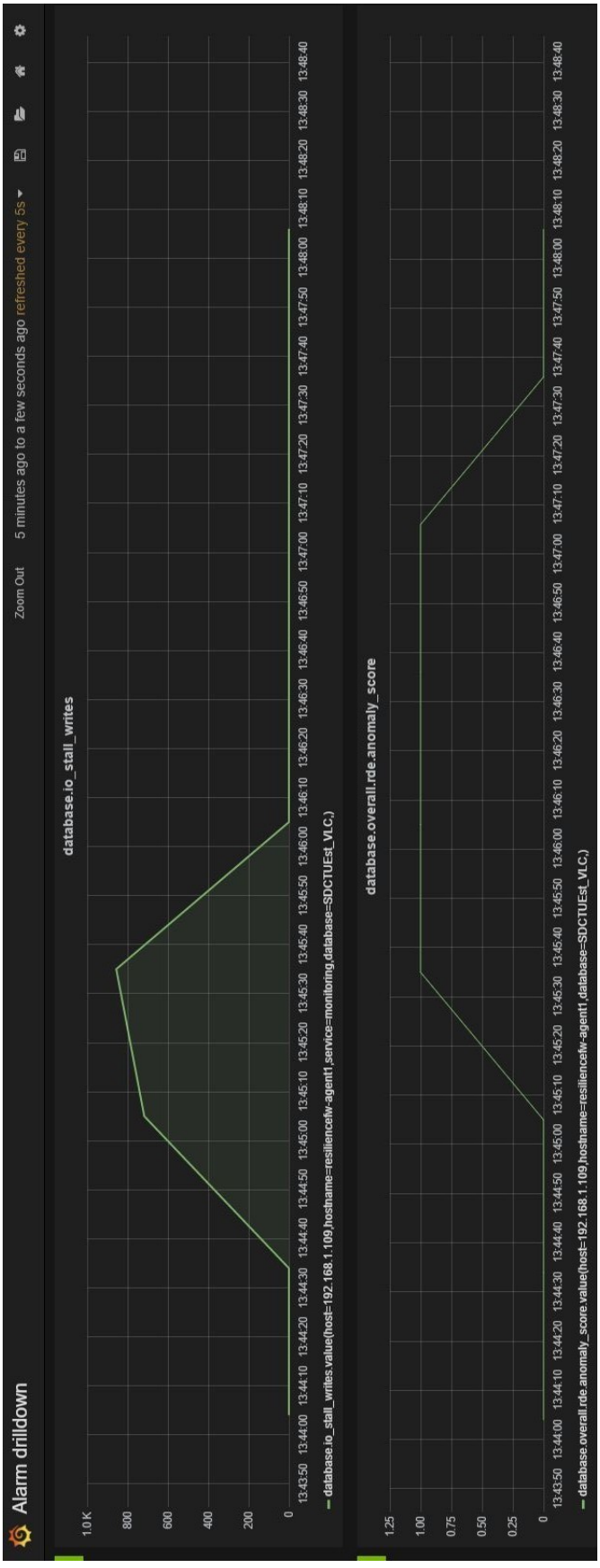


FIGURE C.1: Single metric observation and anomaly score for TC1

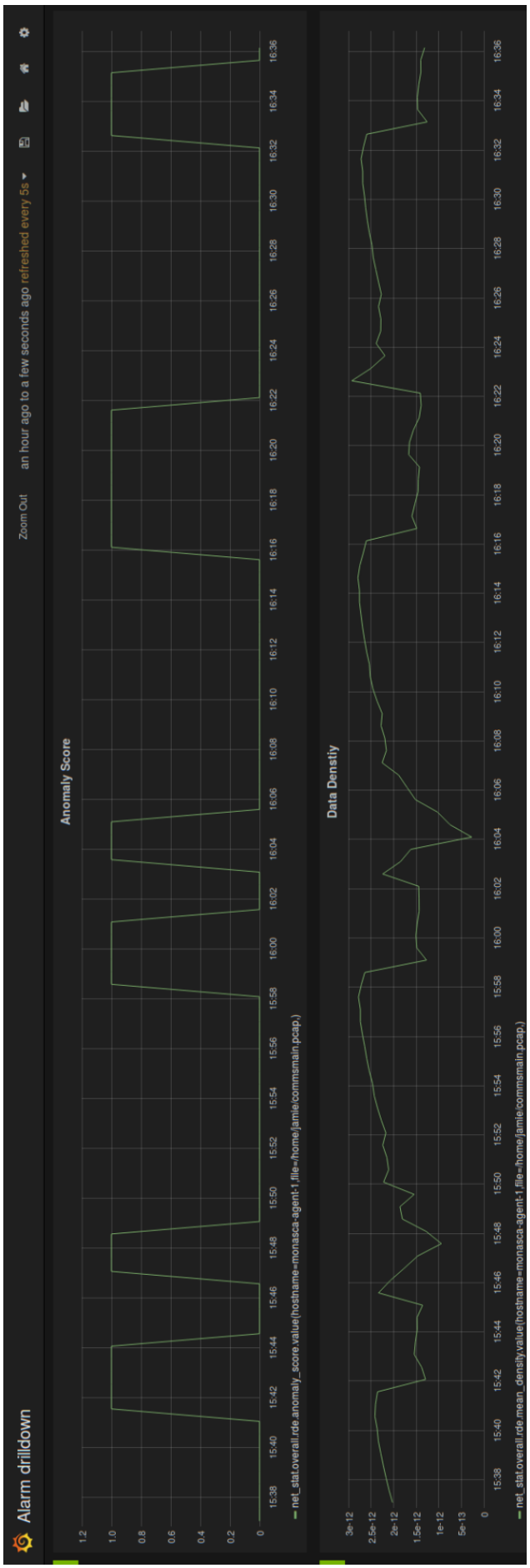


FIGURE C.2: Anomaly score and density plot for TC2

Bibliography

[1] Kirila Adamova et al. “Network anomaly detection in the cloud: The challenges of virtual service migration”. In: *Communications (ICC), 2014 IEEE International Conference on*. IEEE. 2014, pp. 3770–3775.

[2] Sandip Agarwala and Karsten Schwan. “Sysprof: Online distributed behavior diagnosis through fine-grain system monitoring”. In: *Distributed Computing Systems, 2006. ICDCS 2006. 26th IEEE International Conference on*. IEEE. 2006, pp. 8–8.

[3] Sandip Agarwala et al. “E2eprof: Automated end-to-end performance management for enterprise systems”. In: *Dependable Systems and Networks, 2007. DSN'07. 37th Annual IEEE/IFIP International Conference on*. IEEE. 2007, pp. 749–758.

[4] Marcos K Aguilera et al. “Performance debugging for distributed systems of black boxes”. In: *ACM SIGOPS Operating Systems Review*. Vol. 37. 5. ACM. 2003, pp. 74–89.

[5] Zubaida Alazawi et al. “Intelligent disaster management system based on cloud-enabled vehicular networks”. In: *ITS Telecommunications (ITST), 2011 11th International Conference on*. IEEE. 2011, pp. 361–368.

[6] Azman Ali et al. “Justifying a Policy Based Approach for DDoS Remediation: A Case Study”. In: *11th Annual Conference on the Convergence of Telecommunications, Networking & Broadcasting (PGNet 2010), Liverpool, UK*. 2010, pp. 21–22.

[7] *An Architectural Framework for Critical Infrastructure in Cloud Computing*. Tech. rep. SECCRIT Consortium, 2013.

[8] P. Angelov, P. Sadeghi-Tehran, and R. Ramezani. “A Real-time Approach to Autonomous Novelty Detection and Object Tracking in Video Stream”. In: *International Journal of Intelligent Systems*. Vol. 26/3. 2011, pp. 189–205.

