Washington University in St. Louis
School of Engineering and Applied Science
Department of Computer Science and Engineering

Dissertation Examination Committee:
Raj Jain, Chair
Roger Chamberlain
Chenyang Lu
Benjamin Moseley
Ning Zhang

Management And Security Of Multi-Cloud Applications

By

Lav Gupta

A dissertation presented to the Graduate School of Arts and Sciences
of Washington University in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

May 2019
Saint Louis, Missouri

Table of Contents

List of Figures vii

List of Tables xi

Acknowledgments xiii

Chapter 1 Introduction 1

Chapter 2 Challenges in management and security of multi-cloud applications 5

2.1 Optimization of the Multi-Cloud Management Platform 5

2.1.1 Extracting Behavioral Data 6

2.1.2 Analyzing Platform Performance 7

2.2 Placement of Virtual Network Services in a Multi-Cloud Environment 8

2.2.1 Achieving Dynamic Placement in Multi-cloud Systems 10

2.2.2 Optimizing the SFC Performance 12

2.2.3 Meeting the Cost and Quality of Service Constraints 12

2.2.4 Speed and Accuracy of the Placement 13

2.3 Fault and Performance Management In Multi-cloud Virtual Network Services 14

2.3.1 Gaps in the Interface Specifications 15

2.3.2 Incomplete Definition of FCAPS 15

2.3.3 Complexity of VNSs 16

2.4 Security in Next Generation Healthcare 16

2.4.1 Inadequacy of the Current Models 17

2.4.2 Security of Data Flows Among Domains 18

Chapter 3 Addressing the multi-cloud management and security challenges 20

3.1 Optimization of Multi-cloud Management and Control Platforms 20

3.1.1 Collecting Behavioral Data 21

3.2 Placement of Virtual Network Services in a Multi-cloud Environment 25

3.2.1 Methods Based on ILP and its Variants for Optimization 25

3.2.2 ILP and Heuristic to Speed up the Solution 26

3.2.3 ILP and Heuristics for Comparison 27

3.2.4 Non-ILP Heuristic Solutions 28

3.3 Fault and Performance Management In Multi-cloud Virtual Network Services 29

3.3.1 NFVI Level Diagnostics 31
3.3.2 Causal Inference Based Methods 32
3.3.3 Statistical and AI-based Methods 33
3.4 Problems with Existing Security Approaches 35
Chapter 4 Optimization of multi-cloud management and control platform 41
4.1 Introduction and Motivation 41
4.2 Managing Software Defined Infrastructure Over Multiple Clouds – OpenADN 44
4.2.1 Components of OpenADN and Bootstrapping 46
4.2.2 Massively Distributed Nature of OpenADN 49
4.2.3 Design and Coding Considerations 50
4.3 Performance Modeling of OpenADN 51
4.3.1 Services of OpenADN 51
4.3.2 Metrics, Factors and Parameters 52
4.4 Profiling techniques for Gathering Behavioral Data 53
4.5 Evaluation Techniques And Experiment Design 54
4.5.1 Dynamic Layered Analysis for Identification of Factors for Optimization 55
4.5.2 Experimental Results and Analysis 63
4.6 Optimization Results 67
4.6.1 OpenADN design choices that affect performance 67
4.6.2 The Bootstrap Process Revisited 69
4.6.3 Details of Optimization 70
Chapter 5 Placement of Virtual Network Functions on Multi-Cloud Systems 77
5.1 Introduction and Motivation 78
5.2 Contributions 80
5.3 Virtual Network Service Environment 82
5.3.1 Constituents of a Virtual Network Service 82
5.3.2 The Multi-cloud Hierarchy 84
5.3.3 Representation of the Tenant Profile 85
5.4 Problem Definition 85
5.4.1 Achieving Dynamic Placement in Multi-cloud Systems 86
5.4.2 Optimizing the SFC Performance 86
5.4.3 Meeting the Cost and Latency Constraints 87

5.4.4 Speed and Efficiency of Placement 87
5.4.5 Interference Among VNFs 87
5.4.6 Problems Addressed in this Dissertation 88
5.5 The Proposed P-ART Framework 88
5.5.1 Information Available from Carriers and CSPs 89
5.6 Predictive Adaptive Real Time Strategy 91
5.6.1 Handling Change of State of the System by Predictive Placement 93
5.6.2 Incorporating Temporal Variation of Traffic in the Model 96
5.6.3 Corrections for Short-Term Traffic Variations 99
5.7 Cost optimization 101
5.7.1 Optimization of Cloud Selection by Random Search 101
5.8 Increasing Speed and Acceptance Rate of Placement 105
5.9 Combining the Elements of the Framework 106
5.10 Evaluation of the Framework 108
5.10.1 The Experimental Set-up for Evaluation 108
5.10.2 Selection of Features for Training the Prediction Models 109
5.10.3 Obtaining Training Datasets 110
5.10.4 Inter-VNF Traffic Flow Simulation 111
5.10.5 CloudLab Implementation 112
5.10.6 Selection of the Machine Learning Model 113
5.10.7 Tuning and Testing of the Predictive Model 117
5.10.8 Refinement of Latency Prediction by Compensating Concept Drift 119
5.10.9 Compensating Shot Term Traffic Variations 122
5.10.11 Cloud Optimization with Iterative Random Selection 122
5.10.12 Speed and Efficiency 125
5.11 Summary and Future Work 129
Chapter 6 Fault and Performance Management 131
6.1 Introduction and Motivation 132
6.2 Contributions 134
6.3 Structure of VNSs 135
6.3.1 A Top-down Discussion of VNS 136
6.4 Management of VNSs from an FP Management Perspective 140

6.4.1 Virtual Infrastructure Manager (VIM) 141
6.4.2 Virtual Network Function Manager (VNFM) 141
6.4.3 NFV Orchestrator (NFVO) 142
6.4.4 Catalogs and Repositories 142
6.4.5 MANO Reference Points 142
6.5 Comparison of Competitive Network Service Orchestration Offerings 143
6.6 Fault and Performance (FP) Problem Description 145
6.6.1 FP Issues and their Criticality 147
6.6.2 Shared FP Responsibilities 148
6.6.3 The FP Problem Statement 149
6.7 Methods for FP Management 150
6.7.1 NFVI Level Diagnostics 150
6.7.2 Causal Inference Based Methods 151
6.7.3 Statistical and AI-based Methods 152
6.8 Description of the Proposed FP Management Framework 155
6.8.1 Markers and Metrics for Fault Detection and Localization 156
6.8.2 AI Techniques used in the Framework 157
6.8.3 Design and Implementation of the HYPERVINES Framework 163
6.8.4 Data Pre-processing 165
6.8.5 The FP Detection Subsystem 166
6.8.6 The FP Localization Subsystem 168
6.9 Evaluation of the Model 173
6.9.1 Analysis of the Datasets Used 173
6.9.2 Evaluation of the detection subsystem 176
6.9.3 Evaluation of Localization Subsystem 179
6.9.4 Localization of Impending FP Issues 181
6.10 Summary 184
Chapter 7 Security in Next Generation Healthcare 186
7.1 Introduction and Motivation 187
7.2 Contributions 189
7.3 The Conceptual Framework of the Next Generation Healthcare 190
7.3.1 The Healthcare Network Domains 190

7.3.2 Design of Architecture for Next Generation Healthcare 193
7.3.3 Security Architecture for the Next Generation Healthcare 196
7.4 Threat Model for the Healthcare Network 199
7.4.1 Attack Surfaces 200
7.5 Merged Hierarchical Security for Data in Motion 204
7.5.1 Design Aspects of the Security System 204
7.5.2 The Choice of Deep Learning System 206
7.5.3 Deep Neural Networks – Stacked Autoencoders 208
7.5.4 Hierarchical Merged Model with Layer Reuse 211
7.6 Evaluation of the Proposed System 214
7.6.1 BOT-IoT Dataset 215
7.6.2 Data Generated on Testbed 216
7.6.3 Evaluation Results 218
7.7 Summary and Future Directions 225
Chapter 8 Conclusions and future directions 228
8.1 What Have We Achieved in this Dissertation? 229
8.2 Future Directions 231
Bibliography 234

List of Figures

Figure 4.1 OpenADN Multi-Cloud Management System 45

Figure 4.2 Key Components of OpenADN 47

Figure 4.3 Functions of OpenADN relevant to the bootstrap process 48

Figure 4.4 The OpenADN bootstrap flow diagram 49

Figure 4.5 The experimental set-up 56

Figure 4.6 The Layered Profiling Model 57

Figure 4.7 User, System and overall CPU time for OpenADN 58

Figure. 4.8 Extract of Function Level Profile 60

Figure 4.9 Extract of Statement Level Profile showing large sleep time 2000 users and 4 hosts 61

Figure 4.10 Extract of Statement Level Profile showing large time taken by Poller 2000 users and 4 hosts 62

Figure 4.11 Extract of profile for 2000 users and 8 hosts showing 68.9% host creation and linking time 62

Figure 4.12 Concurrency profiling with 2000 users and 16 hosts 63

Figure 4.13 (a) Residual vs. Predicted Response 66

Figure 4.13 (b) Quantile-Quantile Plot 66

Figure 4.14 http client connection (1 s case) 70

Figure 4.15 http client connection (1ms case) 71

Figure 4.16 Setting up the pPort 72

Figure 4.17 Setting up the pPort (after optimization) 72

Figure 4.18 Setting up the cPort 73

Figure 4.19 Setting up the cPort (after optimization) 73

Figure 4.20 Setting up the sPort 73
Figure 4.21 Setting up the sPort (after optimization) 74
Figure 4.22 UDP tunnel port initialization 74
Figure 4.23 cPort without udp_tunnel_port optimization 75
Figure 4.24 cPort bootstrapping without udp_tunnel_port optimization 75
Figure 4.25 cPort bootstrapping with udp_tunnel_port_optimization 75
Figure 5.1 Broadband service function chain and associated modules 83
Figure 5.2. Mapping service function chain to the multi-cloud system 84
Figure 5.3. The configuration of the experimental service chain 88
Figure 5.4. Need for predictive placement 95
Figure 5.5. (a) Traffic variation on Chicago-Seattle link 97
Figure 5.5 (b) Traffic variation on Los Angeles-San Jose Link 97
Figure 5.6. Comparison of generalization error with an integrated model and FPTV model 99
Figure 5.7 The P-ART placement framework 106
Figure 5.8 Traffic Delay Model for Data Generation 111
Figure 5.9. The CloudLab Implementation 112
Figure 5.10 Training and test error ratios (with standard error bars) 119
Figure 5.11 Extract of FTVP windows 119
Figure 5.12 Comparison of performance window-based integrated models 121
Figure 5.13. (a) 50 experiments with1500 iterations each 123
Figure 5.13. (b) 50 experiments with 1700 iterations each 123
Figure 5.14. Number of convergences in 50 experiments 123
Figure 5.15. Number of iterations required by randomized and M-Sequential to achieve latency below the threshold 124

Figure 5.16. Latencies achieved by randomized and M-Sequential in the number of iterations shown in Figure 5.15 124

Figure 5.17 Placement time Vs. No of SFCs 125

Figure 5.18 Placement time reported in [148] 126

Figure 5.19. Average placement time reported in [149] 127

Figure 5.20 Acceptance rate Vs. Number of SFCs 128

Figure 5.21. Acceptance % reported in [153] 129

Figure 6.1 A carrier's broadband service network 135

Figure 6.2. The virtualization hierarchy 136

Figure 6.3 Virtual broadband service 137

Figure 6.4 An SFC created on multiple clouds 137

Figure 6.5. A VNF implementation 138

Figure 6.6. Network function virtualization infrastructure 139

Figure 6.7. Orchestration and management of VNS 141

Figure 6.8 HYPER-VINES Fault and performance management mechanism 163

Figure 6.9 The environment of HYPER-VINES 164

Figure 6.10 The Architecture of HYPER-VINES Framework 165

Figure 6.11 The detection subsystem 166

Figure 6.12. The localization process 169

Figure 6.13 The stacked encoder used for prediction 171

Figure 6.14 Stacked sparse autoencoders 172

Figure 6.15 Detection Level 1 performance graph 178

Figure 6.16. Detection Level 2 effectiveness compared to baseline 178

Figure 6.17 Localization Layer 1 effectiveness compared to baseline 180

Figure 6.18 Localization Layer 2 effectiveness compared to baseline 181

Figure 6.19 Mean square error for reconstruction of the input 182
Figure 6.20 MSE in training and test dataset 183
Figure 6.21 Localization of impending fault stacked autoencoder and SVM (baseline) 183
Figure 7.1 Overall layout of the IoT-Cloud based healthcare 190
Figure 7.2 Architecture of the IoT-Cloud healthcare system 194
Figure 7.3 The security architecture of the next generation healthcare 199
Figure 7.4 Trust boundaries between zones 200
Figure 7.5 Hierarchical autoencoder based system 213
Figure 7.6 The merged model 214
Figure 7.7 Healthcare testbed for dataset generation 216
Figure 7.8 Extract of the Training Dataset 218
Figure 7.7 Three edge cloud autoencoders 219
Figure 7.8 Training and test performance of the edge cloud models 221
Figure 7.9 Merged model with all edge layers reused 221
Figure 7.10 Merged model with selective layer reuse 222
Figure 7.11 Public cloud merged model training and testing 223
Figure 7.12 Edge cloud training 224
Figure 7.13 Public cloud training 224
Figure 7.14 Learning Performance and Explainability of current and future systems [DARPA] 227

List of Tables

Table 4.1 Platform level execution data 57

Table 4.2 Time used for user and system activities on physical machines 59

Table 4.3 CPU Time for functions and workloads 64

Table 4.4 Analysis of variance for Functions and Workloads 65

Table 4.5 The estimated y and the residuals 65

Table 4.6 Calculation of CI of Effects 66

Table 4.7 Performance of client connection for varying execution suspension times 71

Table 4.8 Outcomes for port optimization 74

Table 5.1 Symbols used 90

Table 5.2 Categorization of server resources 109

Table 5.3 Predictor variables and the output label 110

Table 5.4 An extract of the integrated training dataset 113

Table 5.5 Comparative study of machine learning algorithms 116

Table 5.6 Training Error 118

Table 5.7 Test Error 118

Table 5.8 Probability distribution parameters in different windows 120

Table 5.9 Errors with integrated and multiple models 120

Table 5.10 Performance before online updation 122

Table 5.11 Performance after online updation 122

Table 6.1 Examples of commercially available VNFs 138

Table 6.2 Fault And performance information over NFV reference points 143

Table 6.3 Comparison of some competitive network service orchestration solutions from FP perspective 144

Table 6.4 Shared FP responsibilities of different management entities 149

Table 6.5 List of markers for different carrier services 156

Table 6.6 Metric for network availability and resiliency 157

Table 6.7 Machine/Deep learning algorithms for the FP problem 162

Table 6.8 Coarse (Layer1) and fine (Layer 2) categorization of faults 169

Table 6.9 (a) Training dataset 174

Table 6.9 (b) Test dataset 174

Table 6.10 List of features from network fault dataset 175

Table 6.11 Consolidated training dataset 176

Table 6.12 Stage-1 detection results 176

Table 6.13 Metric used 177

Table 6.14 Detection Level 1 performance 177

Table 6.15 Detection Level 2 performance 178

Table 6.16 Localization Performance 179

Table 7.1 Suitability of deep learning for IoT healthcare 207

Table 7.2 The feature set of the Bo-IoT dataset 215

Table 7.3 Features in the healthcare dataset 217

Table 7.3 Comparison of reuse and non-reuse models 224

Table 7.4 Confusion matrices for attack detection 225

Acknowledgments

It takes a village to raise a child, goes and old African proverb. In the same vein, a thesis is a product of the hard work of the protagonist supported by the crew and cast and steered by a capable helmsman. In this work there are too many contributors to acknowledge everyone individually, but I would not be fulfilling my responsibility if I do not express my gratitude to as many as possible. My sincerest apologies are due to those whom I inadvertently leave out.

The first and the foremost, I cannot thank enough my advisor Prof. Raj Jain whose faith in me was unflinching from beginning to the end. I joined PhD at an advanced stage in my life and thankfully, neither he ever made me conscious of that, nor mercifully showed any leniencies. This maintained a sense of normalcy through out the program and helped me to work as a normal student. He provided guidance in his trademark way of giving functional independence, at the same time not letting me go astray.

The others members of my dissertation committee have been simply astonishing. Prof Roger Chamberlain has provided me invaluable guidance whenever I needed to discuss any issue with him. Prof Chenyang Lu impressed me immensely with his sophisticated and effective way of guiding students. Prof Benjamin Moseley, now at CMU, has been a guiding force all through my research journey. With just one remark on my DSS evaluation, he infused in me the will to perform better in a far more effective way than hours of counseling. Prof Ning Zhang came late into my committee but discussions with him have provided me insight into working on the security of networks. My erstwhile dissertation committee member, Prof M. Samaka was my co-authors in many publications and was ever so helpful.

The computer science department has an outstanding faculty and staff. I would like to mention some with whom I have interacted and who have left an indelible mark on me. I did a rotation with Prof. Roch Guerin and learned greatly from the discussions I had with him. I found Prof Victor Gruev to be very personable and easy to work with. He never compromised on the teaching and the learning process and yet made students comfortable. Though I had limited interactions with Prof Sanmay Das but I greatly enjoyed his machine learning classes and owe to him whatever little I know about it. I had very fruitful discussions with Prof. Tao Ju and Prof. Yevgeniy Vorobeychik that changed my perspective about applied machine learning. One professor with whom I never took any class, or interacted personally, but whom admire immensely is Prof Ron Cytron, for the positivity that he emanates through his personable demeanor. I thank them all.

The Staff of the CSE Department and the Graduate School have been most amazing and helpful. I would like to mention Myrna Harbison, Monét Demming, Kelli Eckman, Sharon Matlock, Lia Garofolo, Cheryl Newman, Lauren Huffman and Johanna Sengheiser. They all have the knack of making any hopeless situations seem like a cakewalk.

I feel privileged to have collaborated with so many talented and delightful members of Prof Jain's network lab group, both students and the visiting postdoctoral researches. They have provided me with valuable insights through discussions, being my co-authors on a number of publications or polishing my half-baked ideas for publication. I would like to mention two past member of the group, Subharathi Paul, for his OpenADN multi-cloud platform that served as the starting point for this and Jianli Pan, who provided invaluable guidance on many aspects of proposal and dissertation defense. Of the current member of the group, I would like to thank Tara Salman for providing incessant help in the matters concerning the OpenADN platform and providing in-depth critique of the papers of which she was my co-author. I would also like to thank my other co-author, Maede Zolanwari, whom I jokingly call Maedely (a take on 'Grammarly'), for her ability to find even the minutest error in papers and presentations. Ali Ghubaish and Marcio A. Teixaira have been responsible for generating the attack data that helped me test the security aspects of my work. Both Tara and Ali have been very helpful in many other ways. Of the other students I would like to mention, not in any particular order, Hila Ben Abraham, Adam Dressler, Chao Wang, Mithun Chakraborty, Yijian Li, Dolvara Gunatilaka and Robert Utterback for helping in discussion on courses or matters of personal importance.

I would like to thank the funding agencies NSF, QNRF and Cisco for supporting my research. All the work that I have done for this thesis with regards to multi-cloud platform optimization, virtual network function placement, fault and performance management of virtual network services and cybersecurity of next generation healthcare have been supported in part or full by the following grants: QNRF grants NPRP 6-901-2-370, 8-634-1-131, NPRP10-0125-170250 from the Qatar National Research Fund (a member of Qatar Foundation) and National Science Foundation grants NSF CNS-1718929 and NSF CNS-1547380 .

I reserve the most heartfelt thanks for my wife Neeta for being constantly by my side in my journey of life and supporting me unconditionally. This work would not have seen the light of the day without her listening to my incessant rants about the difficulties of a PhD Scholar

In the end I would like to thank my father for always believing in me and for not being able to understand why PhD should take such a long time. All my children and their progenies, Dhruv, Manan, Aditi, Upasana, Nishka and Myra have always provided me with welcome relief from the rigors of the doctoral work.

To My Mother – My First Teacher!

ABSTRACT OF THE DISSERTATION

Management and Security of Multi-cloud Applications

by

Lav Gupta

Doctor of Philosophy in Computer Science and Engineering

Washington University in St. Louis, 2019

Professor Raj Jain, Chair

Single cloud management platform technology has reached maturity and is quite successful for information technology applications. Enterprises and application service providers are increasingly adopting a multi-cloud strategy to reduce the risk of cloud service provider lock-in and cloud black-outs and, at the same time, get the benefits like competitive pricing, the flexibility of resource provisioning and better points of presence. Another class of applications that are getting cloud service providers increasingly interested in is the carriers' virtualized network services. However, virtualized carrier services require high levels of availability and performance and impose stringent requirements on cloud services. They necessitate the use of multi-cloud management and innovative techniques for placement and performance management. We consider two classes of distributed applications – the virtual network services and the next generation of healthcare – that would benefit immensely from deployment over multiple clouds. This thesis deals with the design and development of new processes and algorithms to enable these classes of applications. We have evolved a method for optimization of multi-cloud platforms that will pave the way for obtaining optimized placement for both classes of services. The approach that we have followed for placement itself is predictive cost optimized latency controlled virtual resource placement for both types of applications. To improve the availability of virtual network services, we have made innovative use of the machine and deep learning for developing a framework for fault detection and localization. Finally, to secure patient data flowing through the wide expanse of sensors, cloud hierarchy, virtualized network, and visualization domain, we have evolved hierarchical autoencoder models for data in motion between the IoT domain and the multi-cloud domain and within the multi-cloud hierarchy.

Chapter 1

Introduction

The datacenter boom happened in the late 1990s providing a quick way for businesses to deploy their applications and make their presence felt on the Internet. The legacy datacenters, many of which still exist, consist of complex physical infrastructure, which is application specific and vendor driven. Installation and configuration of applications and services on this infrastructure are predominantly manual and, consequently, tedious and time consuming. With virtualization came software-defined infrastructure (SDI) offering virtual compute, virtual storage and virtual networking resources created in software over commodity hardware. These infrastructures are managed using software based management and control system. With SDI, businesses started getting a scalable infrastructure at reasonable cost that could accelerate their decisions to face competition better.