[9] Plamen Angelov. “An approach for fuzzy rule-base adaptation using on-line clustering”. In: *International Journal of Approximate Reasoning* 35.3 (2004), pp. 275–289.

[10] Plamen Angelov. *Autonomous Learning Systems: From Data to Knowledge in Real Time*. Vol. 1. John Willey and Sons, 2012.

[11] Plamen Angelov. *Evolving rule-based models: a tool for design of flexible adaptive systems*. Vol. 92. Springer, 2002.

[12] Plamen Angelov and Ronald Yager. "Simplified Fuzzy rule-based systems using non-parametric antecedents and relative data density". In: *Evolving and Adaptive Intelligent Systems (EAIS), 2011 IEEE Workshop on*. IEEE. 2011, pp. 62–69.

[13] Fatemeh Azmandian et al. "Virtual Machine Monitor-based Lightweight Intrusion Detection". In: *SIGOPS Oper. Syst. Rev.* 45.2 (July 2011), pp. 38–53. ISSN: 0163-5980. DOI: 10.1145/2007183.2007189.

[14] Paramvir Bahl et al. "Towards highly reliable enterprise network services via inference of multi-level dependencies". In: *ACM SIGCOMM Computer Communication Review*. Vol. 37. 4. ACM. 2007, pp. 13–24.

[15] Sina Bahram et al. "DKSM: Subverting Virtual Machine Introspection for Fun and Profit". In: *Proceedings of the 2010 29th IEEE Symposium on Reliable Distributed Systems*. SRDS '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 82–91. ISBN: 978-0-7695-4250-8. DOI: 10.1109/SRDS.2010.39. URL: http://dx.doi.org/10.1109/SRDS.2010.39.

[16] A. Bakshi and B. Yogesh. "Securing Cloud from DDOS Attacks Using Intrusion Detection System in Virtual Machine". In: *Communication Software and Networks, 2010. ICCSN '10. Second International Conference on*. 2010, pp. 260–264. DOI: 10.1109/ICCSN.2010.56.

[17] Paul Barham et al. "Using Magpie for Request Extraction and Workload Modelling." In: *OSDI*. Vol. 4. 2004, pp. 18–18.

[18] Mandis S Beigi, Seraphin Calo, and Dinesh Verma. "Policy transformation techniques in policy-based systems management". In: *Policies for Distributed Systems and Networks, 2004. POLICY 2004. Proceedings. Fifth IEEE International Workshop on*. IEEE. 2004, pp. 13–22.

[19] Theophilus Benson et al. "A First Look at Problems in the Cloud". In: *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing*. HotCloud'10. Boston, MA: USENIX Association, 2010, pp. 15–15. URL: http://dl.acm.org/citation.cfm?id=1863103.1863118.

[20] Saul Berman et al. *The power of cloud. Driving business model innovation*. Tech. rep. IBM Institute for Business Value, 2012.

[21] Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*. Vol. 1. springer New York, 2006.

[22] R. Blaisdell. *Cloud Benefits in the Energy and Utility Industry*. https://www.rickscloud.com/cloud-benefits-in-the-energy-and-utility-industry/. Feb. 2012.

[23] Peter Bodik et al. "Fingerprinting the datacenter: automated classification of performance crises". In: *Proceedings of the 5th European conference on Computer systems*. ACM. 2010, pp. 111–124.

[24] Robert B. Bohn et al. "NIST Cloud Computing Reference Architecture". In: *Proceedings of the 2011 IEEE World Congress on Services*. SERVICES '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 594–596. ISBN: 978-0-7695-4461-8. DOI: 10.1109/SERVICES.2011.105. URL: http://dx.doi.org/10.1109/SERVICES.2011.105.

[25] Santiago Cáceres Elvira and Oliviero Francesco. *Deliverable 1.2:Report on requirements and use cases*. https://seccrit.eu/upload/D2-1-Report_on_requirements_and_use_cases-v2.0.pdf. 2013.

[26] Santiago Cáceres Elvira and Oliviero Francesco. *Deliverable 2.1:Report on requirements and use cases*. https://seccrit.eu/upload/D2-1-Report_on_requirements_and_use_cases-v2.0.pdf. 2013.

[27] Santiago Cáceres Elvira and Oliviero Francesco. *Deliverable 4.3: Mechanism and tool for anomaly detection*. https://www.seccrit.eu/upload/upload/D4.3_Mechanism_and_tool_for_anomaly_detection_v2.pdf. 2015.

[28] Fazli Can. "Incremental clustering for dynamic information processing". In: *ACM Transactions on Information Systems (TOIS)* 11.2 (1993), pp. 143–164.

[29] M Casassa Mont, Adrian Baldwin, and Cheh Goh. "POWER prototype: Towards integrated policy-based management". In: *Network Operations and Management Symposium, 2000. NOMS 2000. 2000 IEEE/IFIP*. IEEE. 2000, pp. 789–802.

[30] D Catteddu. *Security and Resilience in Governmental Clouds: Making an Informed Decision*. Tech. rep. http://www.enisa.europa.eu/act/rm/emerging-and-future-risk/deliverables/security-and-resilience-in-governmental-clouds. European Network and Information Security Agency (ENISA), Jan. 2011.

[31] V. Chandola, A. Banerjee, and V. Kumar. "Anomaly detection: A survey". In: *ACM Computing Surveys*. Vol. 41/3. 2009, pp. 1–58.

[32] Varun Chandola, Arindam Banerjee, and Vipin Kumar. *Anomaly Detection: A Survey*. 2007.

[33] Varun Chandola, Arindam Banerjee, and Vipin Kumar. "Anomaly detection: A survey". In: *ACM Computing Surveys (CSUR)* 41.3 (2009), p. 15.

[34] Mike Y Chen et al. "Pinpoint: Problem determination in large, dynamic internet services". In: *Dependable Systems and Networks, 2002. DSN 2002. Proceedings. International Conference on*. IEEE. 2002, pp. 595–604.

[35] Piotr Cholda et al. "A survey of resilience differentiation frameworks in communication networks". In: *Communications Surveys & Tutorials, IEEE* 9.4 (2007), pp. 32–55.

[36] *Cisco intrusion prevention system*. Tech. rep. http://www.cisco.com/go/ips. Cisco.

[37] Ira Cohen et al. "Capturing, indexing, clustering, and retrieving system history". In: *ACM SIGOPS Operating Systems Review*. Vol. 39. 5. ACM. 2005, pp. 105–118.

[38] Ira Cohen et al. "Correlating Instrumentation Data to System States: A Building Block for Automated Diagnosis and Control." In: *OSDI*. Vol. 4. 2004, pp. 16–16.