Cloud computing enables on-demand network access to a shared pool of configurable computing resources that can be rapidly provisioned and released, through a management platform, with minimal service provider interaction [1]. While the ultimate vision is to have a multi-cloud management and control platform (MMCP) that will completely automate management of resource lifecycle over multiple clouds, what proliferate today are single cloud platforms with reactive provisioning and re-configuration. Examples of such platforms are Amazon's EC2, Microsoft Azure, and Google Compute Engine. Use of single clouds has become commonplace in recent years in the government [2] and businesses [3]. This could have continued if it had not been for frequent cloud outages. Based on industry reports, in 2015 the three biggest providers, Amazon Web Services had 56 outages of total 2 hours and 30 minutes, Microsoft Azure had 71 outages of 10 hours and 49 minutes and Google Cloud Platform had 167 outages of 11 hours and 34 minutes [4]. Outages are equally common today. According to Google cloud status dashboard, in 2018, the Google Cloud compute engine, storage and networking were out for about 45, 22 and 33 hours respectively. For performance critical applications, like healthcare, such outages may be disastrous. Any ISP or carrier service provider, having five nines availability requirement, would allow a failure of less than 5 minutes and 15 seconds each year. This makes a strong case for multi-cloud systems, as statistically all the involved clouds will not fail together. In such a system Application and Internet Service Providers (ASPs and ISPs) would obtain resources from multiple clouds directly or through cloud aggregators. In either case, MMCP would help them automate deployment of their applications over virtual resources from the contracted clouds. With such automatic deployment and management of services, the ASPs and ISPs get agility of deployment, flexibility of resource provisioning, competitive provisioning, distributed points of presence and reduced risk of a total blackout.

Other than taking care of clouds going on the blink, the key forces for deployment over multiple clouds has so far been to avoid vendor lock-in and to come closer to the users. In today's multi-clouds platforms, some form of lifecycle management and auto scaling appear to be generally available. Inter-cloud migration of virtual machines (VMs) is complex and has been solved in a very few cases. None of the platforms offer optimization of its own functioning for the intended application. The placement algorithms built into these platforms may take tens of minutes to place a 50-function chain. Support for multi-level security for multi-cloud placement is not available. Finally, fault and performance management appropriate for carrier applications, like virtual cellular network services, needs to be incorporated in these platforms. It is thus evident that crucial pieces for enabling the virtual network services and distributed healthcare applications are missing.

Carrier services are today delivered through predominantly physical infrastructure. This causes lock-in with telecommunications equipment manufacturers and leads to networks that are high in total cost of acquisition and operation. They are time-consuming to deploy and inflexible in scaling and reuse. Virtualization of network functions, like routers and switches, and their deployment over clouds, makes them non-proprietary, reduces cost, improves scalability and reduces deployment time. However, such deployments do not yet meet the five nines availability and reliability of the traditional, physical appliance based, networks.

Virtualization of datacenter resources has been immensely successful for IT applications and carriers would like to replicate this success for their own services. Carriers see Network Function Virtualization (NFV) as a paradigm that could help them transpose this success to their networks by instantiating network functions on virtual machines hosted on commercial, off-the-shelf servers or better still on the clouds. The resulting Virtual Network Functions (VNFs), e.g., virtual routers and virtual load-balancers, form the basic building blocks of the Virtual Network Services (VNSs) like cellular mobile service and broadband service. To carriers, such deployments would mean freedom from proprietary solutions, ease of installation and scaling, reduced cost of operation and reduced time to market. Additionally, instantiating VNSs on multiple clouds brings advantages like proximity to users and avoidance of a single point of failure.

The catch in this utopian scheme is that VNSs do not yet meet the requirements of five-nines availability and quality of service that the traditional, largely physical and standards-based carrier networks have been assiduously built to provide [4]. The current research challenges relating to NFV include: the complexity of cloud-based virtualization, incomplete definition of interfaces and lack of fault and performance framework [5]. Our research addresses these problems by developing a framework, called HYPER-VINES, which crystallizes useful information from high dimensional data, through machine and deep learning models, for improving availability and reliability.

Internet of Things (IoT), multi-cloud hierarchy and virtual network services would be key to successful future deployment of critical applications, like pervasive next generation healthcare. Use of these technologies would become imperative to improving emergency care and diagnostics while controlling investments. However, the use of IoT, clouds and virtualized network services increase the attack surface and provide the opportunity to adversarial players to perpetuate malicious attacks, leading to increase in patient morbidity and mortality. Our research shows that innovative use of hierarchical deep learning models can make data, while in motion among the IoT, cloud and visualization domains, secure.

Our thesis is that by understanding the challenges in creation, performance, availability and reliability faced in the deployment of applications like virtual network services over multiple cloud systems, it is possible to apply innovative techniques of dynamic platform analysis, efficient placement algorithms and fault and performance management to get successful virtual carrier deployments. This new networking paradigm enables critical applications like pervasive next generation healthcare if data in motion in the cloud hierarchy is adequately secured.

The rest of the dissertation has the following structure:

- Chapter 2 brings out the challenges faced in the deployment of carrier services and critical applications like healthcare on multi-cloud systems.
- Chapter 3 reviews the related works to discuss how far these challenges have been addressed and the motivation that they provide for the work that has been done in the dissertation.
- Chapter 4 introduces the methods developed for optimization of management platforms that controls the lifecycle of virtual resources obtained from multiple clouds.
- Chapter 5 discusses virtual network function placement, in multi-cloud systems, to meet availability, performance and other service level agreement (SLA) parameters.
- Chapter 6 develops the HYPER-VINES framework for reliable fault and performance management system for improving availability in the virtualized carrier network services.
- Chapter 7 brings out the security issues in IoT and cloud based healthcare network and describes our work on the threat model, architecture and development of merged hierarchical deep learning model with layer reuse.
- Chapter 8 summarizes the work and discusses future research directions.

Chapter 2

Challenges in management and security of multi-cloud applications

In the pursuit of creating virtual network services that have the performance and availability of the traditional physical network services and security level that allows deployment of pervasive IoT, cloud and virtual network services based health-care set-up, we had to confront a number of challenges. In this chapter we discuss the challenges and roadblocks to the large-scale deployment of cloud based virtual network services and to critical services like next-generation healthcare that use this virtual infrastructure.

2.1 Optimization of the Multi-Cloud Management Platform

Multi-cloud platforms are software systems that automate provisioning and life-cycle management of virtual resources obtained from multiple clouds. Multi-cloud platforms are particularly complex as they are expected to dynamically allocate resources over geographically distributed clouds, creating, scaling, migrating and destroying virtual resources frequently during normal operation of the deployed applications. They generally use multi-threading for concurrent execution of processes to ensure that applications and services get the amount of resources they require and the desired performance is achieved. Many modules of a Multi-cloud Management and Control Platform (MMCP) may themselves be hosted on virtual cloud resources, along with the applications they manage. Optimization of MMCP is important for two reasons. Firstly, modules of an unoptimized MMCP may consume disproportionately higher amount of virtual resources leading to higher cost. If any of the modules, of such a system, does not work efficiently, overall performance suffers resulting in sub-optimal behavior in terms of not only computing, storage and networking resource usage but also greater power consumption [9]. Secondly, an unoptimized MMCP may not achieve the best possible placement of the target application. Stated briefly, a non-optimized MMCP leads to inefficient resource utilization, increased cost of deployments and likely breach of the service level agreements (SLA) between the cloud service provider (CSP) and the carrier.

The main challenge in this regard, therefore, is to optimize the software based MMCP for the environment to which it is applied. In this situation it makes sense for the ASPs and ISPs to require that the control and management software they use, for allocating virtual resources to their applications, are optimized for their situation. Optimization can be expensive and time consuming, as it needs apriori understanding of a platform's behavior in execution. The effort should, therefore, be to reliably confirm the factors that need optimization before launching into one. To the best of our knowledge no other work has focused on behavioral analysis and optimization of multi-cloud management and control platform [7], [8].

2.1.1 Extracting Behavioral Data

Optimization requires collection of data through intensive but unobtrusive testing. Traditional testing, which uses a limited set of sample inputs, does not guarantee semantic coverage. Symbolic testing can be done to tackle the latter [10]. In symbolic testing a program is symbolically executed for a set of classes of inputs rather than individual samples. However, industry case studies show symbolic techniques do not scale because of path explosion and they give high false positives [11]. Use of concrete test data, instead of symbolic, avoids false positives but only covers one execution path depending on the test data [10]. Another common method, static analysis is scalable but gives relative assessment of execution time and is prone to false positives and negatives. It becomes computationally expensive to explore all inter-leavings [12]. Dynamic analysis gives absolute execution times but may not offer full coverage. However, it is a technique that has potential in virtual environments for giving a deterministic assessment of execution [13]. In their survey of current practices and opportunities for improvement of software testing methods and tools (STMT), the authors conclude that the usage of STMT is low, and even where they are used the use is limited in scope [4]. Thus, there is a gap between available methods and the industry needs. We will see some more related works in Chapter 3 to get an idea of the current state-of-the-art.

In our assessment, the problem of optimization of complex software like MMCP cannot be effectively handled with any single existing technique. This provides us motivation for developing a combination methodology that uses dynamic execution for data collection and initial discovery of factors, followed up with full factorial analysis to decide with certainty the necessity of optimization using these factors. We have evaluated our method on OpenADN (Open Application Delivery Network) multi-cloud test-bed at Washington University in St Louis with good results. The work done for this part of research has been published in [7] [8].

2.1.2 Analyzing Platform Performance

Components of an ASP or an ISP application deployed over virtual resources from many clouds may be run at different times on different virtual machines. These virtual machines are migrated from server to server and cloud to cloud, as the platform optimizes cost and performance of the hosted application. The platform and the application software may be split on different, and geographically dispersed machines and their communication dynamics may change frequently. Additionally, many processes have to take place concurrently with some waiting for others to provide inputs. In order that the platform software keeps making progress on the whole, the processes are run in different threads [15]. Any method of analysis that attempts to capture interactive behavior of all the threads may result in exponential analysis times. Unlike the case in the physical environments, the use of tools based on the processor performance counters has not advanced significantly in the virtual ones [16]. As we shall see in Chapter 4, the reason for problematic behavior of guest machines sometimes lies in the platform or other guest machines. Diagnosing performance problems of software running in a virtualized environment is, therefore, an involved process [17]. Some work has been done to optimize data centers for performance improvement related to cost savings but that does not ensure optimized performance for the intended applications [18]. Optimization of performance of cloud management systems and applications running on cloud resources is in a nascent stage and quite challenging.

2.2 Placement of Virtual Network Services in a Multi-Cloud Environment

Assuming that the multi-cloud platform has been optimized for placement of carriers' virtual network services, it still remains a challenge to allocate resources and create virtual network functions (VNF) in such a way that meets the stringent requirements of the carriers' service environment. One of the biggest challenges, in deploying network function virtualization (NFV) based virtual network services (VNS) over multiple clouds, is low VNS performance. The traditional physical networks have had strict performance standards stipulated by standards organizations like International Telecommunications Union (ITU), International Standards Organization (ISO) and Internet Engineering Task Force (IETF). These standards prescribe stringent control over performance parameters like latency, jitter and packet loss [22]. The availability requirement is of the order of five nines (permissible downtime of just 5 minutes and 15 seconds per year). In the current state of technology, VNSs are presently not able to meet all of these standards. In fact, there is a general concern regarding the current technological capability to extract carrier-grade performance from NFV-based services [19] [20]. The Internet Engineering Task Force (IETF) has identified performance and guaranteeing the quality of service as open research areas and technology gaps in NFV [21].

It is important to study these problems, as carriers perceive NFV as a disruptive technological development that has the potential of delivering them from the problems of the traditional physical networks. The combination of NFV and cloud computing has great potential for carriers that would help them in making their networks agile i.e., create new services or phase out old services with relative ease. It also allows them to move from proprietary to open source, the flexibility of scaling and de-scaling, having points of presence closer to the users and avoiding a single point of failure. It is expected that these two powerful paradigms would evolve together to support the requirements of VNS.

As far as the challenges of creating and configuring VNFs in a multi-cloud environment are concerned, there are a number of reasons why these software versions of the network functions, do not give a performance that matches the performance of the specialized physical appliances used in the traditional networks. Besides the limitation of software running on general purpose hardware, a reason for lower performance is the reduction in control that carriers can exercise when the network appliances move from their own switch rooms and transmission centers onto the Cloud Service Providers' (CSPs') VMs. Additionally, the newfound ease of creation, destruction, migration, and scaling of virtual resources, and opportunities for indiscriminate virtualization proliferate. All of these issues cause performance of the VNS to deteriorate. Previous work has also shown that virtualization may lead to abnormal latency variations and significant throughput instability [18]. In their infrastructure overview, ETSI has indicated latency and throughput constraints as the discouraging factors for the use of public clouds for hosting NFV. Even though researchers have proposed ways of improving the performance of virtual network functions legitimate concerns still remain [23] [24].

In the VNS game, carriers and CSPs may not always have a cordial relationship. It is challenging to co-optimize their conflicting goals when they collaborate to provide VNSs. Carriers look for standards-grade performance and availability at the minimum cost and in the desired time frame. So, not to take any chances, they incorporate these in their Service Level Agreements (SLAs) with the CSP. On the other hand, the CSPs aim to maximize the utilization of their physical and virtual resources to improve their profit margin. All said and done, the advantages of the VNSs are far too important for researchers to ignore.

We now summarize some of the key outstanding problems in the placement of carrier VNSs in a multi-cloud environment, which we have attempted to handle in the Predictive Adaptive Real Time (P-ART) placement framework that we have developed for this purpose. We discuss this framework in detail in Chapter 5.

2.2.1 Achieving Dynamic Placement in Multi-cloud Systems

Some carrier services may be fairly static, e.g., fixed voice network. Thus, the number of instances of VNFs and link capacities required only change slowly over time. On the other hand, many services may be extremely dynamic, requiring a change in the number and types of VNF instances, re-dimensioning of links and changes in the offered features of the service, very frequently. An example of such a service is the intelligent network service like televoting in a TV reality show. Different reality shows may require different features and the number of voters may swing unpredictably during the voting window. If the CSP only offers largely static placement, with reactive and relatively slow modifications, then the carrier will not able to meet the program's requirements.

We conclude that both, the dynamic and static services would require the CSP to scale VNF capacities or links, albeit at different rates. Dynamic services may be more demanding in terms of types and number of instances of VNFs and link resources and may even require migration of VNFs from one cloud to another to be able to continuously meet the cost and end-to-end latency constraints. A dynamic placement algorithm, that monitors the SLA parameters and proactively causes changes in the amount of resources and the combination of clouds to meet all the requirements, is still a challenging issue.

To meet the problems described above, the MMCP should be able to make optimized initial placement as well as reconfigure Service Function Chain (SFCs) to continuously meet the SLA conditions. Additionally, it is important to have fast placements with high success rates for near real-time reconfigurations. It is know that solving the placement problem is NP-hard [24]. Researchers commonly set it up as an integer linear program (ILP) optimization problem, with one or more objectives optimizing resource level parameters like compute usage, storage usage, power consumption along with constraints like capacity and affinity [25]. Solving these formulations takes minutes to hours becoming unacceptable for dynamic real-time applications. Algorithms like greedy placement and heuristics like first fit decreasing (FFD) have been proposed to limit the time a linear or a quadratic programming solution takes to give a reasonable solution. Achieving dynamic placement still remains an issue.

2.2.2 Optimizing the SFC Performance

When the data are multi-modal, optimizing placement of individual VNFs may not achieve the global minimum. Placing SFCs as a unit yields better results. The opportunity to achieve the global minimum for the parameter being optimized, e.g., latency, is available when placing the SFC. If sufficient resources are not available to implement full-service chains, then the request may be rejected or, if the policy permits degraded service (for instance without firewall) is provided [26] [27]. In our work, we only consider complete SFC placement. The case where the customer accepts degraded performance due to low-capacity chain placement or partial functionality due to incomplete chain placement is left as a future work

2.2.3 Meeting the Cost and Quality of Service Constraints

From the carrier's perspective, the placement problem boils down to placing network functions to meet the cost and quality of service constraints like latency. The placement problem needs to be solved at the time of commencement of the VNS, and repeated again during its operation, to ensure that the carrier requirements are continually met. Performance criteria may vary from one application to another. For the carrier services like voice, broadband, and content delivery some of the common factors are jitter, packet loss, latency, and throughput. ITU standards for the quality of service parameters, in carrier networks, are available in [28]. Latency is one of the most important criteria, and we have taken that as a reference performance parameter. The framework can be extended to include other criteria as well. We will discuss more about this in Chapter 5.

We find that the placement algorithms currently built into the MMCPs do not take into account the time taken in planning the service chain deployment, creating virtual resources and booting them. In addition time take for chaining the functions, acceptance testing and commissioning is also not accounted for. To be correct, any placement needs to take into account the expected latency at the time of placement and not at the time of planning. The method that we have evolved, consists of prediction of latency at the time of commissioning so that placement algorithm can ensure meeting latency requirement when the service becomes operational. Additionally, an iterative random cloud selection method is proposed to select the least cost clouds, which meet the latency requirements To make latency predictions more realistic, the diurnal variations in traffic are taken into account through the windowing concept. Short-term variations in traffic need to be compensated through updation of trained models. Our work on Cost optimized latency aware placement has been published in [29]. A more comprehensive account of the work, incorporating most the ideas that we have incorporated in the solution to the placement problem has been accepted for publication by Elsevier's Computer Communication Journal in March 2019 [121].

2.2.4 Speed and Accuracy of the Placement

Carriers want short placement and reconfiguration time so that the solution can be useful in an operational network. CSPs want the solution to have the high success of placement requests so that the utilization of the virtual resources increases. When the algorithm cannot place, despite the availability of resources, CSPs lose by way of unused resources and possible breach of SLA. None of the solutions reported in literature have tackled both speed and accuracy together. As we shall see in Chapter 5, the methods that we have evolved place thousands of functions in a sub-minute time frame.

2.3 Fault and Performance Management In Multi-cloud Virtual Network Services

In sections 2.1 and 2.2 we have discussed some of the challenges that creation of NFV based virtual network services bring. One may be tempted to question the wisdom carriers for taking all this trouble and not continuing with the physical networks that they already have. At the risk of repetition it can be said that NFV allows carriers to create software based virtual functions and use these to create VNSs that have great agility and flexibility. A big incentive is the possible reduction of the network deployment cost [31] [32] [33]. It is, therefore, no surprise that NFV is being regarded as one of the most important developments of this decade for the communication networks. The Gartner Hype Cycle 2018 describes NFV and network performance as the key technologies, alongside the Internet of Things (IoT) and 5G [30].

This brings us to the third challenge of deploying virtual network services on multi-cloud systems. The performance and availability of the virtual network services do not match those given by the traditional physical networks. It is important to meet five nines availability and standards based quality of service that that traditional networks offer. In our research we have identified gaps in definition of interconnection among the involved platforms, lack of a well defined 'Fault, Configuration, Accounting, Performance and Security (FCAPS)' framework and the complexity of virtualized network services as the primary reasons for not meeting the availability requirement. Traditional carrier networks follow rigorous standards relating to FCAPS, like the one prescribed in the ITU Telecommunications Management Network recommendations [34]. There is no such framework for virtual networks across multiple clouds to ensure availability, stability and quality [37] [38]. ETSI Industry Specifications Group (ISG) specification on NFV resiliency requirements provides a list of faults and their remediation and service continuity but the focus of this work is limited to failure modes, which are introduced by moving network services to a virtualized infrastructure [39]. Academic research on fault and performance management of virtual network services has been scarce.

2.3.1 Gaps in the Interface Specifications

A number of players have to co-operatively interact to make VNSs work and achieve the desired performance and high availability. In the simplest cases these players would be the carrier, CSP and the NFV service provider. Each of these players would interact with the service through their management and control platforms. As far as the NFV is concerned, the two main standardization efforts are the ETSI-NFV and the ONAP project of Linux Foundation. Both of these are relatively new and their standards are not yet fully compatible. At the service level, inter-platform responsibilities and communication among them have not been clearly defined. As a result any service affective event may not be adequately handled causing performance and/or availability to suffer.

2.3.2 Incomplete Definition of FCAPS

Traditional networks are built to the stringent quality of service (QoS) norms defined by FCAPS standards like ISO Common Management Information Protocol (CMIP) and ITU Telecommunications Management Network (TMN) M.3010 and M.3400 recommendations [34] [35][36] [40]. Such norms are still to be fully defined and met for the VNS deployments. Traditional failure detection mechanisms cannot be directly applied to NFV environments as they lack the sophistication of handling virtualization layers. The variety of Fault and Performance (FP) issues that can affect the carrier networks is large and difficult to detect, diagnose and localize [41] [42] [43]. When we add to this the virtualization and the cloud computing layers, the number of ways faults can affect the virtualized network far exceeds that of their physical counterparts. New methods would be required to deal with faults in VMs or VNFs, which cause the VNSs to behave abnormally, even if the underlying hardware is fault-free.

2.3.3 Complexity of VNSs

In NFV-based VNSs, faults may occur for many more reasons compared to traditional physical networks. The cloud infrastructure consists of virtual resources such as virtual machines, virtual storage, and virtual network links. These virtual resources are created on shared physical resources like server hardware, system software or network links, using virtualization software (e.g., Hypervisors). Other virtual layers like VNF and SFC are built on these physical and virtual infrastructure layers. One reason why virtual resources may fail is because of the failure of physical resources. Even if the physical resources are operational, the virtual resources may themselves fail [44]. Taking this argument a little further, even if both physical and virtual resources are healthy, the VNFs, like routers, instantiated on these virtual resources can develop algorithmic problems causing VNSs to malfunction or totally break down. The myriad levels of dysfunctions make handling of FP issues in NFV over clouds more abstract and complex. VMs are managed by the CSPs and the VNFs by both the NFV operator and the carrier. This overlapping responsibility makes it difficult for the traditional FP systems to deal with problems in the virtualized networks.