[39] Frédéric Cuppens, Nora Cuppens-Boulahia, and Céline Coma. "MotOrBAC: un outil d'administration et de simulation de politiques de sécurité". In: *Security in Network Architectures (SAR) and Security of Information Systems (SSI), First Joint Conference*. 2006, pp. 6–9.

[40] Frédéric Cuppens and Alexandre Miege. "Administration model for or-bac". In: *On The Move to Meaningful Internet Systems 2003: OTM 2003 Workshops*. Springer. 2003, pp. 754–768.

[41] Amir Vahid Dastjerdi, Kamalrulnizam Abu Bakar, and Sayed Gholam Hassan Tabatabaei. "Distributed intrusion detection in clouds using mobile agents". In: *Advanced Engineering Computing and Applications in Sciences, 2009. ADVCOMP'09. Third International Conference on*. IEEE. 2009, pp. 175–180.

[42] Tarek Elgamal and Mohamed Hefeeda. "Analysis of PCA algorithms in distributed environments". In: *arXiv preprint arXiv:1503.05214* (2015).

[43] Andreas Fischer et al. "Wide-area virtual machine migration as resilience mechanism". In: *Reliable Distributed Systems Workshops (SRDSW), 2011 30th IEEE Symposium on*. IEEE. 2011, pp. 72–77.

[44] Ian Foster et al. "Cloud computing and grid computing 360-degree compared". In: *Grid Computing Environments Workshop, 2008. GCE'08*. Ieee. 2008, pp. 1–10.

[45] Nir Friedman, Dan Geiger, and Moises Goldszmidt. "Bayesian network classifiers". In: *Machine learning* 29.2-3 (1997), pp. 131–163.

[46] Shekhar R Gaddam, Vir V Phoha, and Kiran S Balagani. "K-means+ id3: A novel method for supervised anomaly detection by cascading k-means clustering and id3 decision tree learning methods". In: *Knowledge and Data Engineering, IEEE Transactions on* 19.3 (2007), pp. 345–354.

[47] Thomas Gamer. "Anomaly-based identification of large-scale attacks". In: *Global Telecommunications Conference, 2009. GLOBECOM 2009. IEEE*. IEEE. 2009, pp. 1–6.

[48] *Ganglia Monitoring System*. http://www.ganglia.info.

[49] Tal Garfinkel and Mendel Rosenblum. "A Virtual Machine Introspection Based Architecture for Intrusion Detection". In: *In Proc. Network and Distributed Systems Security Symposium*. 2003, pp. 191–206.

[50] Tal Garfinkel, Mendel Rosenblum, et al. "A Virtual Machine Introspection Based Architecture for Intrusion Detection." In: *NDSS*. Vol. 3. 2003, pp. 191–206.

[51] F. Gens. *New IDC IT Cloud Service Survey: Top Benefits and Challenges, IDC Exchange*. Tech. rep. International Data Corporation, 2009.

[52] Cheh Goh. *A Generic Approach to Policy Description in System Management*. Hewlett Packard Laboratories, 1997.

[53] D. Goodin. *Webhost hack wipes out data for 100,000 sites*. http://www.theregister.co.uk/2009/06/08/webhost_attack/. June 2009.

[54] Bernd Grobauer, Tobias Walloschek, and Elmar Stocker. "Understanding Cloud Computing Vulnerabilities". In: *IEEE Security and Privacy* 9.2 (Mar. 2011), pp. 50–57. ISSN: 1540-7993. DOI: 10.1109/MSP.2010.115. URL: http://dx.doi.org/10.1109/MSP.2010.115.

[55] Qiang Guan and Song Fu. "Adaptive anomaly identification by exploring metric subspace in cloud computing infrastructures". In: *Reliable Distributed Systems (SRDS), 2013 IEEE 32nd International Symposium on*. IEEE. 2013, pp. 205–214.

[56] Qiang Guan et al. "Exploring Time and Frequency Domains for Accurate and Automated Anomaly Detection in Cloud Computing Systems". In: *Dependable Computing (PRDC), 2013 IEEE 19th Pacific Rim International Symposium on*. IEEE. 2013, pp. 196–205.

[57] Y. Guan and J. Bao. "A CP Instrusion Detection Strategy on Cloud Computing". In: *International Symposium on Web Information Systems and Applications*. 2009, pp. 84–87.

[58] Yizhang Guan and Jianghong Bao. "A CP Intrusion Detection Strategy on Cloud Computing". In: *International Symposium on Web Information Systems and Applications (WISA)*. 2009, pp. 84–87.

[59] H-G Hegering, Sebastian Abeck, and René Wies. "A corporate operation framework for network service management". In: *Communications Magazine, IEEE* 34.1 (1996), pp. 62–68.

[60] Nikolas Herbst et al. "Ready for Rain? A View from SPEC Research on the Future of Cloud Metrics". In: *arXiv preprint arXiv:1604.03470* (2016).