2.4 Security in Next Generation Healthcare

The final challenge that we tackle in this dissertation is the security of applications such as the next generation healthcare that use VNS, IoT and cloud systems [45]. Among the problems of the present healthcare systems are inadequate response to medical emergencies, delays in diagnosis of acute cases, insufficient monitoring of chronic patients, readmissions and too many preventable errors. Preventable errors are the third leading cause of death in the USA [47]. Modern healthcare is capable of reducing the cost of medical care and the resources required [48] [49]. To achieve all this medical care of critical, hospitalized or ambulance bound patients has to rely heavily on technology. It is believed that the use of technology will reduce patient mortality and morbidity by reducing diagnostic response time and improving diagnosis by reducing chances of human error [46]. On the flip side, increasing use of technology has its own security implications.

In the next-generation healthcare architecture, all the subsystems viz., the virtualized network, the IoT domain consisting of sensors and actuators, the multi-cloud domain and the visualization domain, would be intricately enmeshed in cyberspace and become vulnerable to malicious adversarial activities of various kinds. Attacks like device takeover or tampering, mutilation or theft of data flowing among domains or stored in the clouds, change in patient history and planting of ransomware, may result in morbidity and mortality. It is, therefore, important to ensure confidentiality, integrity and availability of data as it flows among the domains of the healthcare system.

2.4.1 Inadequacy of the Current Models

Effective detection of attacks and anomalies has to take into account hundreds of markers, alerts, measurements and interaction patterns across a large number of sensors and other devices in the IoT domain, a hierarchy of clouds, and the visualization devices in the user domain on a continuous basis. Threats exist from inside and outside the healthcare network from cyber criminals, policy negligence and malicious intent of insiders. Traditional security solutions rely on signature-based defense at the boundaries of the functional areas of healthcare. With the healthcare being modernized, the increasing attack surface and hyper activism on the part of malicious players, the signature-based defense is evaded through means like changing tactics, encryption, polymorphism or obfuscation [8]. Traditional security approaches involve manual filtering of exceptions and analyzing each alert. This becomes time consuming and alerts need to be selectively examined because of too many false negatives in the traditional methods. Every medical condition that is present and not detected puts patients at risk. Some traditional systems may not be effective when attacks involve multiple tactics, multiple end-points and change their nature. Additionally, if the threats are continually changing then detecting them across fuzzy functional boundaries will be even more difficult [50].

Because of the problems mentioned above the traditional systems are giving way to machine learning based anomaly detection. Machine learning has been applied to security in healthcare in many forms. Some of the classifiers that have been used are: support vector machines, decision trees, naïve Bayes, K-nearest neighbors and random forest. In machine learning based intrusion detection system, for instance, the idea is to capture underlying statistical features of data and use them to detect any malicious attack [51]. However, the machine learning solutions that have been applied are shallow machine-learning systems that have their own problems. In the presence of high dimensional and high velocity data, these methods get bogged down by the need for manual feature extraction and their high false positive rates, resulting in the risk of overmedication or unnecessary procedures [111].

2.4.2 Security of Data Flows Among Domains

In this work we focus on the patient data generated by sensors and flowing between IoT, cloud and visualization domains of the next generation healthcare. Any mutilation or pilferage of data and device capturing may compromise the system and result in patient morbidity and mortality. It is challenging to detect anomalies produced by these intrusions because of data being large and high dimensional and adversaries employing advanced techniques that change spatially and temporally. Many established procedures do not work well in the ensuring environment. Signature based systems cannot detect previously unseen attacks. Centralized intrusion detection systems at the edge of the IoT domain cannot protect flows in the cloud hierarchy. Also high volume and dimensionality of data, manual feature extraction and lack of unknown threat datasets render shallow machine learning solutions less effective. In the next chapter we discuss the state-of-the-art and the challenges that remain. In chapter 7 we detail the method we have evolved to protect the data in motion.

Chapter 3

Addressing the multi-cloud management and security challenges

In this chapter we discuss the outcome of the survey of the state-of-the-art and the extent to which these works address the challenges described in Chapter 2. I then present how these unaddressed problems provide motivation for the work that I have carried out in this dissertation

3.1 Optimization of Multi-cloud Management and Control Platforms

It has been mentioned before that un-optimized platforms result in slow speed of placement and reconfiguration, inefficient resource utilization and increased cost of deployments [8][15]. Some of the reasons that were brought out, in the last chapter, are the complex nature of these softwares and extensive use of techniques like polling for asynchronous working and multi-threading for concurrent execution of a number of activities like initiating workflow in the cloud manager, allocating resources for a proxy to connect external services and connect hosts [15]. In this situation any method that attempts to carry our exhaustive analysis may result in exponential analysis times. Additionally, software testing methods and tools, like use of performance counters and exhaustive testing, have not advanced sufficiently for the virtualized environments.

[16]. Datacenter optimization for cost has been reported but this does not ensure optimized performance for the application that the ASP or ISP is intending to onboard [18]. Given that the fully virtualized and distributed platforms like OpenADN uses the mentioned techniques and more, most conventional testing methods like profiling, characterization, and modeling methodologies do not work well. None of them have been shown to provide definitive help in pinpointing the sections of code that should be optimized. To get a sense of where we currently are, we look at some of the techniques that can be applied to a distributed, multi-threaded system [52].

3.1.1 Collecting Behavioral Data

For collection of behavioral data, the techniques studied can be broadly classified in different ways: static and dynamic or continuous and discrete or intrusive and non-intrusive. These are not necessarily separate categories and one may be classified under the other. For example, dynamic testing can be continuous and intrusive. Limited use of some of these techniques has been made to study guided datacenter cost optimization in [53]. In this context, the authors argue that performance and utilization characteristics are critically important, because even minor performance improvements translate into huge cost savings. Their contention is that the traditional performance analysis is complicated for datacenter applications and it is easier to monitor the datacenters on live traffic. Palanisamy et al. have proposed Cura for provisioning cost-effective MapReduce services in a cloud [13]. It leverages MapReduce profiling to automatically create the best cluster configuration for the jobs. They have reported up to 80% reduction in cloud compute infrastructure cost with up to 65% reduction in job response times for Facebook like workloads. In [18] the authors argue that the hardware measuring features are inflexible, limiting the types of data that can be gathered. On the other hand, instrumentation based profiling can provide more flexible and targeted information gathering at the cost of some overheads. In order for these techniques to be useful for datacenters, overhead needs to be contained to less than a few percent in terms of both throughput and latency. The authors propose instant profiling, an instrumentation sampling technique using dynamic binary translation. In this technique normal execution is interleaved with instrumented execution. They have achieved less than 6% slowdown and 3% computational overhead on average.

Static analysis: involves careful examination of the programs to diagnose defects that will prevent correct execution. It builds an abstract representation of the program behavior and examines its states. It aims to check all paths and consider all nondeterministic choices. This can be done manually (which ends up being too time consuming) or by tools (which often leads to too many false positives). The kind of errors it looks for is undefined variables, interface between modules, unused variables, syntax violations and dead code. Static analysis can be done using model checking in which it computes the run-time states of the program without running the program [54]. The generated run-time states are used to check if a property holds. If the program has a finite number of states, it is possible to do an exhaustive analysis [55]. This work also mentions that models of multi-threaded systems can be used to explore all feasible inter-leavings and loops exhaustively to ensure correctness properties [56]. However, the multi cloud platforms are complex and may have a vast number of feasible inter-leavings, making model checking computationally expensive. Another shortcoming of static-analysis techniques is that they give an assessment of relative time and temporal ordering and do not give absolute time of running code [57]. For assessment of absolute times, it would be necessary to perform dynamic analysis [58].

Dynamic analysis: This type of profile guided testing allows observing system behavior while it is in execution. It is an active form of profiling in which the system being measured explicitly generates information about its execution parameters. Conversely, passive profiling relies on explicit inspection of control flow and execution state through an external entity, such as a probe or modified runtime environment. Dynamic analysis obtains detailed and precise info for a single test case. It provides no guarantees for other runs. Multiple runs can be resorted to, each testing different paths. Automated tools for dynamic analysis are limited in scope and do what they have been programmed to do. In [52], it has been mentioned that dynamic profiling provides ways to measure the absolute time of events like various function calls or the time spent by the CPU in a particular function. Three main families of dynamic profiling techniques are code instrumentation, statistical sampling and concurrency profiling.

a) Code Instrumentation: A set of additional instructions, called an instrument, is injected into the target program. When the instrumented code is executed, it generates timing and frequency of the events as they happen. Some instrumentation systems [56] count function activations while others count more fine-grained control flow transitions. This method can, thus, provide an absolute measure of these events. Instrumenting a program can cause changes in the performance of the program, potentially causing inaccurate results and has to be carried out carefully in a controlled manner.

b) Statistical Profiling: In this method the program state is randomly sampled when the program is in execution. This involves recording a sample of values of the instruction register, program counter, stack, etc. to apply statistical techniques to these samples to deduce how much time is being spent in different parts of the program. This method is not as intrusive to the target program as the instrumentation method. It can show the relative amount of time spent in user mode versus interruptible kernel mode, such as system call processing, and also the user time out of the total execution time. In the OpenADN environment this could, for example, provide valuable information on whether optimization should at all be attempted.

c) Concurrency Profiling: Concurrency profiling can additionally be used in multithreaded applications. Resource contention analysis collects detailed call stack information every time that competing threads are forced to wait for access to a shared resource. Concurrency visualization also collects more general information about how a multithreaded application interacts with itself, the hardware, the operating system, and other processes on the hosts. It can help locate performance bottlenecks, CPU underutilization and synchronization delays [47].

Distinction is also made between deterministic and statistical profiling of software. In the deterministic technique all function calls, function returns, and exception events are monitored, and precise timings are obtained for the duration of these events and the intervals between them. OpenADN is largely written in Python. In Python, since there is an interpreter active during execution, the presence of instrumented code is not required to do deterministic profiling. Python automatically provides a *hook* (optional callback) for each event. Call count and time consumption statistics can be used to identify hotspots in code, which would be potential candidates for optimization. In the statistical technique the instruction pointer is randomly checked to deduce where time is being spent. It can, therefore, only provide relative indications of where time is being spent.

The existing works do not describe any definitive method or combination of methods for testing virtualized multi-cloud platforms. This motivated us to develop a combination of techniques, which is discussed in Chapter 4.

3.2 Placement of Virtual Network Services in a Multi-cloud Environment

Since solving the placement problem is NP-hard [24], algorithms like greedy placement and heuristics like first fit decreasing (FFD), have been proposed to limit the time a linear or a quadratic programming solution would take to give a reasonable solution. However, most of the work done so far carries out static and reactive virtual machine placement that requires tenants to initiate changes making the process manual [59]. When these methods are in place, there would be times when the placement will not meet the cost or performance criterion. More recently dynamic and proactive techniques have been applied. Another shortcoming of many methods is using VNF/VM as a unit for placement and optimization rather than the SFC. It has been shown that that placing SFCs as a unit rather than individual functions yield better results [24]. The common method of setting up an ILP with the objective of optimizing resource level parameters like compute usage or power consumption under capacity constraint takes minutes to hours even with efficient solvers [59]. This would be unacceptable in real-time or near-real time applications of the type we are focusing on. Carrier networks have stringent latency requirements, while in critical healthcare even seconds could mean the difference between life and death or the extent of morbidity suffered. Dynamic placement with cost optimized performance aware placement of VNFs is still an open question [61]. We propose a combination of techniques to optimize cost, keep latency below threshold and carry out fast placement.

3.2.1 Methods Based on ILP and its Variants for Optimization

In [62] and [16] the authors argue that unlike most other works they have considered QoS/SLA along with resource requirement of network services. They show that the virtualization overhead increases with traffic load and the number of VMs due to factors such as scheduling delays, context switching, and flow routing. The authors include virtualization overhead while setting up their Mixed Integer Linear Programming (MILP) model to optimize resource usage while guaranteeing latency requirements. The model optimizes the cost including the utilized processor, memory and physical links under the latency constraint of maximum round-trip time. It is seen that for a network with 28 nodes and 41 links the model takes about an hour to arrive at an optimum solution. The authors in [63] use an MILP model to optimize network latency and increase the acceptance rate of strict delay requirements. The evaluation is single cloud specific and scaling from 5 VNF to a large network, for delays are unclear. Also, the algorithm chooses a more expensive path to ensure a minimum delay. An intuition, that does not require proof, is that delay will be more with high bandwidth requirement, or when more requests seek the same link. In cases where the number of requests is high, the solver is not able to find an optimal solution in the joint delay and routing cost optimization methods. The solution for the optimal chaining and routing with MILP limits the scale of the problem.

3.2.2 ILP and Heuristic to Speed up the Solution

In [64], the authors optimize the number of physical machines (PM) used, using an Integer Linear Programming (ILP) model. They take into account the time-varying workloads while instantiating VNFs in PM. A two-stage heuristics solution has been suggested to solve the ILP, with a correlation-based greedy algorithm as the first stage and a further adjustment at the VNF in each SFC as the second. The simulation demonstrates improved utilization of network resources and reduced number of PMs compared to the benchmarks. This and some other works presume multi-tenant VNFs to improve utilization. Multi-tenancy allows CSPs to reduce their costs but carriers would usually request exclusive VNFs hosted on exclusive VMs because of security and performance concerns. In [65] the authors propose placement of VNFs in the edge clouds to minimize end-to-end latency. Using an ILP model, the authors show that cloud-only deployments gave more than 3 times more latency than cloud-and-edge deployments. The absolute times for initial placement and for each re-configuration are not known. They also present a way to dynamically re-schedule the optimal placement of VNFs based on temporal network-wide latency fluctuations using optimal stopping theory. Scheduling re-optimization may reduce latency violations, but they may require an increased number of migrations. Periodic migration also has a problem, as it requires human intervention to decide on the periodicity of tuning. The authors suggest a method using optimal stopping theory to select the right time for placement.

3.2.3 ILP and Heuristics for Comparison

In [66], the authors consider an IoT-edge cloud-main cloud scenario in a dynamic multi-user situation. The authors set up an MILP model to minimize the end-to-end communication delay while keeping the cost to the minimum. However, they realize that the MILP formulations rapidly increase in complexity and take a long time to give an optimum solution, as the problem becomes large. To counter this, the authors also propose Tabu search for placement and chaining. They find that the MILP method are 200 times slower than the Tabu Search. The authors in [67] solve VNF placement and chaining problem using ILP and also propose another method called Cost-efficient Centrality-based VNF Placement and chaining algorithm (CCVP). The objective is to minimize the cost by finding an optimal number of VNF instances and their locations for handling the required traffic. To simplify, they assume that the network provider is the owner of NFVI so concerned factors are under its control. The CCVP is based on the Betweenness centrality algorithm. The high centrality indicates that a vertex of a graph G can reach other vertices on relatively short paths. This results in lower network cost. They show that the overall cost of their method is close to ILP. It should be noted that processing delays and link bandwidths are not considered in the analysis. In [68], the authors pursue the objective of optimization of energy consumption as an ILP model. This purportedly gives a reduction in the operational cost of the placement. They also propose a near-optimal approximated algorithm to solve the problem using the Markov approximation technique. They show that their algorithm can achieve the performance arbitrarily close to the global optimum. Simulation results show that the algorithm saves up to 14.84% energy consumption compared with previous VNF placement algorithms.

3.2.4 Non-ILP Heuristic Solutions

In [69] the authors presume sharing of VNFs among different service chains. It should be noted that while sharing may improve VM utilization, it might consume more link bandwidth because these chains may need to go through a longer path in order to reach the shared VM. From carriers' point of view this arrangement may give rise to security issues as well as make it difficult to control latency. The authors contend that most of the existing works are mainly targetted on improving VM utilization, without considering the required bandwidth resources. This work has examined the joint VNF placement and Path Selection problem, so as to maximize the served traffic demands. In [70] the authors discuss a proactive placement model in the context of a content distribution network (CDN). They argue that VNF chaining and placement affect QoS, and formulate an optimization problem to find the optimal number of locations as well as efficient chaining such that the CDN cost is minimized and QoS is satisfied. The authors set up the problem as a bin-packing problem that involves selection of bins (surrogate servers) and dropping the items (VNFs) into them. The authors conclude that while their solution gives fewer servers, it may give a high communications cost. In [71], the authors investigate the optimal placement of virtual resources to minimize the average response time in mobile edge computing (MEC) environment with a capacity constraint on the edge network. They use OEPA (Optimal Enumeration Placement Algorithm) as a benchmark to compare Latency-Aware Heuristic Placement Algorithm (LAHPA), which has lower computation complexity, Clustering Enhanced Heuristic Placement Algorithm (CEHPA) to enhance the performance of LAHPA, Substitution Enhanced Heuristic Placement (SEHPA). SEPHA turns out to be better than LAHPA. CEHPA and outperforms LAHPA and both are better than the general Greedy Placement Algorithm. The authors in [72] describe a dynamic placement algorithm based on traffic variations that saves operational expenditures. Their algorithm consolidates VNFs in the fewer possible number of network nodes while maintaining low blocking probability and guaranteeing latency targets to the supported services. They reuse VNFs, select VNFs based on locality and activate them based on the shortest path. The authors claim that their algorithm is able to balance the trade-off between minimizing latency violations, decreasing blocking probability and reducing operational expenditure. The success rate of the algorithm has not been mentioned. The authors claim 50% saving in telecom operators cost.

3.3 Fault and Performance Management In Multi-cloud Virtual Network Services

The traditional fault and performance management systems do not perform well when applied to the virtual environment in which virtual resources can be constantly scaled, migrated or destroyed. The commonly used rule based systems of physical networks are rigid and not good at unseen faults. With the network functions virtualized, faults propagating through layers change their semantics and thus a correlation between an original fault and an observed system disorder is not obvious. Though a number of standards organizations, including ETSI, 3GPP, Broadband

Forum, IETF, ITU-T SG 15, MEF, ONF, OPNFV and TM Forum, are working on FCAPS in virtualized networks there are still many gaps in the specifications [73] [74]. The metric list needs to be supplemented with issues related to specific network services, e.g., ‘continuous dial tone’ or ‘line card fault’ in relation to fixed phones and similarly ‘call drops’ or ‘weak signal strength’ for mobile networks and, ‘WAN light flashing’ or ‘low data rate’ for ADSL broadband networks. Some of the efforts, like OPNFV ‘Doctor’ for fault management of network services are based on OpenStack. OpenStack telemetry and alarming and are not geared up for multiple clouds and it lacks some capabilities that are critical requirements for an NFV platform [75]. ETSI has specified the architecture of NFV-MANO [76], which will interface with the CSP infrastructure and the carrier's Operation Support System (OSS) (including Network Management System) for fault detection and localization. The interfaces among these platforms have not been fully defined. There is multiplicity of responsibility in handling faults. Escalation of fault indicators from lower to higher layers has also not been defined. Predictive fault management can prove to be effective for distributed real-time systems but is yet to be effectively employed. In the machine learning assisted methods the problem is of availability of real network training datasets. Publically available datasets like UCI repository [77] and Stanford SNAP [78] do not have datasets relating to faults in telecommunications networks. Deep learning has been found to be effective in image, speech and text recognition as well as fraud but is still new in network anomaly detection.

Relevant to FCAPS are ETSI specifications of NFV resiliency requirements [79] and service quality metrics [80]. The former provides a list of faults and relationship between them while the latter gives VNF related metrics useful for quality of service. Both ETSI and ONAP describe the specifications for fault management support functionality [81] [82]. There has been extensive work on performance modeling systems for distributed Internet applications of the pre-NFV era, notably TIPME (2000)[83], Pinpoint (2002) [84] and Magpie (2003) [85]. TIPME helps in identifying and eliminating causes of long response times. Pinpoint uses data mining to correlate the behavior of each active user request with the past failures and successes to determine failed components. Magpie works on individual user requests and compares the observed behavior, with saved normal models, to identify anomalous requests and malfunctioning components. Recently, the ‘mPlane’ consortium of European telecom companies and academic institutions, has worked on developing a measurement plane for Internet and CDN (2013- 2016). The core of the project is ‘mpAD-Resoner,’ which uses machine learning to detect anomalies involving multiple flows or users. It compares the current distribution with stored average distributions [86]. The OPNFV Doctor project deals with the problems of the underlying hardware [87]. Most techniques relate to the IT environment with the three nines availability, as against the five nines required for carrier networks, is considered appropriate for most applications. These techniques work on the static or dynamic dependency models, which makes previously unobserved faults difficult to detect. These methods are limited by the implausibility of having up-to-date models in a dynamic environment. They deal with faults in the physical compute components. The NFV over cloud networks have a virtual network function layer that calls for a totally different set of markers, metrics and methods. These challenges have led us to develop the HYPER-VINES framework. We will discuss the details of this framework in Chapter 6.

3.3.1 NFVI Level Diagnostics

In VNSs, NFVI relates to the totality of hardware resources and the virtual compute, storage and networking resources created over these. The hardware component of the NFVI is in the domain of the CSP and generally inaccessible to the carriers. The methods in this category would rely on

VM level alarms and metrics such as compute load or memory leak. These techniques generally rely on the monitoring and diagnostic techniques for cloud computing resources used for IT applications. An explicit or implicit assumption would usually be that the higher level alarms and other markers, e.g., those at network function and network service level, would usually have corresponding host level alarms which can be correlated to detect and possibly localize network function and service level manifested and impending issues. A correlation between telemetry information from the CSP and the higher level alarms in the domain of the carrier, would have to be built up for diagnosing faults in the VNSs. Correlation of metrics with anomalies at the virtual layer has been applied by authors in [88]. The applicability of these techniques in a large distributed network needs to be studied.

3.3.2 Causal Inference Based Methods

These methods are also normally applied on VM level alarms like high CPU load and insufficient memory availability. The expectation here is that determining the causal relationship among them would help to get to the root cause of FP issues at the network function and service levels. The process involves looking for anomalous behavior based on VM level alarms, correlate alarms in pairs or clusters, determine causality, i.e., the effect of one alarm on the others and attempt to build causality templates that could be used for future alarms. The complex architecture and dynamics of NFV pose significant challenges from the point of view of causality inference. For instance, in [89], the authors carry out analysis of uncorrelated alarms in order to recover the pairwise causal relationship between them. To take care of the fact that higher-level faults (e.g., VNF or VNS levels) do not only depend on the pairwise relationship among VM level alarms, the authors propose clustering to infer multi-way causality templates. The patent documentation at [90] goes a step further and uses alarm data from different layers (e.g., NFVI and VNF). It takes into account the temporal proximity and the order of the alarm types in the clusters to make causality templates. It is challenging to exhaustively define all the relevant causal relationships among the markers and the faults.

3.3.3 Statistical and AI-based Methods

The large volume of operational data generated in an operational telecommunications network could emanate from within one layer or across multiple layers and possibly contain many different types related and unrelated markers. In such a complex environment, it would be difficult to analyze the available data to produce information that can be used to manage FP issues. This situation, thus, creates a perfect set up for removing humans from the loop and resorting to machine intelligence. In this category, there are methods based on machine learning and deep learning that could be used for the detection and localization of FP issues.

Researchers' interest in AI-based machine intelligence for the identification of FP issues dates back to the era of expert systems [91] [92] [93] [94]. During the intervening decades, the carrier networks have undergone changes in technology and form, but the interest in intelligent fault handling has persisted. We look at AI as a way to empower machines to mimic and outperform human intelligence. Machine learning is a subset of AI, chiefly consisting of statistical techniques that allow machines to exhibit behavior that improves with learning. Deep learning is a way to implement machine learning using neural networks with more than one level of non-linearity. When using neural networks for difficult tasks, complex relationship among variables modeled with several levels of non-linearity improves the generalization process [95] [96] [97].

VNSs are a new development and their deployment over multi-cloud is still to be explored fully. Many of the AI methods developed for intrusion detection have been explored, with varying degrees of success, for managing the FP issues in VNSs. Some researchers have applied AI methods directly to fault detection and, to a lesser extent, to fault localization. A very important reason for exploring AI for the problem of FP management for cloud-based NFV is the intractability introduced by the known gaps in the NFV specifications. Interaction among multiple domains, especially between the legacy carrier OSS and the MANO and the legacy OSS and the MMCP have not being fully defined. In this situation the learning methods of AI make the best use of the features learned from the available markers and can assist in FP management. It has been shown that learning methods give a way to relatively easily learn structure in the data and draw inferences [99].

Shallow machine learning algorithms, characterized by a single convolution stage, are suitable for cases where a large amount of labeled training data, including normal and fault cases, are available. They can derive intelligence from data and do not depend on experts to build complex interacting rules to derive patterns or models. Even dependencies, which cannot otherwise be explicitly modeled, can be learned. These advantages make them attractive for handling FP problems. In FP applications, machine learning methods can not only be trained with historical fault and performance data but can also be made to improve themselves as they operate and encounter new situations. This makes the machine learning systems adaptive and intelligent when they have been adequately trained, as they can generalize well from the training environment to the real-life situations. Use of different algorithms has been reported for detection and localization.

The authors in [100] use Artificial Neural Networks (ANN) for one and two alarms simulated scenarios. They show that in a simulated environment ANN provides better performance in comparison with the other implemented methods. The researchers in [101] propose a system for fault analysis and prediction in the telecommunications access network for the Rijeka area of

Croatia. Temporal decision trees for fault prediction in telecommunications networks have been used in [102]. As per findings in [103], fuzzy cluster means can be used to classify network faults. The current research indicates the possibility of advancing the state-of-the-art in FP management through deep learning structures.

In [104], the authors use the Random Forest machine learning method to detect performance degradations in the VNFs. However, these researchers have chosen to rely on virtual resource layer level features data like CPU consumption, disk I/O, and free memory based on their suitability to computing systems. Evaluation has been carried out in a centralized IMS system. Application of the proposed method to a highly distributed multi-domain network has not been reported.

The authors in [105] have worked on the premise that underlying all the VNF failures are the NFVI level failures like disk I/O or memory usage. They propose Self Organizing Map (SOM), a type of unsupervised learning neural network, for clustering the statistical data and analyzing them to detect the faults. In [106], the author mentions that machine-learning algorithms are expected to detect invisible failures and anomalies. However, more work is required to validate them.

3.4 Problems with Existing Security Approaches

Traditionally, for protecting internal resources like hardware software or network equipment, perimeter security is commonly used, which puts a barrier separating internal resources from intrusion from outside. The intrusion detection system can have a combination of firewall, anti-malware functionality and access control mechanisms to establish perimeter defense. Despite all these efforts, the strategy fails as attackers become more and more sophisticated. In cloud-based deployments the boundaries are fuzzy as resources gets distributed across geographically disparate clouds. Establishing a perimeter defense in such a situation becomes difficult. Virtual resources created for different tenants on the same physical resources may also prove to be a security problem.

Traditional security solutions rely on signatures, predefined database of known attack patterns or bursts of activity [107] [108]. Not only the traditional signature based systems are penetrable, they are not useful for unseen threat models [109]. Other traditional methods like firewalls to secure critical assets like electronic health records have often been breached by hackers through unhardened IoT devices. Security approaches involving manual filtering of exceptions and analyzing each alert become time consuming. Some traditional methods may not be effective when attacks involve multiple tactics, multiple end-points and change their nature [110]. In cloud computer network visibility is limited and east-west traffic cannot be easily monitored on virtual networks. Traditional methods involving Layer 3 and 4 packet filtering and security controls are not effective when virtual resources are leased across heterogeneous clouds.

To tackle the problems of unseen attacks researchers have examined machine-learning solutions in a bid to differentiate between ‘normal’ and ‘anomalous’ behavior. The shallow machine learning solutions may prove to be inadequate in the virtual healthcare environment. Data in healthcare are high dimensional and, with the shallow machine learning techniques, extracting relevant features requires human intervention. The curse of dimensionality renders the available data sparse and finding statistical significance difficult. On the other hand, using all the features would make the training process time consuming. Also, these methods usually have relatively high false positive rates for detection [111], which causes risk of over medication or unnecessary procedures.

3.5 Challenges Addressed by this Dissertation

Based on the discussion of the state-of-the-art and the remaining challenges, in this section we describe the challenges that we have addressed for different parts of the problem.

The combination technique for multi-cloud platform optimization: Both of the top-level challenges – collecting behavioral data of the platform and analyzing the data to produce actionable results- have been addressed in our work. These results have then been used to carry out optimization of OpenADN. In Chapter 4, we have discussed the results of optimization to see how far the two-level techniques have been successful in optimization of the multi-cloud management platform.

At the first level, we have used multi-level behavioral data collection techniques in the form of onion rings. It starts with a top-level view of the platform software and works down to the function and statement levels. Top-level analysis provides the overall CPU time utilization among the system and the user activities as well as the idle time. The detailed statement-by-statement profiling gives a tentative idea of the factors that prevent the platform from operating at the optimum level. At the second level, we use a technique for confirming which of the factors identified are significant. We use for the first time the two factorial analysis to confirm the factors before launching into an expensive full-scale optimization. We evaluate the model and find that our method can accurately identify the factors, taking care of which improves the performance.

The main contributions of this work in meeting the described challenges are threefold:

1) Elaborate how the behavior of a complex multi-cloud platform can be analyzed, while it is in operation, to obtain data for optimization, i.e., obtaining and using virtual resources from multiple clouds.

2) Evolve a methodology to examine the usefulness of the identified factors for optimization of the platform and avoid unnecessary optimization efforts.

3) Carry out optimization of the OpenADN platform using the result of the above analysis to show the usefulness of the techniques evolved.

***The P-ART framework for placement of virtual network services*:** The primary challenge for creating network services over multiple clouds is the placement VNFs of the service under several constraints. The placement has to meet the policies and objectives defined by the carrier and embodied in the SLA with the CSP. These usually include cost and quality of service parameters. Other important requirements, which we have seen in the last chapter, are speed and accuracy of placement. In this research, elaborated in Chapter 5, we have comprehensively addressed each of these requirements. The challenges have been addressed through multiple criteria optimization in an innovative placement strategy. Specifically, placements have been carried out to optimize cost and keep latency within the specified threshold. The placement is based on prediction of the state of the clouds at the time of placement. A number of innovations have been proposed in this part of the work. One such refinement, that makes predictions more accurate, is the compensation of the concept drift due to diurnal variation of traffic. The selection of clouds is through a generalized random selection algorithm. To the best of our knowledge, all of these techniques have been developed and used by us for the first time. We have also seen that the ILP solutions are slow in giving optimal solution. This limits their utility in responding fast to the change of state of the multi-cloud system and renders the method unusable in real-time applications. We avoid the ILP route and use machine learning for placement, which reduces the time taken even for large placements and renders the re-evaluation problem trivial. The method that we have developed falls in the category of dynamic and proactive placement algorithms rather than being either of those. Our objective and constraint-based determination of clouds, on which the SFC will be placed, removes the tight binding between resources and the VNFs of the SFC. During operation, the placement is frequently re-evaluated to ensure continued optimality. If required, new placement and virtual resource dimensioning will be done consistent with the carrier SLA requirements and CSP policies. The methods adopted also lead to the high efficiency of the placement process, which ensures that placement requests are successful in all cases where enough capacity is available and constraints can be met. The high speed of placements allows the CSP to make changes in the network dynamically, in real-time or near real-time, as the factors like demand, traffic congestion on links, availability of resources on various clouds change.

The HYPERVINES fault and performance management framework: This part of the work deals with the examination of the major reasons for performance and availability challenges in NFV and cloud-based VNS deployments. One of the major reasons is found to be the absence of a credible fault and management system. We find that handling detection and localization of fault and performance issues is difficult because of multiple layers in implementation of VNSs and the ill-defined interfaces among different management platforms for handling the distributed and overlapping responsibilities of fault and performance management. To address these challenges we have carried out the following work:

i) Develop an architectural framework for detection and localization of manifest and impending fault and performance problems.

ii) Develop mechanisms, within the described architectural framework, which make use of the network’s operational markers for detection and localization of faults and performance issues.

iii) The innovative use of shallow and deep predictive algorithms to obtain high accuracy of detection and localization. We achieve accuracies markedly better than the baselines and any other reported result in similar environment.

iv) Demonstration of the feasibility and effectiveness of the proposed framework using real network data

The merged hierarchical model with layer reuse for dataflow security: Subsystems of IoT and Multi-cloud based (also referred to as the next generation in conformity with a funded proposal in this area) healthcare would be connected in cyberspace, and therefore, prone to new vulnerabilities. Protection of patients' data, as it flows between domains and from cloud to cloud within the cloud hierarchy, against the effects of malicious intrusions, is an important part of the overall security strategy of the healthcare system.

To address the inter-cloud dataflow security challenges, our contributions are as follows:

i) Evolving system and security architecture for the next generation healthcare.

ii) Evolving a threat model for the system

iii) Innovative use of deep neural network, in the form of stacked autoencoders at the edge clouds and public cloud, for protecting dataflow in motion among the clouds.

iv) Developing a merged model for reducing the training time and improving accuracy of models in the public cloud.

v) Evaluating the merged model on public and lab generated datasets.

Chapter 4

Optimization of multi-cloud management and control platform

In Chapters 2 and 3 we discussed the challenges, state of the art and contributions of this thesis towards collection of behavioral data and optimization of multi-cloud management platforms. In this chapter we will discuss the work that we have carried out for optimizing management platforms, supplemented with the specific case of optimization of OpenADN multi-cloud management and control platform developed at Washington University in St. Louis.

4.1 Introduction and Motivation

Multi-cloud management and control platforms control the lifecycle of virtual resources obtained from multiple clouds. The ability to deploy and manage resources across heterogeneous clouds is important for both enterprise and carrier segments. Enterprises may obtain virtual resources from cloud service providers for their internal functions or to provide services to other clients. For instance, a cloud aggregator may obtain resources from multiple clouds and lease them to carriers for deployment of VNSs. Our focus on the problem is with a view to deploy VNFs over the virtual resources obtained from multiple cloud service providers. As the analysis made in this work is applicable equally to a company, like an airline, deploying IT applications, as well as to carrier/ISP obtaining virtual resources for their own services, we refer to both as ASPs.

The term software-defined infrastructures (SDI) would be used to refer to virtualized resources that the CSPs and the Network Service Providers (NSPs) offer through software-based control and management systems. The physical devices, on which these virtual infrastructures are created, could themselves be located in one or more datacenters of a cloud or diverse and geographically separated clouds each associated with one or more datacenters. Software control of infrastructure allows ASPs the flexibility of creating and managing *application specific virtual clouds* carved out of virtual resources from multiple clouds interconnected with virtual networking resources. After gaining experience of single cloud deployments, the enterprises are now turning to multiple public clouds [76], for added benefits of lower cost, increased flexibility, greater reliability by removing the possibility of single point of failure, proximity to users, reduced latency and a larger number of specialized features. A converged view of resources across multiple clouds allows them to use resources from many providers in a manner that enables optimization cost and quality of service parameters. SDIs provide ASPs with such a converged view of resources owned by different cloud service providers. In the case of carriers, this also brings the advantage of being close to the subscribers that they are serving and reducing the cost of the access network. On the flip-side, use of resources from multiple clouds brings in the complexity of not only interfacing with disparate clouds, but also the requirement of managing the network connecting the clouds.

Enterprises that choose to adopt a multi-cloud strategy are required to lease wide area network capacity to interconnect resources on different clouds. The Internet only gives best-effort performance and ASPs requiring performance guarantees must bear the cost of static preprovisioning of resources or of creating smart overlays. Having its own private network would prove to be quite expensive for small and medium enterprises. This means that most enterprises adopting a multi-cloud strategy would also need a shared network infrastructure that can satisfy their requirements. Virtualization is helpful as it creates customized network contexts, on a common physical infrastructure, for tenants' application specific requirements. OpenADN helps applications to automatically and dynamically communicate their requirements to the network.

Large ASPs, like Google, have the resources to install application layer proxies at distributed points of presence, so as to intercept service request and route it to the nearest datacenter. Through OpenADN smaller, network constrained ASPs can obtain such services from third party infrastructure providers, e.g. ISPs, who can route application messages through an appropriate set of controllers, proxies, and middleboxes. This way the ASPs can get the benefit of economically deploying distributed applications on multiple clouds to get increased responsiveness and resiliency [112].

Software, like OpenADN, presenting an integrated virtualized environment of physical resources of public clouds, operating under disparate control and management softwares, tends to be a complex system. Modularity is important in such systems for ease of development, maintenance and fate decoupling of the processes. They also generally use multithreading for concurrent execution of a number of activities. Any attempt to optimize such a platform would require use of behavioral testing to understand the behavior of the software, if possible in the production environment. Software engineers need to conceive innovative techniques to isolate problem areas that consume a disproportionate share of resources leading to sub-optimal behavior. Multithreading makes the system difficult to profile because characterizing the effects of interactions between threads becomes difficult as described in [52]. Efficient abstractions need to be developed to capture this behavior without resulting in exponential analysis times.

In the context shared above, we have carried out work on three fronts: collecting platform behavioral data, analyzing the data to identify factors significant for optimization and optimizing multi-cloud platform followed by extensive testing on OpenADN. To this end we elaborate how the operational behavior of the platform can be analyzed to obtain data and the two-factor full factorial analysis [113] can be used to make initial assessment of factors that lead to inefficient operation of the platform. Optimization of the platform, using the identified factors, brings forth interesting results that are discussed later in this chapter. This chapter is based on our work published in [7] [8].

4.2 Managing Software Defined Infrastructure Over Multiple Clouds – OpenADN

Most contemporary and future application deployments like Internet-of-Things (IOT) based next generation healthcare, Cyber-Physical Systems like autonomous mobile systems, mobile apps, massively parallel gaming and virtual reality tend to be distributed and need to use multiple clouds, primarily due to cost and latency considerations. OpenADN can be used by ASPs to manage such distributed applications over multiple clouds as if they were deployed on a single cloud. Alternatively, they may use services of a cloud aggregator, who would in turn use a multi-cloud management platform like OpenADN. In the following sections, we shall see why the architecture of OpenADN is suitable for such massively distributed application scenarios. It is also relevant to discuss these details as they affect the performance of the system.

OpenADN is interposed between the clouds and the interconnecting network on one side and the application deployment environment on the other. As shown in Figure 4.1, it has two types of external interfaces. The northbound interfaces are for the application developers, application architects, and application deployment administrators to define the application resource requirements and deployment policies. The southbound interfaces allow OpenADN to interact with the management and control systems of each cloud and network service providers to manage virtual resources obtained from them. The northbound interfaces of single cloud/network management platforms like OpenStack/OpenDaylight become southbound interfaces of OpenADN. OpenADN architecture has a modular structure similar to the OpenDaylight SDN controller [114].

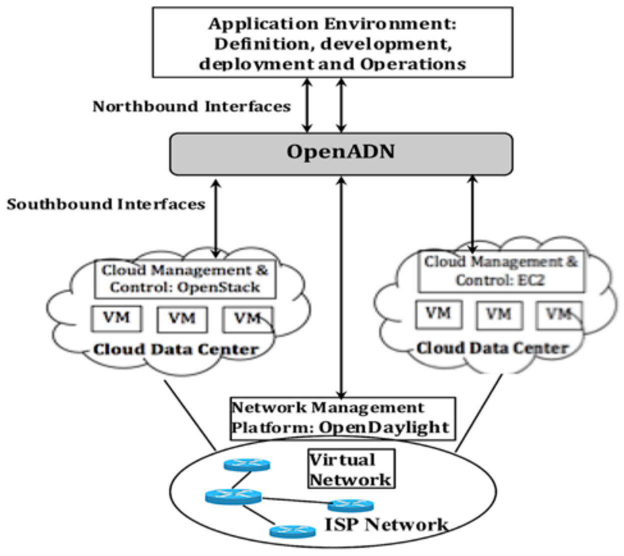

Application Environment: Definition, development, deployment and Operations
Northbound Interfaces
OpenADN
Southbound Interfaces
Cloud Management & Control: OpenStack
VM
VM
VM
Cloud Data Center
Cloud Management & Control: EC2
VM
VM
VM
Cloud Data Center
Network Management Platform: OpenDaylight
Virtual Network
ISP Network

Figure 4.1 OpenADN Multi-Cloud Management System

The ASP specifies the policies regarding when and where to create the resources and OpenADN interacts with the respective cloud manager to create those resources, rather than directly manipulating them. The management plane of OpenADN is centralized so that the policies are uniformly applied. The control plane has a hierarchical architecture with part centralized and part distributed controllers. This hybrid design of the control plane with a centralized global controller and distributed local controllers gives benefits of both the worlds. While distributed architectures are more scalable and also more resilient against failures and security threats, centralized architectures are simpler to manage. A proper division of work between the global and the local controllers ensures a good combination of latency and accuracy. The distributed nature of data plane takes advantage of the distributed applications and the network.

A brief description of the architecture of OpenADN and its initialization activities is provided in the following sub-section. These help in easier understanding of the optimization process of the platform.

4.2.1 Components of OpenADN and Bootstrapping

The key components of the OpenADN multi-cloud management platform are shown in Figure 4.2. The global controller is part of the hybrid control plane of the OpenADN that consists of a global controller and one or more local controllers. The hierarchical design of the control plane achieves scalability and resilience of the distributed architecture and ease of maintenance of the centralized one. This design helps in multi-datacenter environments where virtual resources are widely distributed. Each cloud will have its own local controller. The global manager is centralized and compiles the ASP policies received through the northbound interface. It maintains a database of resources available from contracted clouds, access policies and billing information. The primary function of the global manager is to bootstrap the application deployment based on the policies provided.

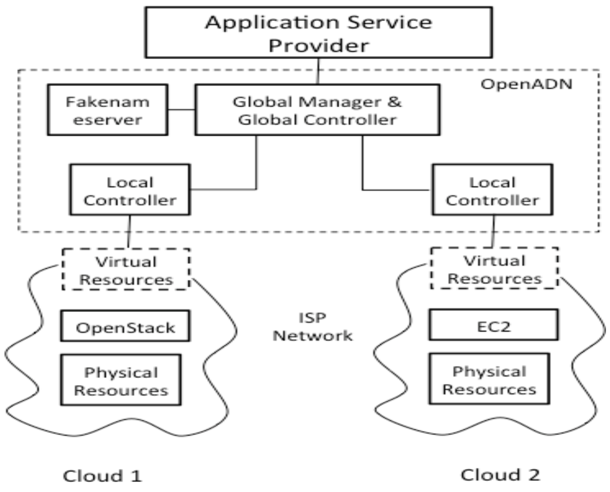


Figure 4.2 Key Components of OpenADN

The bootstrap process: The global manager is the only module that needs to be manually started. It chooses an initial set of clouds and acquires a pre-configured minimum number of virtual resources. It starts a global resource manager thread for each datacenter. The global resource manager adds resources from different sites to the global resource pool. It also allocates a specific role to each node and configures it accordingly.

The global manager then starts the global controller. The global controller can be on-site or in the cloud on virtual resources. The global manager also starts a local controller for each cloud. It then launches the OpenADN virtual machines and switches. After completing these steps, it tells the global controller to start the workflow manager. The global manager also creates event alarms for the ASP based on the runtime logs. During runtime it accepts request form the global controller for additional resources or for surrendering of resources not in use.

From here the global controller takes over. It launches one workflow manager (WFM) for each workflow (Figure 4.3). The WFM creates a workflow thread (WFT) for each zone/cloud where application is deployed. Each WFT can spawn multiple workflow instances. The workflow registers the application instance with the fakenameserver. The WFT also tries to gets a proxy node allocated to itself. A proxy port (pPort) is required to connect all external users and third party services, which are OpenADN unaware to the OpenADN gateway node. These services can be connected to internal OpenADN aware services, which are connected to the gateway through an sPort (message level services) or a tPort (packet level services). Since WFM does not yet have the resources it queues the request and sends request failed to the WFT. WFT runs an exponential backoff and repeats request checks for resources and launches workflow instances.