[61] Nikolas Roman Herbst, Samuel Kounev, and Ralf H Reussner. "Elasticity in Cloud Computing: What It Is, and What It Is Not." In: *ICAC*. 2013, pp. 23–27.

[62] Michael R. Hines and Kartik Gopalan. "Post-copy Based Live Virtual Machine Migration Using Adaptive Pre-paging and Dynamic Self-ballooning". In: *Proceedings of the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*. VEE '09. Washington, DC, USA: ACM, 2009, pp. 51–60. ISBN: 978-1-60558-375-4. DOI: 10.1145/1508293.1508301. URL: http://doi.acm.org/10.1145/1508293.1508301.

[63] *HP Systems Insight Manager*. http://www8.hp.com/uk/en/products/server-software/product-detail.html?oid=489496.

[64] Dawei Huang et al. "Virt-LM: A Benchmark for Live Migration of Virtual Machine". In: *Proceedings of the 2Nd ACM/SPEC International Conference on Performance Engineering*. ICPE '11. Karlsruhe, Germany: ACM, 2011, pp. 307–316. ISBN: 978-1-4503-0519-8. DOI: 10.1145/1958746.1958790. URL: http://doi.acm.org/10.1145/1958746.1958790.

[65] Mary Inaba, Naoki Katoh, and Hiroshi Imai. "Applications of weighted Voronoi diagrams and randomization to variance-based k-clustering". In: *Proceedings of the tenth annual symposium on Computational geometry*. ACM. 1994, pp. 332–339.

[66] Anil K. Jain and Richard C. Dubes. *Algorithms for Clustering Data*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1988. ISBN: 0-13-022278-X.

[67] Wayne Jansen and Timothy Grance. *SP 800-144. Guidelines on Security and Privacy in Public Cloud Computing*. Tech. rep. Gaithersburg, MD, United States, 2011.

[68] Ravi Jhawar and Vincenzo Piuri. "Fault tolerance and resilience in cloud computing environments". In: *Computer and Information Security Handbook,* (2013), pp. 125–141.

[69] Prasan Kaikini et al. *Method and apparatus for defining and enforcing policies for configuration management in communications networks.* US Patent 5,872,928. 1999.

[70] Anas Abou El Kalam et al. "Organization based access control". In: *Policies for Distributed Systems and Networks, 2003. Proceedings. POLICY 2003. IEEE 4th International Workshop on.* IEEE. 2003, pp. 120–131.

[71] W. Kampichler and D. Eier. "Cloud based services in air traffic management". In: *Integrated Communications, Navigation and Surveillance Conference (ICNS), 2012.* 2012, G5–1–G5–9. DOI: 10.1109/ICNSurv.2012.6218402.

[72] Hui Kang, Haifeng Chen, and Guofei Jiang. "PeerWatch: a fault detection and diagnosis tool for virtualized consolidation systems". In: *Proceedings of the 7th international conference on Autonomic computing.* ACM. 2010, pp. 119–128.

[73] Hui Kang, Xiaoyun Zhu, and Jennifer L Wong. "DAPA: diagnosing application performance anomalies for virtualized infrastructures". In: *Presented as part of the 2nd USENIX Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services.* 2012.

[74] Emre Kiciman and Armando Fox. "Detecting application-level failures in component-based internet services". In: *Neural Networks, IEEE Transactions on* 16.5 (2005), pp. 1027–1041.

[75] Samuel Kounev et al. "Providing dependability and resilience in the cloud: Challenges and opportunities". In: *Resilience Assessment and Evaluation of Computing Systems.* Springer, 2012, pp. 65–81.

[76] Michael A Kozuch, Michael Kaminsky, and Michael P Ryan. "Migration without Virtualization." In: *HotOS.* 2009.

[77] Alex Krikos. "Cloud computing as a disruptive technology". In: *Cloudbook Journal* 2.2 (2011), pp. 13–18.

[78] A. Lakhina, M. Crovella, and C. Diot. "Mining anomalies using traffic feature distributions". In: *SIGCOMM Computer Communications Review.* Vol. 35/4. 2005, pp. 217–228.

[79] Anukool Lakhina, Mark Crovella, and Christophe Diot. "Mining Anomalies Using Traffic Feature Distributions". In: *Proceedings of the 2005 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications.* SIGCOMM '05. Philadelphia, Pennsylvania, USA: ACM, 2005, pp. 217–228. ISBN: 1-59593-009-4. DOI: 10.1145/1080091.1080118. URL: http://doi.acm.org/10.1145/1080091.1080118.

[80] Aleksandar Lazarevic et al. "A Comparative Study of Anomaly Detection Schemes in Network Intrusion Detection." In: *SDM*. SIAM. 2003, pp. 25–36.

[81] Jun-Ho Lee et al. "Multi-level intrusion detection system and log management in cloud computing". In: *Advanced Communication Technology (ICACT), 2011 13th International Conference on*. IEEE. 2011, pp. 552–555.