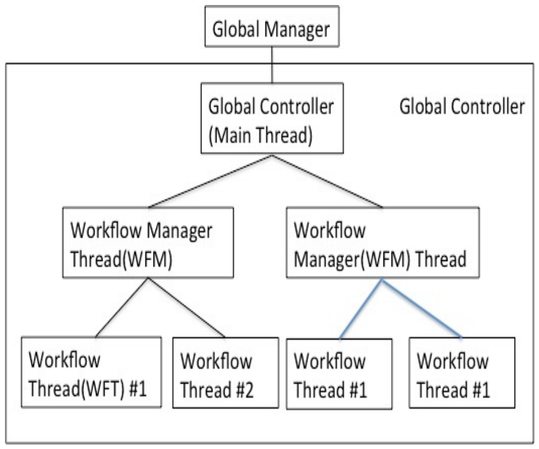


Figure 4.3 Functions of OpenADN relevant to the bootstrap process

WFT starts the services for the workflow after the WFM has allocated it the required resources. Initially it launches at least one workflow instance for the workflow. Each local controller (LC) boots up separately. The LC registers with the GC and starts connection dispatcher, which allows data plane nodes to register with the LC. The LC makes these resources available to the GC on request.

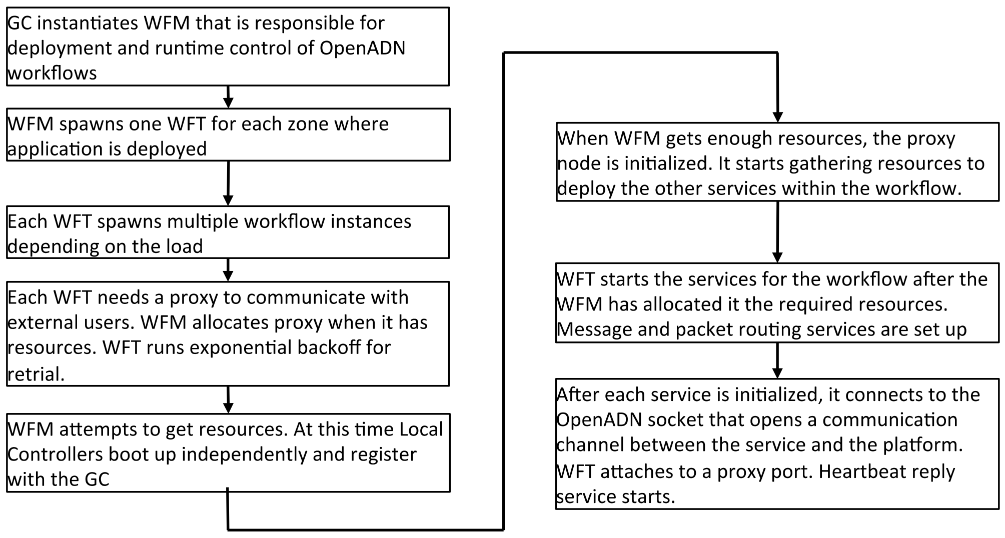

GC instantiates WFM that is responsible for deployment and runtime control of OpenADN workflows
WFM spawns one WFT for each zone where application is deployed
Each WFT spawns multiple workflow instances depending on the load
Each WFT needs a proxy to communicate with external users. WFM allocates proxy when it has resources. WFT runs exponential backoff for retrial.
WFM attempts to get resources. At this time Local Controllers boot up independently and register with the GC
When WFM gets enough resources, the proxy node is initialized. It starts gathering resources to deploy the other services within the workflow.
WFT starts the services for the workflow after the WFM has allocated it the required resources. Message and packet routing services are set up
After each service is initialized, it connects to the OpenADN socket that opens a communication channel between the service and the platform. WFT attaches to a proxy port. Heartbeat reply service starts.

Figure 4.4 The OpenADN bootstrap flow diagram

OpenADN is an integrated infrastructure comprising both, message-level devices (e.g., firewalls) and packet-level devices (e.g., intrusion detection devices), hosting application-layer services as well as network-layer services. For massively distributed applications, like mobile healthcare monitoring or mobile app delivery, OpenADN allows multiple zones with each zone consisting of multiple clouds [112].

Policies specified by the application administrator include specifying how to distribute the application delivery network deployment initially and during runtime. It is important to decide when and where to instantiate new instances and shutdown or move existing instances to support change in the application context. This massively distributed data plane structure makes the performance evaluation of OpenADN difficult and calls for specialized techniques that we shall discuss in the following sections.

4.2.2 Massively Distributed Nature of OpenADN

OpenADN optimizes application service deployment by deploying the hosts (nodes) of the distributed data plane, on virtual resources of various clouds. The system can perform many different tasks at the same time leading to better utilization of the hardware resources and ensuring that the system as a whole makes progress all the time. Applications like carrier's virtual network services or IoT based healthcare have scaling and latency requirements that can only be solved by having a massively distributed compute infrastructure. Another requirement is that the application should be able to change its topology dynamically based on usage patterns. OpenADN has been designed for these types of applications. Consequently it is has a massively distributed data plane structure, with several threads in the state of operational stupor, making the performance evaluation of OpenADN difficult. This platform calls for specialized techniques that we shall discuss in the following sections.

4.2.3 Design and Coding Considerations

Most of the control plane code has been implemented using Python while the data plane has been implemented with a mix of C and Python. The total size of the code base currently consists of about 10,000 lines of code. OpenADN has been designed as modular software to provide code readability and maintainability. Modularity also restricts inter-module interference in case of failures. Multiple operations are performed simultaneously to support multiple clouds, multiple users and multiple services. Partitioning of OpenADN into concurrently executable modules leads to better utilization of the hardware resources [115] [116]. The platform modules and application services are executed in separate processes. Application services are designed as external modules that connect to the platform through an external communication interface. A failed application service(s) can be handled by the platform without affecting other services. The services may run on the same or separate hosts. On the same host they use inter-process communication (IPC) while on different hosts they communicate using the network transport layer.

In each host, platform modules may run within the same address space (same process) but in separate threads to achieve concurrency. The ports handling packet level services and message level services run in separate processes because kernel network stack has been used for packet level communication. These threads share the process heap, which provides them a way to communicate with each other. However, in order to avoid fate sharing completely, threads communicate inside the platform process through messaging. Depending on the profiling techniques used, these design considerations could affect the outcome of profiling.

4.3 Performance Modeling of OpenADN

This section defines the experimental modeling that assists in gathering behavioral data, while the platform is in operation, and carries out performance evaluation decision for optimization. While we have considered the OpenADN as a representative platform, the techniques described here are applicable to any other cloud platform.

The main goal of the study is to first methodically and scientifically locate areas of code in the platform that might cause it to consume unduly large amount of computing resources during bootstrap and normal operation. Then we apply an experimental design technique to find whether any of the located hotspots have significant impact on the metric described in sub-section 4.3.2. The actual setup of the experiments described in detail in Subsection 4.5 would be used as the basis for carrying out the collection of profiling data using techniques mentioned therein.

4.3.1 Services of OpenADN

OpenADN offers all the basic services expected of such a platform, e.g., allocation of resources from multiple clouds, distribution of applications, scaling/de-scaling, and performance assurance of workflows. Its uniqueness, however, lies in the additional application and network layer services it offers for highly distributed and multi-threaded applications to run on multiple clouds [117] [1]. These services include:

1) Application layer services including message and packet level services (called middleboxes)

   a) Message level services: webservers, database servers, and web firewalls.

   b) Packet level services: Intrusion detection and intrusion prevention systems.

2) Network layer services like packet forwarding and routing

The expected outcome is the effective use of resources, assurance of meeting quality of service and dynamically ensuring efficient operation of the system. However, if the system operates sub-optimally, say under a computationally demanding application, it results in higher cost, exactly opposite of what it was supposed to achieve. Performance parameters like latency may be met for some applications and may not be met for others at all times. Communication among message level or packet level devices may take unduly long time. These issues were kept in mind for deciding metrics and parameters as discussed in the next sub-section.

4.3.2 Metrics, Factors and Parameters

The main metric is the CPU time taken to execute the platform software during the complete process of bootstrap and as the services start. Execution times for individual functions that consume a large amount of time would be of interest. The system parameters include: the type of virtual machines setup, storage capacities, intra-cloud and inter-cloud network bandwidth. The workload parameters, which presumably affect the metric, are the users' requests for services, types of services – message or packet service and the amount of resources available.

4.4 Profiling techniques for Gathering Behavioral Data

In this section, we discuss the importance of profiling led optimization followed by a selection of techniques that are suitable for profiling OpenADN and gathering data for optimization.

Often application software has code that consumes a disproportionate amount of resources and produces high CPU loads. Cloud management platforms are no different. The whole idea of profiling multi-cloud delivery platforms is to work through the tiers and threads of these platforms and collect information about their behavior in different operational situations. To this end, it is important to use program analysis tools that are appropriate to the distributed, multi-threaded nature of these platforms and gather as much information as possible. As against this, if we choose to carry out intuitive optimization, it may result in modification of parts of the code that were not responsible for performance degradation and as a result may be a waste of time. A word of caution: too much optimization or too little of it are both considered detrimental. Donald Knuth stated in [118] that programmers waste an enormous amount of time thinking about the speed of non-critical part of the program. About 97% of the time we should forget about small inefficiencies as premature optimization is the root of all evil. It is not only important for profiling to precede optimization, it is also important to use correct techniques that would produce reliable data based on which it can be decided whether optimization should be carried out, and if the answer is in the affirmative, what parts of the code should be optimized [15] .

Given the nature of OpenADN, most conventional profiling, characterization, and modeling methodologies do not work well because of full system virtualization. They do not provide definitive help in pinpointing the sections of code that should be optimized. We shall see here a combination of techniques that can be applied to a distributed, multi-threaded system [52].

The basic techniques were discussed in Section 3.1.1 under the categories of static and dynamic testing. We saw that static testing profiles the software through manual or automated walkthroughs to find problems with the code. Model checking as a tool for static analysis becomes computationally expensive when multi-threading is extensively used. We also saw that these techniques give relative time assessments of events like function calls or time spent in a function [58]. We, therefore, conclude that this technique would not be applicable to the OpenADN environment.

Dynamic analysis could be statistical where state of the program is sampled to make a relative assessment of timing of events or deterministic where events can be precisely timed by using instrumented code. Dynamic analysis allows collection of performance data from a system in execution. It also provides absolute time of events. Dynamic profiling could be done through code instrumentation. The injected code provides information about execution of the programs with varying degree of granularity. However, care must be taken that the injected code does not cause any change in the program behavior, potentially causing inaccurate results. While the event timing with deterministic profiling takes into account interaction of threads, using concurrent analysis, a more precise thread level examination can be made (Microsoft, 2013 and Oracle, 2012)

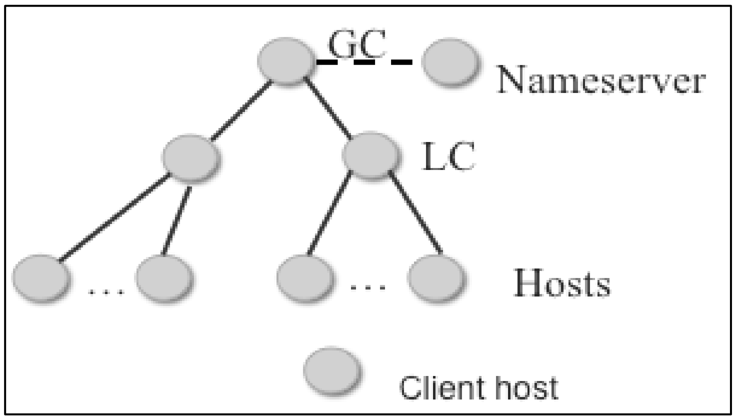
4.5 Evaluation Techniques And Experiment Design

OpenADN is modular, multi-threaded and complex software, making it difficult to analyze using traditional analysis. However, if the platform operates sub-optimally, it may result in leasing more resources than are necessary, higher deployment cost and unusual communication delays. The existing prototype of OpenADN provides the test and evaluation environment. The platform software was loaded and executed in the virtual environment. Experiments were conducted to observe effect of different workloads (involving varying number of clients and hosts) and also various functions of platform on CPU time required.

The experiment was designed as a two factor full factorial design without replications [113]. The reason for choosing this design actually became obvious while conducting profiling studies and collecting data. We had a situation where two sets of parameters, i.e., functions (host creation, polling and sleep) and workloads (users and the number of hosts) were affecting the CPU time. A careful control of these two sets of parameters was required. We assumed that the factors are categorical. A full factorial design with two factors functions (A_j) and workloads (B_i) having i= j=3, i.e., 3 levels each. The results are deterministic in nature and, therefore, single replication of each experiment was considered sufficient.

4.5.1 Dynamic Layered Analysis for Identification of Factors for Optimization

To validate the functionality, we ran OpenADN in a virtual environment created by Mininet [119]. Mininet allows emulating a whole virtual network running real kernel, switch and application code, on shared physical resources of a machine. The following virtual resources were created for profiling OpenADN: One service zone consisting of a global controller, a fakenameserver, two data center sites with a local controller each, a name-server, seven hosts per site and client host simulating 10,000 users. The selection of stimuli (set-up and input data) and multiple runs of the platform ensured that behavioral data for most control paths were collected. The experimental OpenADN setup is shown in Figure 4.5. The host OS is Mac OSX while the guest OS is Ubuntu 12.04 LT on all the VMs.



Legends: GC – Global Controller LC: Local Controller

Figure 4.5 The experimental set-up

Multi-layered dynamic and deterministic techniques were used to collect behavioral data and find potential factors for optimization. Deterministic method of profiling monitors functions or even statements and collects precise timing of execution as well as call count. In interpreted Python the presence of instrumented code is not required to do deterministic profiling. We did not draw conclusions directly from the profiling data. These were subjected to further analysis before coming to any conclusions. Our method consists of using a combination of dynamic analysis and full factor analysis to locate and confirm factors that have significant impact on performance and should be taken up for optimization to achieve improved performance of the platform. These methods were applied to OpenADN and results obtained were published in [7].

The layered profiling model shown in Figure 4.6 helped us to progressively get more detailed information and zero in on the problem areas. Platform level profiling provided overall execution data for the complete platform. However, as we shall see, while it gives useful information to start with, it does not pinpoint the problems in the code. Function level profiling enables us to measure the CPU times for execution of various functions so that we could isolate the blocks those took disproportionate time to execute. It does not, however, tell us the exact location of these time consuming operations. Some functions are called repeatedly in different modules. Thus, statement level profiling was carried out to get the location of the calls that were resulting in inappropriate behavior. Deterministic and concurrent techniques were used to be able to measure absolute timing of events for carrying out the experimental study.

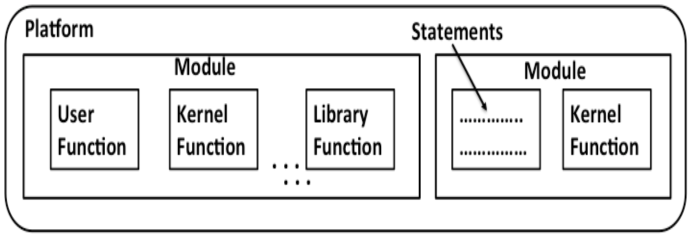


Figure 4.6 The Layered Profiling Model

Platform level analysis: To get a broad idea of the efficiency of the platform code executing in a virtualized environment, the built in timing utility of the operating system was used. Table 4.1 shows the CPU times (in seconds) for user space functions, kernel (system) functions, total of user and kernel and the overall run time of the platform across seven runs. At the platform level, across many runs, of the average total elapsed time of 49.269 seconds for which the platform software was executed, the time spent in user functions and kernel space was 1.3% and 1.77%, respectively. The information is graphically depicted in Figure 4.7. This gives a sense that a large part of the time is spent in activities such as waits and sleep times for dealing with dependent asynchronous concurrent processes. However, it cannot be yet said whether this time relates to unavoidable delays and the situation can be improved through optimization. This called for the next level of profiling, i.e., at module/function level to see which of the modules are more CPU intensive.

Table 4.1 Platform level execution data

Run	User Space	System Calls	User+System	Run Time
1	0.872	1.248	2.12	43.068
2	0.948	1.796	2.744	54.675
3	0.864	1.064	1.928	40.464
4	0.94	1.256	2.196	46.65
5	1.016	1.04	2.056	35.936
6	2.5	3.096	5.596	64.05
7	2.004	2.91	4.914	60.041
Averages	1.306	1.773	3.079	49.269
% of Run Time	2.651	3.598	6.250	

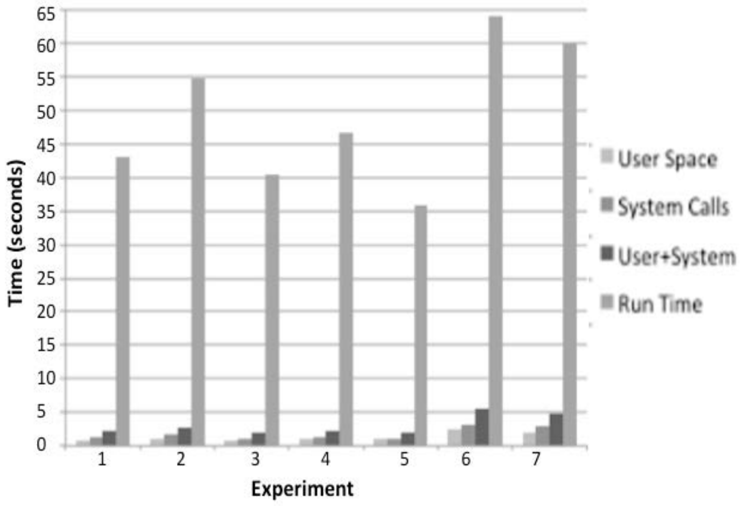


Figure 4.7 User, System and overall CPU time for OpenADN

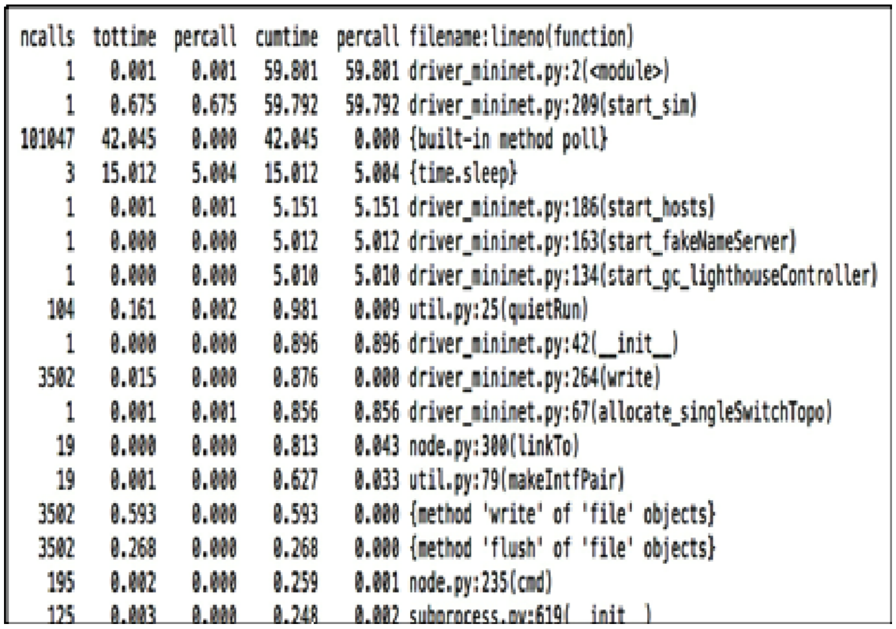
The same modules were also run on separate physical machines for comparison and the results obtained are given in Table 4.2. On physical machines, the platform does not have to spend time creating virtual machines for its own modules as well as for running services. Even in this case the overall user-space time is 17.49% and even less for kernel calls. Among these, the global controller used the time more effectively with user functions taking up to 41.64% of run time on an average. However, in the actual operational environment, these modules will be hosted on VMs that will take a finite amount of time to create, start, augment or migrate to another cloud (unless otherwise indicated all times are in seconds).

Table 4.2 Time used for user and system activities on physical machines

Function	**User Space**	**System Calls**	**Run Time**	**User (%)**
Name Server	14.161	5.072	229.438	6.17
Global Controller	83.637	15.797	200.835	41.64
Local Controller	18.549	7.16	175.57	10.57
Node Controller	19.95	8.86	156.99	12.71
Client	0.428	0.036	18.855	2.27
Total	136.725	36.925	781.688	17.49

This simple profiling indicates the possibility of higher load on the CPU, because of potentially wasteful activities like waiting on I/O calls and the sleep functions. While in many cases, where asynchronous linking of threads is used, some waiting would be unavoidable. However, one needs to see whether these could be optimized for 1) making the platform more efficient 2) correctly dimensioning the resources leased, and 3) distributing the workload properly.