[82] Alert Logic. *The Changing State of Cloud Security*. Tech. rep. 2015.

[83] Edwin Lughofer and Carlos Guardiola. "On-line fault detection with data-driven evolving fuzzy models". In: *Control and intelligent systems* 36.4 (2008), p. 307.

[84] Na Luo and Feng Qian. "Estimation of Distribution Algorithm sampling under Gaussian and Cauchy distribution in continuous domain". In: *Control and Automation (ICCA), 2010 8th IEEE International Conference on*. 2010, pp. 1716–1720. DOI: 10.1109/ICCA.2010.5524432.

[85] M.A.C.Dekker. *Critical cloud computing: A CIIP perspective on cloud computing services*. Tech. rep. European Network and Information Security Agency, 2012.

[86] M.A.C.Dekker. *Resilience in the era of enterprise cloud computing*. Tech. rep. IBM Global Technology Services, 2014.

[87] Markos Markou and Sameer Singh. "Novelty Detection: A Review - Part 1: Statistical Approaches". In: *Signal Processing* 83 (2003), p. 2003.

[88] A. Marnerides et al. "A snapshot of Malware Analysis over the Cloud: Network and System Characteristics". In: *In Proc. IEEE Globecom 2013 Workshop on Cloud Computing Systems, Networks, and Applications (CCSNA)*. 2013.

[89] Angelos Marnerides et al. "Multi-level network resilience: traffic analysis, anomaly detection and simulation". In: *ICTACT Journal on Communication Technology, Special Issue on Next Generation Wireless Networks and Applications* 2.2 (2011).

[90] Angelos K Marnerides, David Hutchison, and Dimitrios P Pezaros. "Autonomic diagnosis of anomalous network traffic". In: *World of Wireless Mobile and Multimedia Networks (WoWMoM), 2010 IEEE International Symposium on a*. IEEE. 2010, pp. 1–6.

[91] L. Martin. *Awareness, Trust and Security to Shape Government Cloud Adoption*. Tech. rep. Lockheed Martin, Apr. 2010.

[92] Matthew L Massie, Brent N Chun, and David E Culler. "The ganglia distributed monitoring system: design, implementation, and experience". In: *Parallel Computing* 30.7 (2004), pp. 817–840.

[93] Claudio Mazzariello, Roberto Bifulco, and Roberto Canonico. "Integrating a network IDS into an open source Cloud Computing environment." In: *IAS*. IEEE, 2010, pp. 265–270. ISBN: 978-1-4244-7407-3. URL: http://dblp.uni-trier.de/db/conf/IEEEias/IEEEias2010.html#MazzarielloBC10.

[94] Peter Mell and Tim Grance. "The NIST definition of cloud computing". In: (2011).

[95] Bernd Meyer, Frank Anstötz, and Claudia Popien. "Towards implementing policy-based systems management". In: *Distributed Systems Engineering* 3.2 (1996), p. 78.

[96] *Migrating Virtual Machines in VSphere Client*. 2010.

[97] Chirag Modi et al. "A survey of intrusion detection techniques in cloud". In: *Journal of Network and Computer Applications* 36.1 (2013), pp. 42–57.

[98] Chirag Modi et al. "A survey of intrusion detection techniques in Cloud." In: *J. Network and Computer Applications* 36.1 (2013), pp. 42–57. URL: http://dblp.uni-trier.de/db/journals/jnca/jnca36.html#ModiPBPPR13.

[99] Jose Moura and David Hutchison. "Review and analysis of networking challenges in cloud computing". In: *Journal of Network and Computer Applications* 60 (2016), pp. 113–129.

[100] Z Muda et al. "Intrusion detection based on K-Means clustering and Naïve Bayes classification". In: *Information Technology in Asia (CITA 11), 2011 7th International Conference on*. IEEE. 2011, pp. 1–6.

[101] D Neal. *Amazon Web Services outages raise serious cloud questions*. Tech. rep. http://www.v3.co.uk/v3-uk/news/2035726/amazon-web-services-outages-raise-cloud-questions. Mar. 2011.

[102] Jon Oberheide, Evan Cooke, and Farnam Jahanian. "Empirical exploitation of live virtual machine migration". In: *Proc. of BlackHat DC convention*. 2008.

[103] Simon Oblak, Igor Škrjanc, and Sašo Blažič. "Fault detection for nonlinear systems with uncertain parameters based on the interval fuzzy model". In: *Engineering Applications of Artificial Intelligence* 20.4 (2007), pp. 503–510.

[104] Mrutyunjaya Panda and Manas Ranjan Patra. "Network intrusion detection using naive bayes". In: *International journal of computer science and network security* 7.12 (2007), pp. 258–263.

[105] Husanbir S Pannu, Jianguo Liu, and Song Fu. "Aad: Adaptive anomaly detection system for cloud computing infrastructures". In: *Reliable Distributed Systems (SRDS), 2012 IEEE 31st Symposium on*. IEEE. 2012, pp. 396–397.