Function Level Profiling and Analysis: Python library provides routines to collect behavioral data at the function level. These routines provide a set of statistics that describes how many times different functions are called and how much time the CPU is spending to execute various modules. The statistical data collected needs to be processed through some other conversion routintes like ‘pstats’ to make them amenable to analysis. A large volume of data was produced of which a part of output is shown in Figure 4.8. From line 1 and 3 of the figure it can be seen that the platform was executed for a total of 59.801 seconds out of this the polling function took 42.045 seconds. At the function level polling and sleep took 70.31% and 25.1% time respectively. OpenADN uses the ZeroMQ™ polling function that provides communication between modules on different virtual hosts.



```
ncalls  tottime  percall  cumtime  percall filename:lineno(function)
     1    0.001    0.001   59.801   59.801 driver_mininet.py:2(<module>)
     1    0.675    0.675   59.792   59.792 driver_mininet.py:209(start_sim)
101047   42.045    0.000   42.045    0.000 {built-in method poll}
     3   15.012    5.004   15.012    5.004 {time.sleep}
     1    0.001    0.001    5.151    5.151 driver_mininet.py:186(start_hosts)
     1    0.000    0.000    5.012    5.012 driver_mininet.py:163(start_fakeNameServer)
     1    0.000    0.000    5.010    5.010 driver_mininet.py:134(start_gc_lighthouseController)
   104    0.161    0.002    0.981    0.009 util.py:25(quietRun)
     1    0.000    0.000    0.896    0.896 driver_mininet.py:42(__init__)
  3502    0.015    0.000    0.876    0.000 driver_mininet.py:264(write)
     1    0.001    0.001    0.856    0.856 driver_mininet.py:67(allocate_singleSwitchTopo)
    19    0.000    0.000    0.813    0.043 node.py:300(linkTo)
    19    0.001    0.000    0.627    0.033 util.py:79(makeIntfPair)
  3502    0.593    0.000    0.593    0.000 {method 'write' of 'file' objects}
  3502    0.268    0.000    0.268    0.000 {method 'flush' of 'file' objects}
   195    0.002    0.000    0.259    0.001 node.py:235(cmd)
   125    0.003    0.000    0.248    0.002 subprocess.py:619(__init__)
```

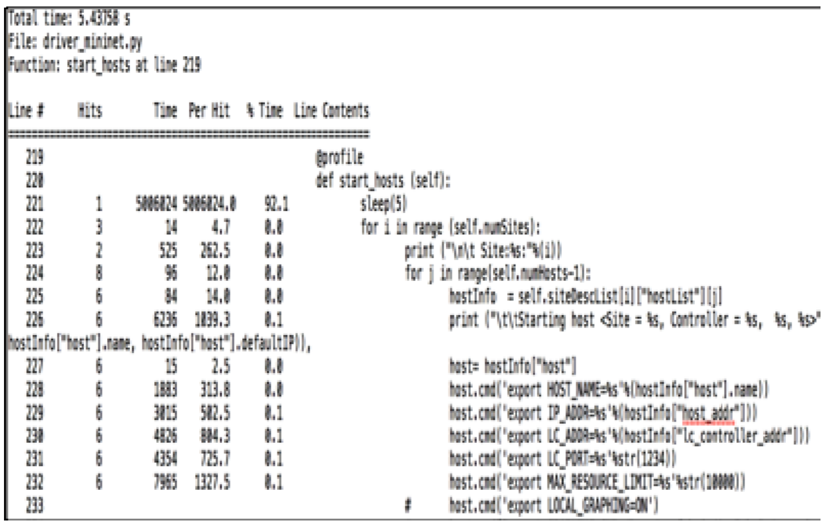
Legend: **ncalls:** the total number of calls, **tottime:** total time spent in the given function (excluding sub functions) seconds, **percall**: tottime divided by ncalls, **cumtime**: total time in this and all sub-functions seconds, **Percall**: cumtime divided by primitive calls, **filename**: data for each function

Figure. 4.8 Extract of Function Level Profile

The communicating services have to poll the sockets to check for new incoming messages. Polling is, however, called many times in different functions. If a large amount of time is taken then this may be an indication that the entire process of platform execution may be slowing down. To know the exact location of this time consuming operation and other such operations statement level profiling was done.

Statement Level Profiling: As is often the case, the reason for a particular module or functionality taking a large amount of time could be pin-pointed to some small part which may seem to be innocuous on simple reading of the code. Some statements could trigger a library function or call a special method that may not be so obvious. A more detailed line-by-line analysis of the program was undertaken to find out which parts of the program take more CPU time. The line-profiler described in pypi.org [120] used in a judicious manner allows this kind of analysis. This profiler keeps track of multiple statement executions, sums up the total time each statement takes in multiple passes and avoids profiling overheads. The profiling result is a binary file that could be deciphered with ‘pstats’ or a similar function.

Workload was varied to get CPU times for various statements and identify the functions that should be taken up for further analysis. Figure 4.9 shows a section of the profiling output with a large proportion of sleep time (92%) and also the time taken for creation of hosts.

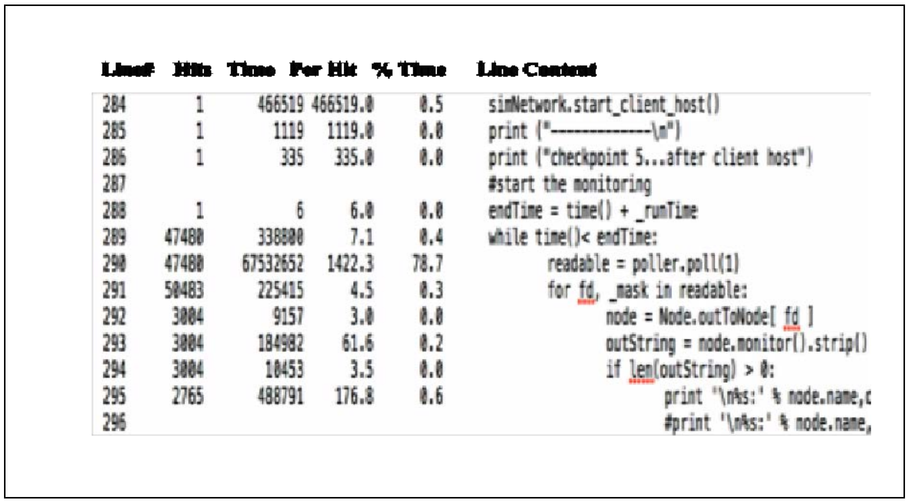


```
Total time: 5.43758 s
File: driver_mininet.py
Function: start_hosts at line 219

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
   219                                           @profile
   220                                           def start_hosts (self):
   221         1      5006024 5006024.0     92.1         sleep(5)
   222         3           14      4.7      0.0         for i in range (self.numSites):
   223         2          525    262.5      0.0                 print ("\n\t Site:%s:"%(i))
   224         8           96     12.0      0.0                 for j in range(self.numHosts-1):
   225         6           84     14.0      0.0                         hostInfo  = self.siteDescList[i]["hostList"][j]
   226         6         6236   1039.3      0.1                         print ("\t\tStarting host <Site = %s, Controller = %s,  %s, %s>"
hostInfo["host"].name, hostInfo["host"].defaultIP)),
   227         6           15      2.5      0.0                         host= hostInfo["host"]
   228         6         1883    313.8      0.0                         host.cmd('export HOST_NAME=%s'%(hostInfo["host"].name))
   229         6         3015    502.5      0.1                         host.cmd('export IP_ADDR=%s'%(hostInfo["host_addr"]))
   230         6         4826    804.3      0.1                         host.cmd('export LC_ADDR=%s'%(hostInfo["lc_controller_addr"]))
   231         6         4354    725.7      0.1                         host.cmd('export LC_PORT=%s'%str(1234))
   232         6         7965   1327.5      0.1                         host.cmd('export MAX_RESOURCE_LIMIT=%s'%str(10000))
   233                                                  #       host.cmd('export LOCAL_GRAPHING=ON')
```

Legend: Hits: Number of times that line was executed, Time: Total execution time Per Hit: Average amount of execution time, % Time: Percentage of time spent on that line relative to the total amount of recorded time spent in the function, Line Contents: Actual source code.

Figure 4.9 Extract of Statement Level Profile showing large sleep time 2000 users and 4 hosts

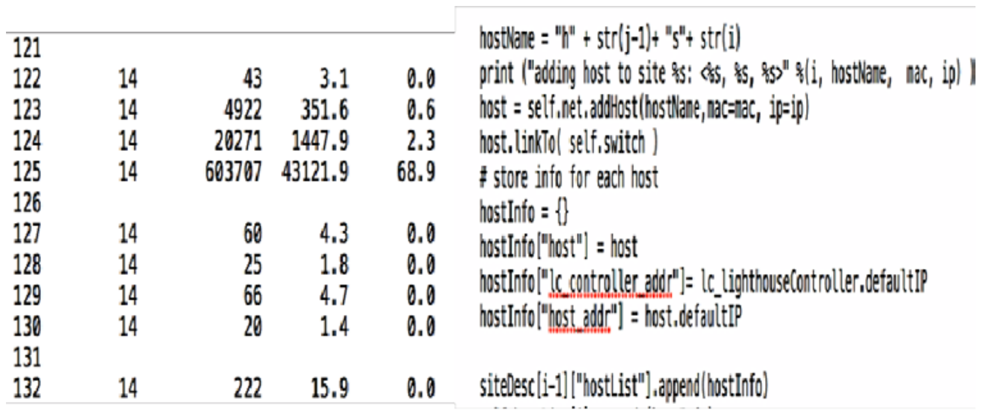
Figure 4.10 shows that, in this section of the profile, the polling function takes 78.7% of the time. Legends of Figure 4.9 are applicable.



```
Line#   Hits   Time     Per Hit   % Time   Line Content
284        1    466519 466519.0     0.5    simNetwork.start_client_host()
285        1      1119   1119.0     0.0    print ("------------\n")
286        1       335    335.0     0.0    print ("checkpoint 5...after client host")
287                                        #start the monitoring
288        1         6      6.0     0.0    endTime = time() + _runTime
289    47480    338800      7.1     0.4    while time()< endTime:
290    47480  67532652   1422.3    78.7            readable = poller.poll(1)
291    50483    225415      4.5     0.3            for fd, _mask in readable:
292     3004      9157      3.0     0.0                    node = Node.outToNode[ fd ]
293     3004    184982     61.6     0.2                    outString = node.monitor().strip()
294     3004     10453      3.5     0.0                    if len(outString) > 0:
295     2765    488791    176.8     0.6                            print '\n%s:' % node.name,c
296                                                                #print '\n%s:' % node.name,
```

Figure 4.10 Extract of Statement Level Profile showing large time taken by Poller 2000 users and 4 hosts

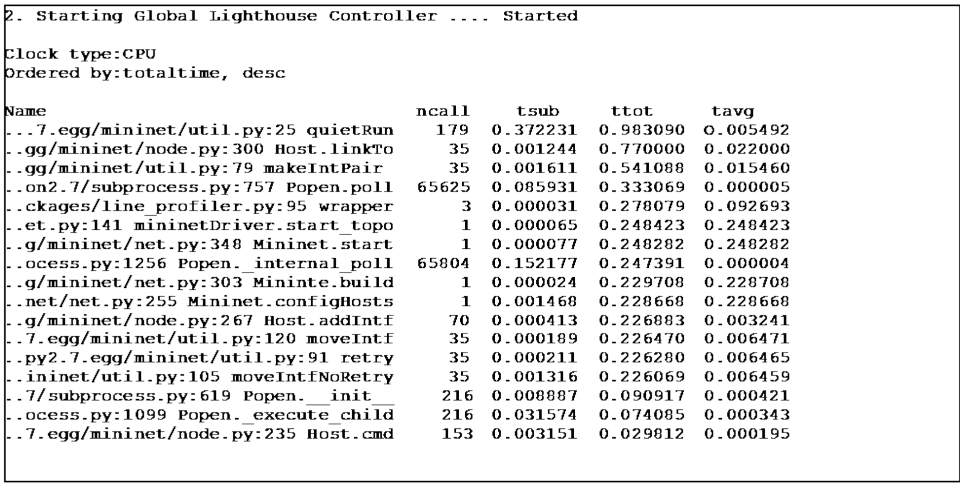
Figure 4.11 shows 68.9% of the CPU time taken by host creation and linking. Legends of Figure 4.7 are applicable.



```
121                                  hostName = "h" + str(j-1)+ "s"+ str(i)
122   14       43      3.1    0.0    print ("adding host to site %s: <%s, %s, %s>" %(i, hostName,  mac, ip) )
123   14     4922    351.6    0.6    host = self.net.addHost(hostName,mac=mac, ip=ip)
124   14    20271   1447.9    2.3    host.linkTo( self.switch )
125   14   603707  43121.9   68.9    # store info for each host
126                                  hostInfo = {}
127   14       60      4.3    0.0    hostInfo["host"] = host
128   14       25      1.8    0.0    hostInfo["lc_controller_addr"]= lc_lighthouseController.defaultIP
129   14       66      4.7    0.0    hostInfo["host_addr"] = host.defaultIP
130   14       20      1.4    0.0
131
132   14      222     15.9    0.0    siteDesc[i-1]["hostList"].append(hostInfo)
```

Figure 4.11 Extract of profile for 2000 users and 8 hosts showing 68.9% host creation and linking time

Concurrency Profiling Data: While the recursive function level profiling, that includes timing of execution of sub-functions and statement level profiling, reflects the effect of execution of various threads, individual thread behavior may not be evident. To get a better understanding of the multi-threaded platform, thread level profiling was carried out while the program was in execution. A sample of concurrency profile is given in Figure 4.12.



```
2. Starting Global Lighthouse Controller .... Started

Clock type:CPU
Ordered by:totaltime, desc

Name                                       ncall    tsub      ttot      tavg
...7.egg/mininet/util.py:25 quietRun         179  0.372231  0.983090  0.005492
..gg/mininet/node.py:300 Host.linkTo          35  0.001244  0.770000  0.022000
..gg/mininet/util.py:79 makeIntPair           35  0.001611  0.541088  0.015460
..on2.7/subprocess.py:757 Popen.poll       65625  0.085931  0.333069  0.000005
..ckages/line_profiler.py:95 wrapper           3  0.000031  0.278079  0.092693
..et.py:141 mininetDriver.start_topo           1  0.000065  0.248423  0.248423
..g/mininet/net.py:348 Mininet.start           1  0.000077  0.248282  0.248282
..ocess.py:1256 Popen._internal_poll       65804  0.152177  0.247391  0.000004
..g/mininet/net.py:303 Mininte.build           1  0.000024  0.229708  0.228708
..net/net.py:255 Mininet.configHosts           1  0.001468  0.228668  0.228668
..g/mininet/node.py:267 Host.addIntf          70  0.000413  0.226883  0.003241
..7.egg/mininet/util.py:120 moveIntf          35  0.000189  0.226470  0.006471
..py2.7.egg/mininet/util.py:91 retry          35  0.000211  0.226280  0.006465
..ininet/util.py:105 moveIntfNoRetry          35  0.001316  0.226069  0.006459
..7/subprocess.py:619 Popen.__init__         216  0.008887  0.090917  0.000421
..ocess.py:1099 Popen._execute_child         216  0.031574  0.074085  0.000343
..7.egg/mininet/node.py:235 Host.cmd         153  0.003151  0.029812  0.000195
```

name: function name, ncall: callcount of the function, tsub: time spent in the function, ttot:time spend in the function and sub-functions, tavg:ttot/ncall

Figure 4.12 Concurrency profiling with 2000 users and 16 hosts

The internal polling operation at line 8 of the output in Figure 4.12 shows that this function was called 65804 times after the global controller was started (even before the local controllers were activated) and a total of about 0.25 sec were spent in this operation. This amounts to about 26% of the time the thread spends in this function and the subfunctions it calls.

Finer granularity execution at statement level showed polling, sleep and host creation functions as the most expensive. Armed with this information we launch into the confirmatory phase.

4.5.2 Experimental Results and Analysis

From all the profiling runs with different workloads it is observed that three types of activities are consuming a large amount of time during the execution of the platform software:

1) Creation and linking of the host to the network
2) Polling of sockets for inter-service communication
3) Sleep function

This part of the work consists of conducting a two factor full factorial analysis on the factors revealed by the multi-layered dynamic analysis. The potential optimization opportunities revealed in sets of factors, functions (A): polling, sleep and host creation and workloads (B): the number of clients and hosts. The full factorial analysis confirms the significance of these factors by analyzing whether their effects on the execution times are real or just random. In our evaluation the workload was varied, by changing the number of users, from 500 to 2000, each accessing from a list of web pages, and also by varying the number of hosts per cloud from 4 to 16 for hosting the platform modules as well as the application. It is seen that execution time varies more with the number of hosts than with users. Without any loss of generality, we fix the number of users to 2000, and vary the hosts from 4 to 16 in steps of 4. Table 4.3 shows the execution time as a ratio of total module time to make them comparable across runs.

Effect of selected factors: Table 4.3 reveals that row effect (β_i) i.e. of the workloads are within 1% of the average effects of all factors (0.7001). The column effects (α_j) are, respectively, host creation on average 39% less than that of all factors and that of polling and sleep functions are 7.66% and 31.35% more.

Table 4.3 CPU Time for functions and workloads

Work-loads	**Functions**			Row Mean	Row effects (β_i)
	Host creation	**Polling**	**Sleep**		
2000/4 hosts	0.3307	0.7865	0.9641	0.6938	-0.0063
2000/8 hosts	0.4316	0.7571	0.9325	0.7071	0.0070
2000/16 hosts	0.5186	0.7174	0.862	0.6993	-0.0007
Column Mean	0.4270	0.7537	0.9195	**0.7001**	
Column Effects (α_j)	-0.2731	0.0536	0.2195		

Explanation of variations: The total variation of CPU time (y) can be attributed to ‘functions’, ‘workloads’ and the ‘experimental’ errors. The second column of Table 4.4 shows sum of squares explained by these factors (the individual means of Table 4.3). The percentage variation explained by functions (α_j) and workloads (β_i) is 68.29% and 0.05%, respectively. The unexplained variation due to errors (e_i) is 31.68%. This clearly shows that functions selected are important for optimization regardless of the workloads.

Table 4.4 Analysis of variance for Functions and Workloads

Component	Sum of Squares	% Variation	Degrees of Freedom	Mean square	F computed	F Table
y	4.8131		9			
μ	4.4107		1			
y-μ	0.4024	100	8			
α_j	0.2748	68.29	2	0.1374	8.6212	$F_{0.90,2,4}$= 4.32
β_i	0.0002	0.05	2	0.0001	0.0031	$F_{0.90,2,4}$= 4.32
e_i	0.1275	31.68	4	0.0159		

Analysis of Variance And Visual Results: We test the significance of the two factors as far as the execution time is concerned. From Table 4.3 we observe that the workloads had comparable runtimes. This ensured that effect of the functions selected is not overshadowed by the differences in the workload run times. The number of functions are a=3 and the number of workloads are b=3. In Table 4.4, the mean square is obtained by dividing the sum of square by the degrees of freedom. The ratio of mean square of functions and workloads and that of errors gives the computed F-statistic. The calculated F-statistic for functions is greater than that obtained from the F-table at 90% confidence level so they are significant for our study. The F-statistic for workload is less so they are not significant.

Visual examination of residuals and responses can be seen from the graphs below (Figure 4.13 (a) and (b)). In order to check the homogeneity of the error variance we obtain errors (Table 4.5) and plot them against predicted response.

Table 4.5 The estimated y and the residuals

$\hat{y}_{ij}=\mu+\alpha_j+\beta_i$			$e_i=y_{ij}-\hat{y}_{ij}$		
0.4207	0.7474	0.9132	-0.0900	0.0391	0.0509
0.4340	0.7607	0.9265	-0.0024	-0.0036	0.0060
0.4263	0.7530	0.9188	0.0923	-0.0356	-0.0568

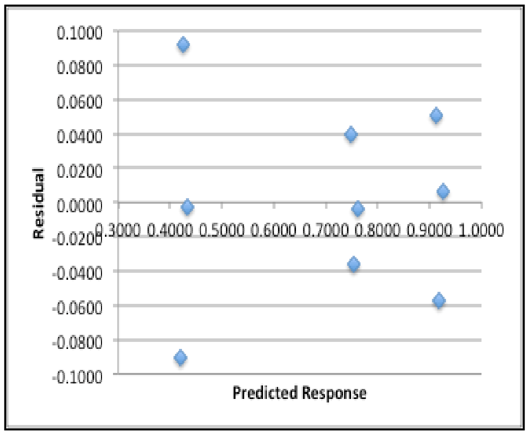


Figure 4.13 (a) Residual vs. Predicted Response

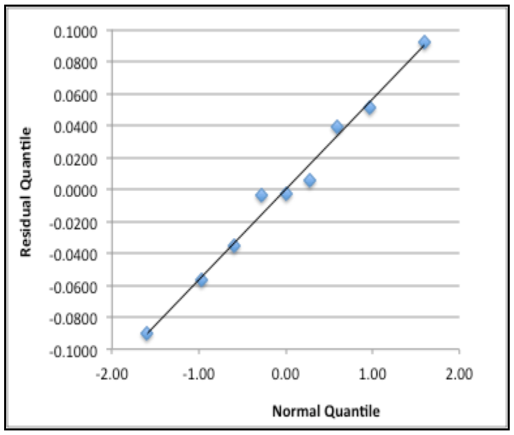


Figure 4.13 (b) Quantile-Quantile Plot

Confidence Intervals for the effects: To check the sanity of our results we took the analysis further by calculating standard deviations (SDs) and 90% confidence intervals (CIs) for the effects related to functions and workloads (Table 4.6). The 't' value used for the calculation of CIs is for 90% confidence interval and 4 degrees of freedom (the degrees of freedom for the errors).

Table 4.6 Calculation of CI of Effects

MSE =0.0159	s_e= 0.1261
SD of grand mean	s_μ=0.0420
SD of α_j	$s_{\alpha j}$ 0.0594
SD of β_is	$s_{\beta i}$ 0.0594
90% confidence interval for α	90% confidence interval for β
(0.3003, 0.5537)	(0.5670, 0.8205)
(0.6269, 0.8804)	(0.5803, 0.8338)
(0.7928, 1.0463)	(0.5726, 0.8261)

It can be seen that CIs of the functions are all significant. Also the means of α do not lie in the CI of each other so they are significantly different from each other. This implies that optimization with respect to these three functions should result in improvement in performance of the platform. The workloads on the other hand have their means in the CI of the others so they are not significantly different from each other.

4.6 Optimization Results

We first discuss the design choices that have been made in OpenADN and then the results of changes made on the platform software based on the profiling study and two factorial analysis.

4.6.1 OpenADN design choices that affect performance

OpenADN is envisaged to be deployed in a multi-cloud environment, instantiating multiple hosts in possibly geographically distributed datacenters. To make it efficient, it has been designed with asynchronous bootstrap mechanism. If a node attempts to connect to another and fails, it retries after a certain interval. This interval could be fixed or adaptive. In OpenADN it has been fixed. Another design choice of significance is the asynchronous request-response messaging in which sender send a message with an ID, the receiver maintains a message queue. When the response comes the sender matches the ID of the response. This allows the system to handle many requests simultaneously. The design of OpenADN is meant to be dynamic in adapting to application needs and scalable to handle large applications. To keep updated view of the whole system the control plane has to poll each node or the nodes must report their state at frequent intervals. Lazy updates involve polling before critical decision or data plane nodes report an event when load crosses a particular pre-set threshold.

Each data plane node (that is a virtual machine hosting a message-level or packet-level service) is part of an OpenADN Distributed Virtual Switch (DVS) implementation. Also, the OpenADN data plane implementation exposes two types of interfaces for services to connect to the DVS, the socket interface for OpenADN aware services and the proxy port (pPort) for legacy, non-OpenADN-aware services. OpenADN service flow will have message router, packet router, service node and proxies. OpenADN aware services connect through the sPort or the tPort and OpenADN unaware services through the pPort. The cPort is the realization of the control element in each of the virtual machines. Various ports associated with OpenADN are described below:

sPort: The sPort or the service port connects to a message-level service such as an web server, a storage server or a message level middlebox (e.g., Firewall, transcoder, etc.). The sPort is configured through the control port, which connects it to the local control agent that is in turn connected to a control plane controller.

pPort: The pPort or proxy port connects message-level legacy services to the OpenADN platform through the OpenADN Gateway Node. Users or third-party services working on a legacy application-level protocol such as HTTP or JSON connect to the proxy server. The proxy server assigns a session manager thread to each connection. The session manager thread is responsible for handling all the transactions within the user session. Unlike the sPort, where each sPort hosts only one service, the pPort maybe shared among many different application workflow instances.

tPort: The tPort or tunnel port allows attaching packet-level services to the application workflows. Unlike the sPort, the tPort is a shared port that can be shared with more than one packet-level service and these services may belong to different workflow instances.

cPort: Each virtual machine launched by the platform runs a host controller agent called the cPort. The cPort manages the OpenADN platform ports in the hosts. The cPort is the control plane agent in each virtual machine and is responsible for executing control plane instructions. These instructions may include commands for launching a new service and programming the corresponding platform port through which the service is attached to the platform; or replying to queries from the control plane regarding service liveness, resource availability and load information.

4.6.2 The Bootstrap Process Revisited

The bootstrap process was discussed in the sub-section 4.2.1. We see it very briefly here to facilitate the discussion of the results.

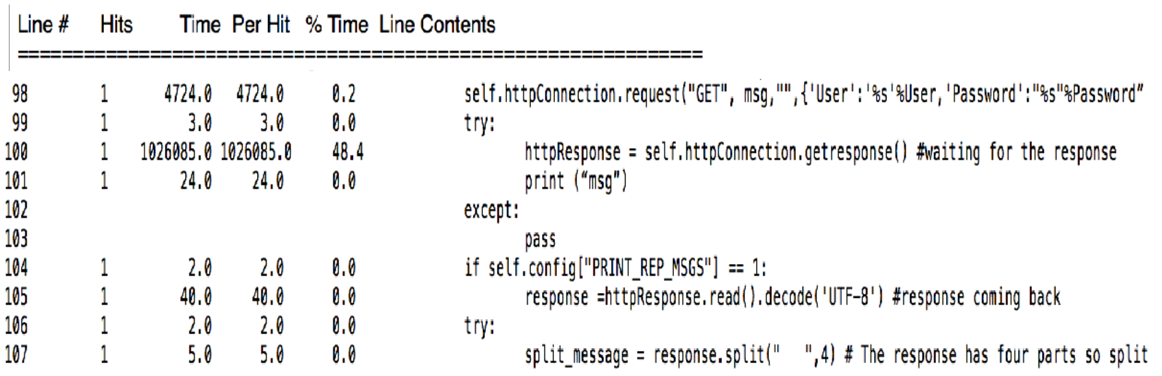
The resource manager initiates the bootstrap process where it assigns a role to each node that is started. On startup, each node checks for the role it has been assigned by the resource manager and fires the appropriate bootstrap script to assume its role. It adds fakenameserver (NS), global controller (GC) and the two local controllers (LC) for our configuration. It also adds a client host to each side, which simulates 10,000 users.

The GC initializes the workflow manager (WFM), WFM spawns workflow thread (WFT) for an application, WFT registers the application instance with NS and tries to get a proxy node allocated by WFM. There are no resources yet so the request will be on hold and a request-failed message is sent. The design requires WFT to try again and it runs exponential backoff to repeat request. WFM tries to get resources. Both the LCs boot up independently and try to register with the GC. They start a module called connection dispatcher, which allows data plane nodes to register with it. This process will make the resources available to WFM. WFM requests both the sites for resources, in the meanwhile WFT keeps polling. When resource update comes, WFM sends request to allocate a data plane node for proxy service. Proxy is initialized and WFM collects resources for other services. WFM has resources, starts application services, message and packet routing services. The internal services connect to the OpenADN gateway through a message level port (sPort) or a packet level port (tPort). After each service is initialized by WFT it connects to OpenADN socket.

4.6.3 Details of Optimization

Our work with OpenADN profiling and analysis, under different workloads conditions, has revealed three types of activities that consume large amount of time during the execution of the platform software: Creation and linking of the host and connecting services, polling of sockets and the sleep functions. We optimized the client host linking and the sleep function to see the effect on the overall execution.

Linking hosts and connecting services: When the client requests a web service, the service is connected to OpenADN through the pPort. The http client thread is run and the service is registered with the fakenameserver. The http server waits for the response from the service. The server waits for these services and clients to connect in what seems to be a wasteful loop. While it is necessary to suspend the operation of the loop in order to artificially synchronize the processes, indiscriminate use of sleep or similar functions would be quite wasteful. However, it was seen from line 100 of the program (Figure 4.14) that the 1s wait causes this program to take 48.4% of the time for execution of this function.

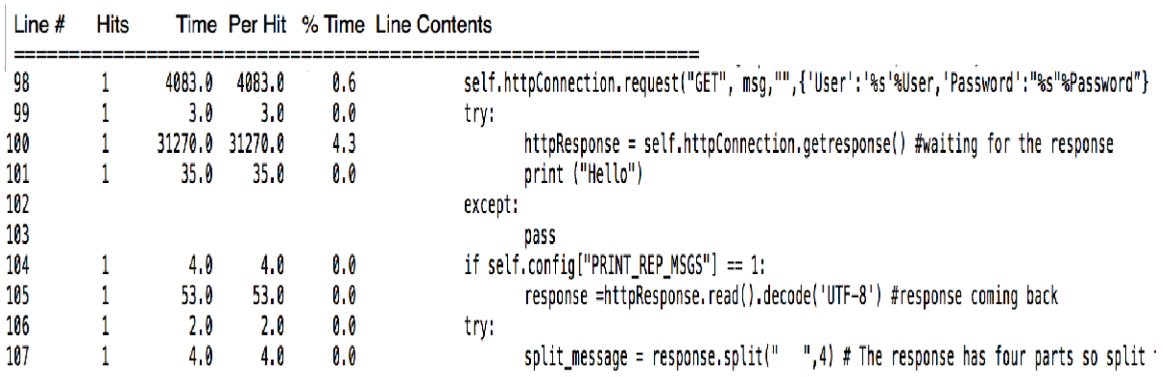


```
Line #   Hits       Time  Per Hit  % Time  Line Contents
==============================================================
 98       1      4724.0   4724.0     0.2         self.httpConnection.request("GET", msg,"",{'User':'%s'%User,'Password':"%s"%Password"
 99       1         3.0      3.0     0.0         try:
100       1   1026085.0 1026085.0   48.4                 httpResponse = self.httpConnection.getresponse() #waiting for the response
101       1        24.0     24.0     0.0                 print (“msg")
102                                              except:
103                                                      pass
104       1         2.0      2.0     0.0         if self.config["PRINT_REP_MSGS"] == 1:
105       1        40.0     40.0     0.0                 response =httpResponse.read().decode('UTF-8') #response coming back
106       1         2.0      2.0     0.0         try:
107       1         5.0      5.0     0.0                 split_message = response.split("   ",4) # The response has four parts so split
```

Total time taken: 2.11979 seconds

Figure 4.14 http client connection (1 s case)

We tried working with different wait times. When we change it to 1 ms we do not experience any change in the functionality of the software but the performance improves and it now just takes 4.3% of the time, giving more time to processing work (line 100 in Figure 4.15).



```
Line #    Hits      Time  Per Hit  % Time  Line Contents
==============================================================
 98       1      4083.0   4083.0     0.6          self.httpConnection.request("GET", msg,"",{'User':'%s'%User,'Password':"%s"%Password"}
 99       1         3.0      3.0     0.0          try:
100       1     31270.0  31270.0     4.3                  httpResponse = self.httpConnection.getresponse() #waiting for the response
101       1        35.0     35.0     0.0                  print ("Hello")
102                                               except:
103                                                       pass
104       1         4.0      4.0     0.0          if self.config["PRINT_REP_MSGS"] == 1:
105       1        53.0     53.0     0.0                  response =httpResponse.read().decode('UTF-8') #response coming back
106       1         2.0      2.0     0.0          try:
107       1         4.0      4.0     0.0                  split_message = response.split("   ",4) # The response has four parts so split
```

Total time taken: 0.728944 seconds

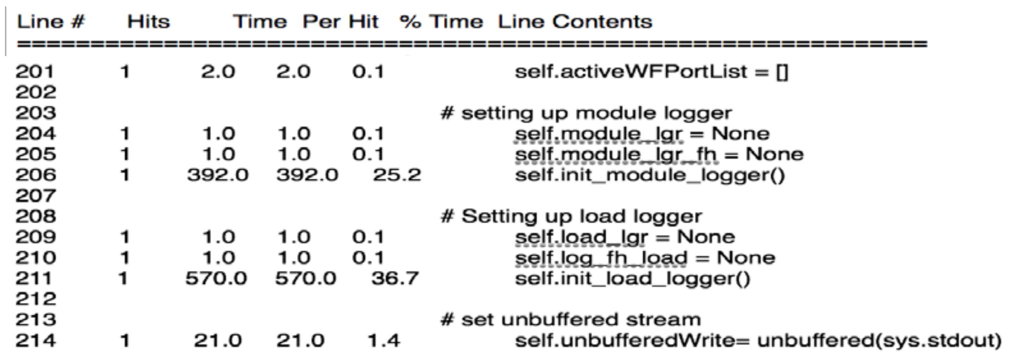
Figure 4.15 http client connection (1ms case)

From Table 4.7 we see the performance of this thread for 1s, 1ms and 1ns sleep times. The total time spent in wasteful waiting reduces as the sleep time is adjusted to a lower level. This is true for both the initial connection and during operation. The conclusion is that the program suspension duration needs to be carefully arrived at as such that the execution of the code is not affected i.e., connection of the client and the http service to OpenADN are not hindered.

Table 4.7 Performance of client connection for varying execution suspension times

Suspension Time	Hits	Time (μs)	Time/hit (μs)	%Time
During initial connection				
1s	1	1026085.0	1026085.0	48.4
1ms	1	31270.0	31270.0	4.3
1ns	1	26151.0	26151.0	5.9
During operation				
1s	24	898498.0	6898.6	42.4
1ms	15	558403.0	37226.9	76.6
1ns	9	337869.0	37541.0	75.8

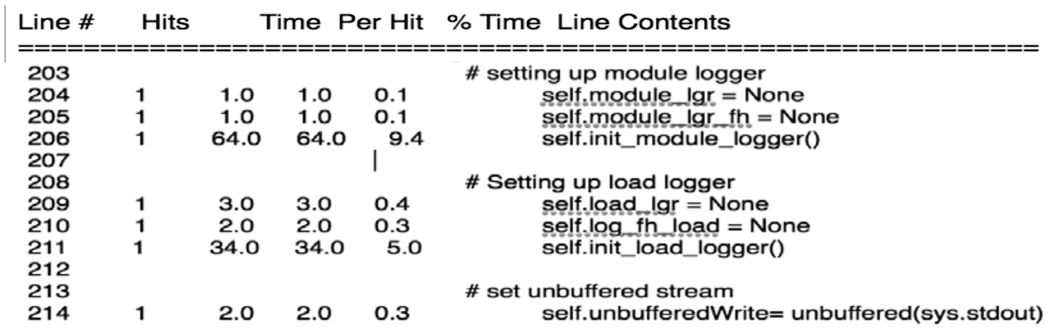
The Port initialization delays: This activity is required for connecting any node or service with OpenADN, its performance, therefore, becomes important. When profiling the initialization of communication ports, it is seen that most of the time is spent, a lot of it wastefully, on either security features or initialization of the logging files. Security features use internal libraries to generate keys and depend on the efficiency of this program. We chose to work with the logging feature, which was specific to OpenADN. Initialization of logging files could be done in a separate thread. By bypassing this, the logging times have been reduced by 89.8% for the pPort and overall reduction in the initialization time by 77.1% (Lines 206 and 211) .



```
Line #     Hits        Time  Per Hit   % Time  Line Contents
==============================================================
201        1          2.0     2.0     0.1            self.activeWFPortList = []
202
203                                            # setting up module logger
204        1          1.0     1.0     0.1            self.module_lgr = None
205        1          1.0     1.0     0.1            self.module_lgr_fh = None
206        1        392.0   392.0    25.2            self.init_module_logger()
207
208                                            # Setting up load logger
209        1          1.0     1.0     0.1            self.load_lgr = None
210        1          1.0     1.0     0.1            self.log_fh_load = None
211        1        570.0   570.0    36.7            self.init_load_logger()
212
213                                            # set unbuffered stream
214        1         21.0    21.0     1.4            self.unbufferedWrite= unbuffered(sys.stdout)
```

Total time for initialization 0.00679 seconds

Figure 4.16 Setting up the pPort

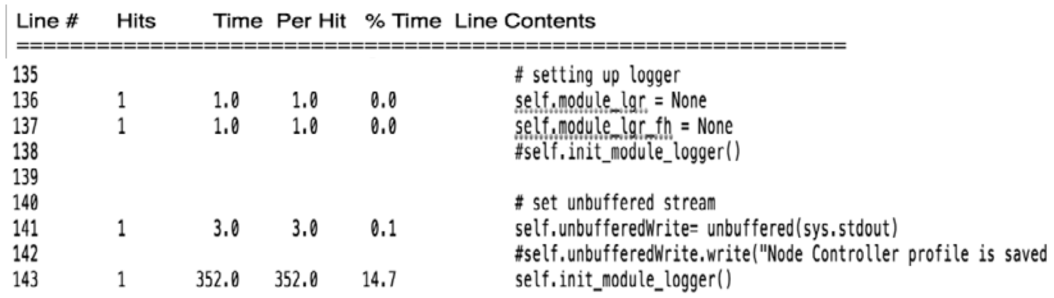


```
Line #     Hits        Time  Per Hit   % Time  Line Contents
=====================================================================
203                                            # setting up module logger
204        1          1.0     1.0     0.1            self.module_lgr = None
205        1          1.0     1.0     0.1            self.module_lgr_fh = None
206        1         64.0    64.0     9.4            self.init_module_logger()
207                                   |
208                                            # Setting up load logger
209        1          3.0     3.0     0.4            self.load_lgr = None
210        1          2.0     2.0     0.3            self.log_fh_load = None
211        1         34.0    34.0     5.0            self.init_load_logger()
212
213                                            # set unbuffered stream
214        1          2.0     2.0     0.3            self.unbufferedWrite= unbuffered(sys.stdout)
```

Total time for initialization 0.001553 seconds

Figure 4.17 Setting up the pPort (after optimization)

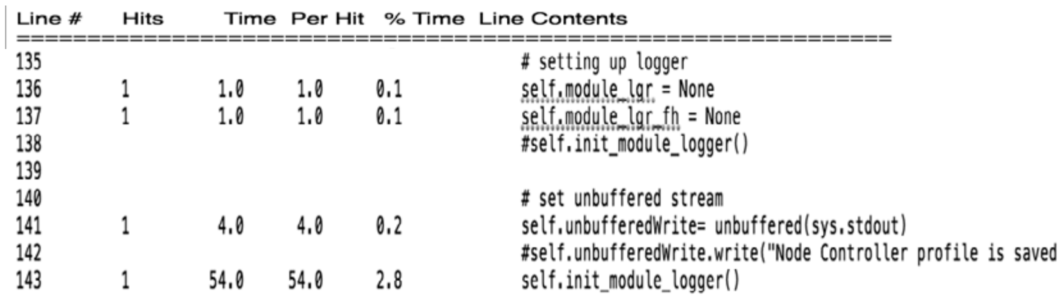
For the cPort the reduction for the logging part was 84.6% while the overall reduction was 19% as shown in. Figures 4.18 shows execution before changes and Figure 4.19 after changes (Line 143).



```
Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
   135                                           # setting up logger
   136         1          1.0      1.0      0.0  self.module_lgr = None
   137         1          1.0      1.0      0.0  self.module_lgr_fh = None
   138                                           #self.init_module_logger()
   139
   140                                           # set unbuffered stream
   141         1          3.0      3.0      0.1  self.unbufferedWrite= unbuffered(sys.stdout)
   142                                           #self.unbufferedWrite.write("Node Controller profile is saved
   143         1        352.0    352.0     14.7  self.init_module_logger()
```

Total time for initialization 0.002394 seconds

Figure 4.18 Setting up the cPort

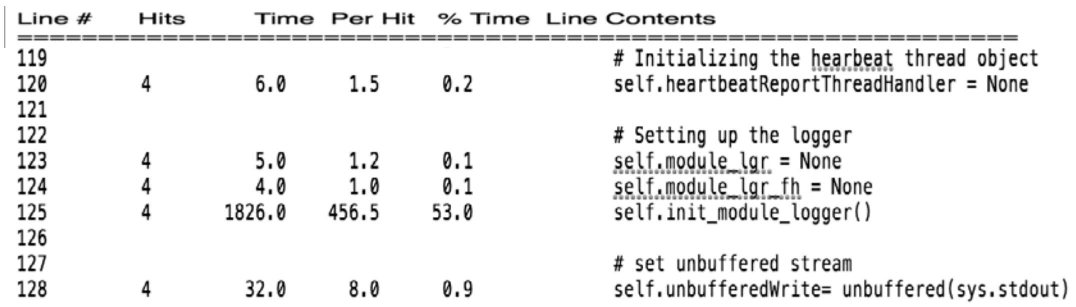


```
Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
   135                                           # setting up logger
   136         1          1.0      1.0      0.1  self.module_lgr = None
   137         1          1.0      1.0      0.1  self.module_lgr_fh = None
   138                                           #self.init_module_logger()
   139
   140                                           # set unbuffered stream
   141         1          4.0      4.0      0.2  self.unbufferedWrite= unbuffered(sys.stdout)
   142                                           #self.unbufferedWrite.write("Node Controller profile is saved
   143         1         54.0     54.0      2.8  self.init_module_logger()
```

Total time for initialization 0.001938 seconds

Figure 4.19 Setting up the cPort (after optimization)

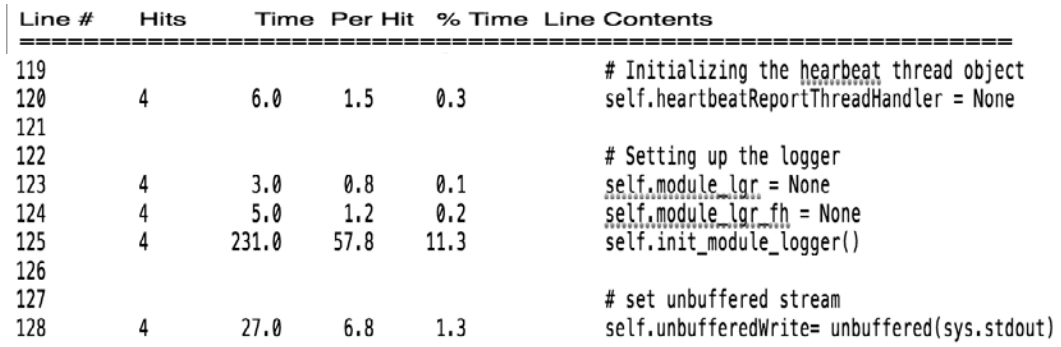
For sPort the reduction was about 87% with the overall reduction by 40.7% as seen from Figures 4.20 and 4.21 line number 125.



```
Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
   119                                           # Initializing the hearbeat thread object
   120         4          6.0      1.5      0.2  self.heartbeatReportThreadHandler = None
   121
   122                                           # Setting up the logger
   123         4          5.0      1.2      0.1  self.module_lgr = None
   124         4          4.0      1.0      0.1  self.module_lgr_fh = None
   125         4       1826.0    456.5     53.0  self.init_module_logger()
   126
   127                                           # set unbuffered stream
   128         4         32.0      8.0      0.9  self.unbufferedWrite= unbuffered(sys.stdout)
```

Total time for initialization 0.003443 seconds

Figure 4.20 Setting up the sPort



```
Line #    Hits       Time  Per Hit  % Time  Line Contents
==============================================================
119                                            # Initializing the hearbeat thread object
120        4        6.0      1.5     0.3       self.heartbeatReportThreadHandler = None
121
122                                            # Setting up the logger
123        4        3.0      0.8     0.1       self.module_lgr = None
124        4        5.0      1.2     0.2       self.module_lgr_fh = None
125        4      231.0     57.8    11.3       self.init_module_logger()
126
127                                            # set unbuffered stream
128        4       27.0      6.8     1.3       self.unbufferedWrite= unbuffered(sys.stdout)
```

Total time for initialization 0.002041 seconds

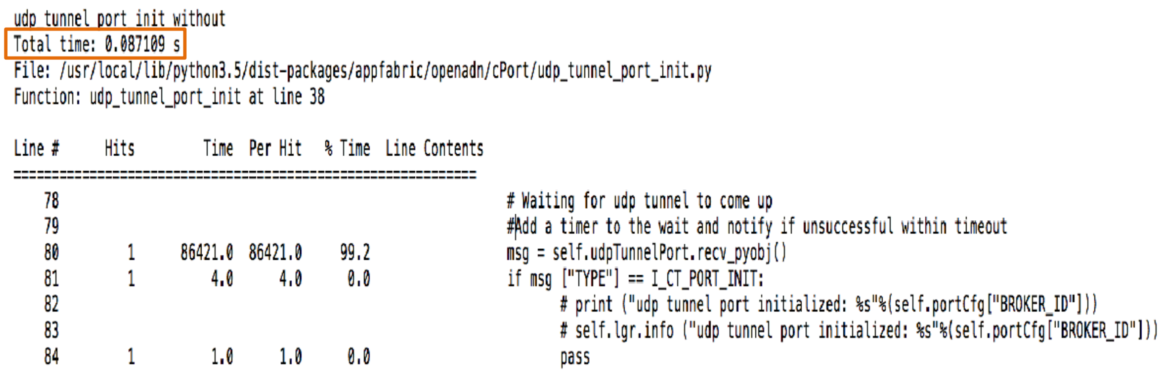
Figure 4.21 Setting up the sPort (after optimization)

Table 4.8 gives a summary of the time gained by optimization of initialization of various ports.