[106] C. Pascoal et al. "Robust Feature Selection and Robust PCA for Internet Traffic Anomaly Detection". In: *IEEE INFOCOM*. Vol. 2012. 2012, pp. 1755–1763.

[107] PCWord. *Spanish Bank to Move 100,000+ Employees to Google Apps*. http://www.pcworld.com/article/247851/spanish_bank_to_move_100000_employees_to_google_apps.html. Jan. 2012.

[108] Richard Piggin. "Are Industrial Control Systems Ready for the Cloud?" In: *Int. J. Crit. Infrastruct. Prot.* 9.C (June 2015), pp. 38–40. ISSN: 1874-5482. DOI: 10.1016/j.ijcip.2014.12.005. URL: http://dx.doi.org/10.1016/j.ijcip.2014.12.005.

[109] *PRECYSE*. http://www.precyse.eu/. Accessed: 2014-10-26.

[110] Light Reading. *Orange/SITA Cloud Prepares for Takeoff*. http://www.lightreading.com/services-apps/cloud-services/orange-sita-cloud-prepares-for-takeoff/d/d-id/693612. Feb. 2012.

[111] Richard A Redner and Homer F Walker. "Mixture densities, maximum likelihood and the EM algorithm". In: *SIAM review* 26.2 (1984), pp. 195–239.

[112] GE Global Research. http://www.geglobalresearch.com/innovation/ge-works-bring-air-traffic-management-cloud. 2011.

[113] *ResumeNet*. http://www.resumenet.eu/. Accessed: 2014-10-26.

[114] Douglas A Reynolds. *Gaussian Mixture Models*. 2009.

[115] H. Ringberg et al. "Sensitivity of PCA for traffic anomaly detection". In: *SIGMETRICS Performance Evaluation Review*. Vol. 35/1. 2007, pp. 109–120.

[116] Haakon Ringberg et al. "Sensitivity of PCA for Traffic Anomaly Detection". In: *Proceedings of the 2007 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*. SIGMETRICS '07. San Diego, California, USA: ACM, 2007, pp. 109–120. ISBN: 978-1-59593-639-4. DOI: 10.1145/1254882.1254895. URL: http://doi.acm.org/10.1145/1254882.1254895.

[117] Jan Roos, Phillip Putter, and Carel Bekker. "Modelling Management Policy Using Enriched Managed Objects". In: *Proceedings of the IFIP TC6/WG6. 6 Third International Symposium on Integrated Network Management with participation of the IEEE Communications Society CNOM and with support from the Institute for Educational Services*. North-Holland Publishing Co. 1993, pp. 207–215.

[118] Tiago Rosado and Jorge Bernardino. "An overview of openstack architecture". In: *Proceedings of the 18th International Database Engineering & Applications Symposium*. ACM. 2014, pp. 366–367.

[119] Joanna Rutkowska. *Subverting Vista Kernel for Fun and Profit*. Black Hat Talk. 2006.

[120] Alberto Schaeffer-Filho, Paul Smith, and Andreas Mauthe. "Policy-driven network simulation: a resilience case study". In: *Proceedings of the 2011 ACM Symposium on Applied Computing*. ACM. 2011, pp. 492–497.

[121] Alberto Schaeffer-Filho et al. "PReSET: A toolset for the evaluation of network resilience strategies". In: *Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on*. IEEE. 2013, pp. 202–209.

[122] M. Schöller et al. "An Architectural Model for Deploying Critical Infrastructure Services in the Cloud". In: *2013 IEEE 5th International Conference on Cloud Computing Technology and Science*. Vol. 1. 2013, pp. 458–466. DOI: 10.1109/CloudCom.2013.67.

[123] *Security network intrusion prevention system*. Tech. rep. http://www-01.ibm.com/software/tivoli/products/security-network-intrusion-prevention/. IBM.

[124] Bikash Sharma et al. "CloudPD: Problem determination and diagnosis in shared dynamic clouds". In: *2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE. 2013, pp. 1–12.

[125] S. N. U.H Shirazi et al. "Anomaly Detection in the Cloud Using Data Density". In: *2016 IEEE 9th International Conference on Cloud Computing (CLOUD)*. 2016, pp. 616–623. DOI: 10.1109/CLOUD.2016.0087.

[126] S.N.U.H Shirazi et al. "A framework for resilience management in the cloud". In: *e & i Elektrotechnik und Informationstechnik* 132.2 (2015), pp. 122–132. ISSN: 1613-7620. DOI: 10.1007/s00502-015-0290-9. URL: http://dx.doi.org/10.1007/s00502-015-0290-9.

[127] S.N.U.H Shirazi et al. "Assessing the impact of intra-cloud live migration on anomaly detection". In: *Cloud Networking (CloudNet), 2014 IEEE 3rd International Conference on*. 2014, pp. 52–57. DOI: 10.1109/CloudNet.2014.6968968.

[128] S.N.U.H. Shirazi et al. "Assessing the impact of intra-cloud live migration on anomaly detection". In: *Cloud Networking (CloudNet), 2014 IEEE 3rd International Conference on*. 2014, pp. 52–57. DOI: 10.1109/CloudNet.2014.6968968.

[129] Benjamin H Sigelman et al. *Dapper, a large-scale distributed systems tracing infrastructure*. Tech. rep. Technical report, Google, 2010.

[130] Vision Solutions. *The 2016 State of Resilience: Keep your data moving forward*. Tech. rep. 2016.

[131] Mohammed H Sqalli, Fahd Al-Haidari, and Khaled Salah. "Edos-shield-a two-steps mitigation technique against edos attacks in cloud computing". In: *Utility and Cloud Computing (UCC), 2011 Fourth IEEE International Conference on*. IEEE. 2011, pp. 49–56.

[132] James PG Sterbenz et al. "Resilience and survivability in communication networks: Strategies, principles, and survey of disciplines". In: *Computer Networks* 54.8 (2010), pp. 1245–1265.

[133] Y. Sverdlik. *Nasdaq and Amazon launch cloud platform for financial services*. http://www.datacenterdynamics.com/content-tracks/colo-cloud/nasdaq-and-amazon-launch-cloud-platform-for-financial-services/69672.fullarticle. Sept. 2012.

[134] Yongmin Tan et al. "Prepare: Predictive performance anomaly prevention for virtualized cloud systems". In: *Distributed Computing Systems (ICDCS), 2012 IEEE 32nd International Conference on*. IEEE. 2012, pp. 285–294.

[135] *TClouds*. http://www.tclouds-project.eu//. Accessed: 2014-10-26.

[136] CSA CCM Leadership Team. *Cloud Security alliance cloud controls matrix v1.1*. Tech. rep. 2010.

[137] *Tivoli Network Manager*. http://www-01.ibm.com/software/uk/tivoli/.

[138] Franco Travostino et al. "Seamless live migration of virtual machines over the MAN/WAN". In: *Future Generation Computer Systems* 22.8 (2006), pp. 901–907.

[139] Robbert Van Renesse, Kenneth P Birman, and Werner Vogels. "Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining". In: *ACM transactions on computer systems (TOCS)* 21.2 (2003), pp. 164–206.

[140] Dinesh C Verma. *Policy-based networking: architecture and algorithms*. New Riders Publishing, 2000.

[141] Chengwei Wang. "EbAT: online methods for detecting utility cloud anomalies". In: *Proceedings of the 6th Middleware Doctoral Symposium*. ACM. 2009, p. 4.

[142] Chengwei Wang et al. "Online detection of utility cloud anomalies using metric distributions". In: *Network Operations and Management Symposium (NOMS), 2010 IEEE*. IEEE. 2010, pp. 96–103.

[143] Chengwei Wang et al. "Statistical techniques for online anomaly detection in data centers". In: *Integrated Network Management (IM), 2011 IFIP/IEEE International Symposium on*. IEEE. 2011, pp. 385–392.

[144] Michael R Watson et al. "Malware detection in cloud computing infrastructures". In: *IEEE Transactions on Dependable and Secure Computing* 13.2 (2016), pp. 192–205.

[145] Brett Winterford. "Stress tests rain on Amazon's cloud". In: *IT News* 20 (2009).

[146] Timothy Wood et al. "Disaster Recovery as a Cloud Service: Economic Benefits & Deployment Challenges." In: *HotCloud* 10 (2010), pp. 8–15.

[147] Ningning Wu and Jing Zhang. "Factor-analysis based anomaly detection and clustering." In: *Decision Support Systems* 42.1 (Jan. 23, 2007), pp. 375–389. URL: http://dblp.uni-trier.de/db/journals/dss/dss42.html#WuZ06.

[148] Praveen Yalagandula and Mike Dahlin. *A scalable distributed information management system*. Vol. 34. 4. ACM, 2004.

[149] Yue Yu et al. "An adaptive approach to network resilience: Evolving challenge detection and mitigation". In: *Design of Reliable Communication Networks (DRCN), 2011 8th International Workshop on the*. IEEE. 2011, pp. 172–179.

[150] Harry Zhang. "The optimality of naive Bayes". In: *AA* 1.2 (2004), p. 3.

[151] Q. Zhang et al. "An Intelligent Anomaly Detection and Reasoning Scheme for VM Live Migration via Cloud Data Mining". In: *2013 IEEE 25th International Conference on Tools with Artificial Intelligence*. 2013, pp. 412–419. DOI: 10.1109/ICTAI.2013.68.

[152] Qi Zhang, Lu Cheng, and Raouf Boutaba. "Cloud computing: state-of-the-art and research challenges". In: *Journal of internet services and applications* 1.1 (2010), pp. 7–18.