Table 4.8 Outcomes for port optimization

	Hits	Time	Per Hit	Time (Before Optimization	(After Optimization)	Reduction in activity time	Overall Reduction
pPort	1	392	392	61.9%	13.4%	89.8%	77.1%%
sPort	4	1825	456.5	53.0%	11.3%	87%	40.7%
cPort	1	352.0	352.0	14.7%	2.8%	84.6%	19.0%

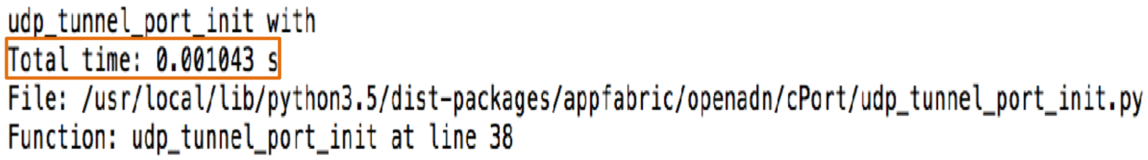
UDP tunnel port initialization: In starting up a cPort quite a bit of time is wasted in udp_tunnel_port_init. The concerned process initialized a communication port with the GC and waits for a reply. This happens because even though this process waits for a reply message, the message is not used. Since it is a UDP tunnel, this wait for the message could be omitted and the process may return after the port initialization. When this was implemented the UDP initialization time was reduced by 98% (Figures 4.22 and 4.23).



```
udp tunnel port init without
Total time: 0.087109 s
File: /usr/local/lib/python3.5/dist-packages/appfabric/openadn/cPort/udp_tunnel_port_init.py
Function: udp_tunnel_port_init at line 38

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
    78                                           # Waiting for udp tunnel to come up
    79                                           #Add a timer to the wait and notify if unsuccessful within timeout
    80         1      86421.0  86421.0     99.2  msg = self.udpTunnelPort.recv_pyobj()
    81         1          4.0      4.0      0.0  if msg ["TYPE"] == I_CT_PORT_INIT:
    82                                                   # print ("udp tunnel port initialized: %s"%(self.portCfg["BROKER_ID"]))
    83                                                   # self.lgr.info ("udp tunnel port initialized: %s"%(self.portCfg["BROKER_ID"]))
    84         1          1.0      1.0      0.0          pass
```

Total time 0.087109 (highlighted)

Figure 4.22 UDP tunnel port initialization

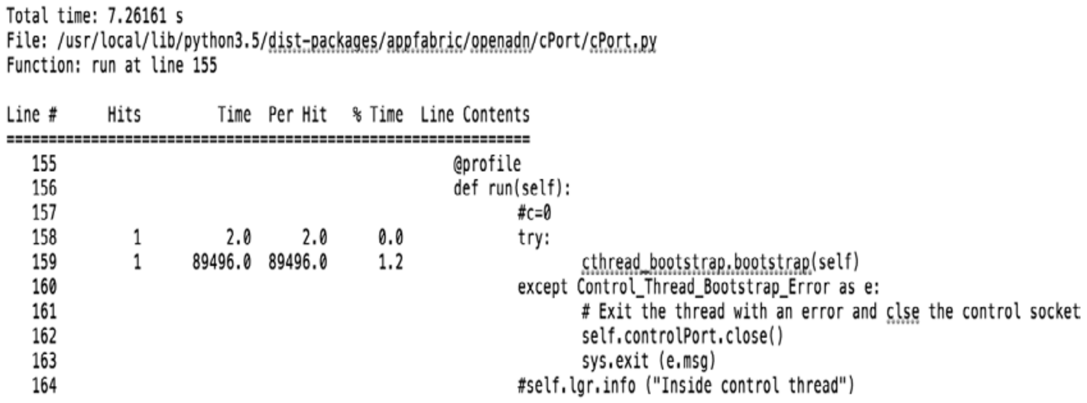


```
udp_tunnel_port_init with
Total time: 0.001043 s
File: /usr/local/lib/python3.5/dist-packages/appfabric/openadn/cPort/udp_tunnel_port_init.py
Function: udp_tunnel_port_init at line 38
```

Total time 0.001043 seconds (highlighted)

Figure 4.23 cPort without udp_tunnel_port optimization

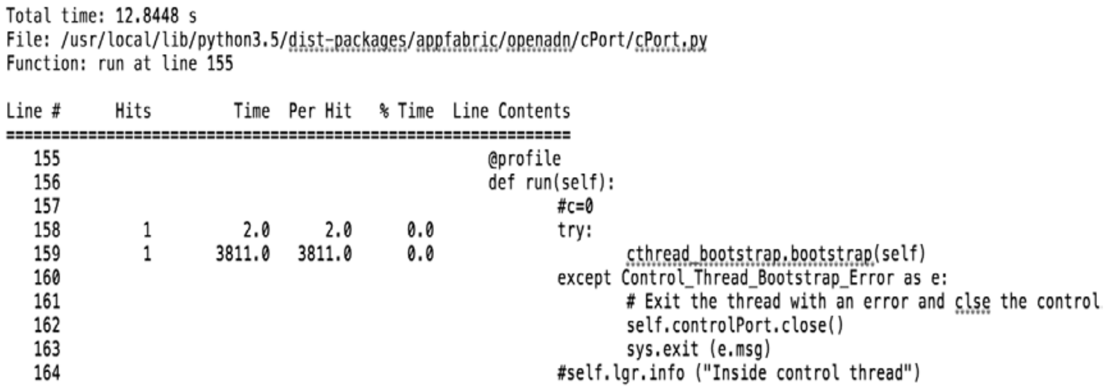
The cPort bootstrapping time was reduced by 95% (line 159 in Figures 4.24 and 4.25)



```
Total time: 7.26161 s
File: /usr/local/lib/python3.5/dist-packages/appfabric/openadn/cPort/cPort.py
Function: run at line 155

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
   155                                           @profile
   156                                           def run(self):
   157                                                   #c=0
   158         1          2.0      2.0      0.0          try:
   159         1      89496.0  89496.0      1.2                  cthread_bootstrap.bootstrap(self)
   160                                                   except Control_Thread_Bootstrap_Error as e:
   161                                                           # Exit the thread with an error and clse the control socket
   162                                                           self.controlPort.close()
   163                                                           sys.exit (e.msg)
   164                                                   #self.lgr.info ("Inside control thread")
```

Total time 0.089496 seconds

Figure 4.24 cPort bootstrapping without udp_tunnel_port optimization



```
Total time: 12.8448 s
File: /usr/local/lib/python3.5/dist-packages/appfabric/openadn/cPort/cPort.py
Function: run at line 155

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
   155                                           @profile
   156                                           def run(self):
   157                                                   #c=0
   158         1          2.0      2.0      0.0          try:
   159         1       3811.0   3811.0      0.0                  cthread_bootstrap.bootstrap(self)
   160                                                   except Control_Thread_Bootstrap_Error as e:
   161                                                           # Exit the thread with an error and clse the control
   162                                                           self.controlPort.close()
   163                                                           sys.exit (e.msg)
   164                                                   #self.lgr.info ("Inside control thread")
```

Total time 0.003811 seconds

Figure 4.25 cPort bootstrapping with udp_tunnel_port_optimization

4.6 Summary and Discussion

When software systems do not perform well, intuition or reading of code alone is not enough to provide reliable information on what could be wrong with the code. Many techniques have been traditionally used for system consuming physical resources. Multi-cloud management systems running in a virtualized environment can present a complex picture and evade optimization with the traditional techniques. It is necessary for both the CSPs and ASPs to use a platform that has been optimized for specific scenarios.

In this chapter we have presented the work that we have carried out to optimize complex multi-cloud platforms like OpenADN. Design choices at the time of the development of OpenADN govern the use of functions that might cumulatively consume substantial time. Non-blocking input-output in the form of polling or putting processes to sleep are an example of these. We have observed that the processes of such a platform might slow down if the use of these functions is not optimized. With a combined the technique of dynamic analysis and two-factor full factorial analysis, we have showed that the method is able to reliably and provably point out to the factors that need to be optimized to make the platform to perform its functions optimally. An optimally performing platform will consume less resources, time and will lead to lower operational cost.

Chapter 5

Placement of Virtual Network Functions on Multi-Cloud Systems

In this chapter we would discuss placement of virtual network functions on the virtual infrastructure obtained from multiple cloud service providers. The requirement here is to meet the quality of service parameters like latency and jitter as per policy agreed between the carrier and each of the cloud service providers or a cloud aggregator. The carrier would normally want the cost to be optimized without compromising on any of the parameters mandated by regulation or required for proper engineering of the service being deployed. In this work we use latency, as the representative parameter, to be constrained within the given threshold while minimizing the cost. This requires latencies of various clouds to be predicted at the time of actual placement. We have evolved innovative machine learning based prediction strategy for predicting latency at the time of commissioning of service for meeting end-to-end latency constraint. To make predictions accurate, appropriate attention has been paid to the dependence of latency on diurnal and also and short-term traffic variations. Since minimization of cost is the objective to be met, even for multi-modal cases, we have worked with a generalized random search method that works fast and is guaranteed to converge to the solution, or close to the solution. Two other requirements that have been imposed are the speed and the efficiency of placements. Speed is important for fast initial placement as well as timely reconfigurations in case of service not longer being able to meet the performance requirements. Efficiency is important from the viewpoint of the cloud service provider and implies that placement would be successful if the required resources can be assembled from the contracted clouds. Finally, all the methods and techniques that were evolved for multi-cloud placement of virtual network services have incorporation in a framework called Predictive – Adaptive Real Time (P-ART) framework. The work has been published in [10] [121].

5.1 Introduction and Motivation

Carriers perceive Network Function Virtualization (NFV) as a disruptive technological development. NFV allows network functions and appliances to be instantiated, in software, on computing and networking resources obtained from datacenters or cloud service providers. The combination of NFV and cloud computing holds a great promise for carriers. With these developments carriers can look forward to freedom from dependence on equipment vendors and their expensive proprietary equipment. Additionally, they get ease of service creation and phasing out, the flexibility of scaling and de-scaling, having points of presence closer to the users and avoiding a single point of failure. Cloud computing and NFV have a natural synergy that awaits full exploitation. It is expected that these two powerful paradigms would evolve together to support the requirements of virtual network services (VNS). The European Telecommunications Standards Institute (ETSI) specification of classification of cloud-native VNF implementations describes the creation of VNFs on different types of clouds [122].

One of the biggest challenges in deploying NFV over multiple clouds today is their inability to confer carrier grade performance to VNSs created using NFV paradigm [19] [20]. The Internet

Engineering Task Force (IETF) too has identified performance and guaranteeing the quality of service as open research areas and technology gaps in NFV [21]. The performance standards have been traditionally strict in telecommunications networks, with International Telecommunications Union (ITU) standards being adopted by most administrations. These standards prescribe stringent control over performance parameters like latency, jitter and packet loss [22]. The availability requirement is of the order of five nines (this translates to permissible downtime of just 5 minutes and 15 seconds in one year). In their infrastructure overview, ETSI has indicated latency and throughput constraints as the discouraging factors for the use of public clouds for hosting NFV.

One of the reasons for software versions of network functions, i.e., VNFs, not giving performance comparable to the specialized physical appliances is their creation over general-purpose hardware. In contrast hardware appliances are based on ASICS designed for good performance. The performance suffers further when these 'softwarized' functions are instantiated over clouds. To compound the problem, carriers have lesser control over flow of east-west traffic when network appliances move from their own switch rooms and transmission centers to the Cloud Service Providers' (CSPs') datacenters. Among the operational reasons for the deterioration of performance are the ease of creation, destruction, migration, and scaling of virtual resources (courtesy NFV), which provide the technical staff with opportunities for indiscriminate virtualization. Previous works have also shown that virtualization may lead to abnormal latency variations and significant throughput instability [37]. Even though researchers have proposed ways of improving the performance of virtual network functions [23][24], legitimate concerns still remain.

In the ultimate analysis, the advantages of the virtualization of carrier network services and their deployment over clouds are far too important than the current problems. This has motivated me to evolve the P-ART framework that will help alleviate some of the main concerns of carriers while deploying VNSs over multi-cloud systems including meeting the contracted performance and keeping the cost within the prescribed budget.

5.2 Contributions

The aim of this part of the research is to develop a framework for dynamic, predictive, adaptive and real-time placement of carrier virtual network services over multi-cloud systems. To this end, the techniques that we have evolved have been put together to form the P-ART framework. The main contributions of this work are summarized below:

1. Techniques for placement of VNSs to meet the carrier requirements of cost and latency and carrier objective of accuracy:

   i) Innovative predictive dynamic placement algorithm that takes care of changes in the state of the cloud environment to ensure the validity of the placement at the time of activation of a service. Noticeably, the algorithm works with complete service function chains (SFC), thereby taking into account the compute, storage and network links together, rather than the commonly followed path of placing VNFs individually. Comparison of our results with those in other works proves this point. As most carrier services are affected by latency, we choose to work with latency as an important performance measure. The work can be extended to other parameters following the same guiding principles.

   ii) We have approached the problem of the scarce availability of public datasets suitable for the in two ways. In the first method we built a queuing-theoretic model to generate train and test datasets. The other method consisted of implementation of the system on CloudLab [123].

iii) One of the important aspects of the framework is a novel method that refines the prediction algorithm, by taking into account variations in network latency because of temporally varying traffic conditions in the carriers' networks. Unaccounted, such variations cause a concept-drift, which affects the accuracy of predictions and makes them unreliable. For this, we introduce the concept of using time as a feature in training the predictive machine learning models. This makes the framework adaptive to diurnal traffic variations.

iv) Short-term traffic changes may occur because of events like a football match or an election rally. These changes do not follow patterns of diurnal traffic variations and need a different treatment. Retraining of models is an option but it is usually quite time consuming and expensive. Our framework uses incremental learning to keep the models up-to-date.

2. We explain in the related works section that, in general, Integer Linear Programming (ILP) and its variants give optimal solutions to the multi-criteria optimization problem but take significantly more time than other methods. This limits their utility in responding fast to the change of state of the multi-cloud system and the subscriber demands from the service during its actual operation. We have worked on an innovative placement strategy to carry put placements to optimize cost and keep latency within the specified threshold. In a first, we have used the random optimization as a viable method to achieve optimized placement. The algorithm converges to the global minimum even in the case of a multi-modal dataset.
3. The high speed of placements allows the CSP to make changes in the network dynamically, in real-time or near real-time, as the factors like demand, traffic congestion on links, availability of resources on various clouds change. The P-ART framework incorporates innovative techniques for making the placement fast with high acceptance rate. A high acceptance rate implies that a placement attempt would be successful every time, if enough resources are

available on the clouds.

5.3 Virtual Network Service Environment

Various services, like wired or wireless voice and data services, Internet services, content delivery, leased circuits and virtual private networks, provided by telecommunication companies have been considered in this work as carrier network services or simply carrier services. In the virtualized form they are referred to as virtual network services. Traditionally, networks providing these services have been built using specialized physical appliances and transmission links that are custom built for carrier-grade performance. Proprietary and closed nature of these appliances creates vendor lock-in leading to high cost, prolonged service deployment time, inflexibility in scaling and introducing new services. NFV and cloud computing provide a way to create network functions, in software, over inexpensive hardware resources. Such virtual functions can be linked with virtual network links to create VNSs. The VNSs result in open, flexible, scalable and less expensive networks that are not proprietary and thus prevent vendor lock-in. In the next sub-section we shall see the constituents of VNS along with the cloud set-up that can be used for hosting such services.

5.3.1 Constituents of a Virtual Network Service

In most discussions on VNSs, VNFs are the basic unit of placement. They exhibit functional behavior similar to their physical counterparts and have well-defined interfaces consistent with relevant industry standards. VNFs can be instantiated on virtual machines (VMs) obtained from datacenters, or from cloud service providers. All the instances of a VNF, for example, that of the core router function, would usually be hosted on one or more dedicated VMs on one or more clouds depending on the carriers' requirements and CSPs own policies regarding these deployments.

An SFC or a VNF forwarding graph is a set of VNFs, interconnected in a well-defined sequence, to route the packets [26]. They are connected in a manner similar to the way the physical appliances are connected in a traditional network [124]. IETF RFC 7498 describes each network service being implemented through one or more SFCs [125] [126]. The SFCs can also be hybrid in which the carrier retains some of the legacy physical network functions (PNFs) while virtualizing the other functions. The SFC may, therefore, consist of VNFs, PNFs, and the links among them. Figure 5.1 shows the components of an SFC and associated modules.

The broadband VNS, shown in Figure 5.1, is an SFC consisting of four VNFs, viz., an aggregation switch, a Border Network Gateway (BNG) and a core router. It also has multiple instances of a Physical Network Function (PNF), viz., Digital Subscriber Line Access Multiplexers (DSLAMs), retained from the legacy network. Each VNF has its own Element Management System (EMS), which interfaces the VNF to rest of the network [124]. The Operation Support System/Business Support System (OSS/BSS) of the carrier manages the VNFs and SFC through the EMSs.

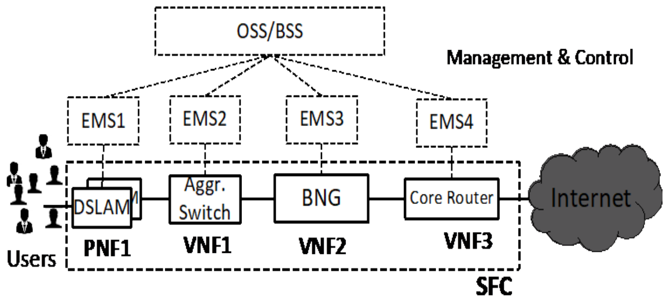


Figure 5.1 Broadband service function chain and associated modules

SFCs can be placed on the available clouds in a number of ways. CSPs, or the niche VNF as a Service (VNFaaS) providers, may offer commonly used network functions, which may be leased by the carriers to form an SFC. Alternatively, with a view to exercise more control over the performance parameters and cost, carriers may lease virtual machines, and associated resources, in the clouds and instantiate VNFs themselves. Unless otherwise stated, our discussion presumes the use of the latter method. Figure 5.2 shows an example of an SFC mapped to multiple clouds. It may be noted that we now have four VNFs, as the SFC has two types of BNGs. The Aggregation Switch is presumed to have a built-in load-balancing function for distributing traffic between the two forked paths. The EMSs have been omitted for simplicity. The end-to-end latency of the SFC would depend on how, when, and where the constituent functions have been placed. When an initially placed SFC does not meet the required conditions, it has to be reconfigured by scaling up functions or even moving the VNFs to different clouds.

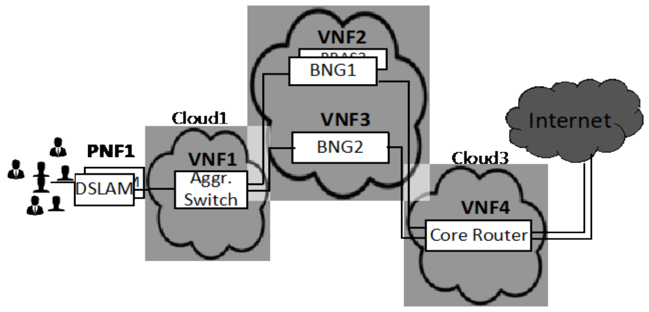


Figure 5.2. Mapping service function chain to the multi-cloud system

5.3.2 The Multi-cloud Hierarchy

Public cloud services like Amazon EC2, Google Cloud Services, and Microsoft Azure provide the advantage of relatively inexpensive resource leasing options. Big public clouds are multi-tenant and have a regional or international presence. These clouds can handle large volume, variety, and velocity of traffic. While a large public cloud does offer greater flexibility in obtaining resources and more analytical sophistication, taking all the data to just one public cloud would create traffic congestion and increase the access latency. Using a single cloud may also often result in a single point of failure leading to service failures because of cloud blackouts, which are not uncommon.

Additionally, the points of presence (PoPs) of even large public clouds may not be close to the carriers' subscriber clusters, and this would give rise to increased access latency. If the application calls for lower access latencies, then edge clouds closer to the subscriber clusters, offer a good solution. Carriers may also build their own private clouds, which they can customize and exercise more control over. This hierarchy of clouds – mobile-edge, private, and public – forms a multi-cloud system that can be designed to provide a combination of features like low latency, high storage, complex computations, lower cost, and better security.

5.3.3 Representation of the Tenant Profile

In this work, a cloud tenant (in our case, a carrier) profile is represented as a tuple $<c_N, v_1, v_2, \ldots, v_m, p>$, for each request. Here, $v_1, \ldots, v_m$ represent the VNFs in the order of traffic traversal in a linear chain. The term c_N is the native cloud for the tenant to which it is parented, and through which its traffic enters an SFC. The desired packet rate is represented as p packet/second. Multiple tuples can be used to represent branched traffic flows. Other stipulations like latency threshold (L_{th}) are part of the SLA. All the requests of the tenant are consolidated to calculate the required number of instances of each VNF and inter-VNF links of appropriate capacities. The cloud topology is represented by the graph $G_c = (C, T)$, where C is the set of available clouds $(c_1, c_2, \ldots, c_k)$ and $t_{i,j}$ are the inter-cloud links. The CSP (or a cloud aggregator who integrates services from multiple clouds) carries out the task of mapping service chains onto the available clouds to achieve optimal results for the carrier. In our case, optimality refers to the least-cost solution that meets the end-to-end latency threshold requirement.

5.4 Problem Definition

In this section, we disscuss some of the key outstanding problems in dynamic placement of multi-cloud carrier VNSs, that we attempt to handle in the P-ART framework.

5.4.1 Achieving Dynamic Placement in Multi-cloud Systems

Some carrier services may be fairly static, e.g., fixed voice network. Thus, the requirement of the number of instances of VNFs and link capacities only change slowly over time. On the other hand, some services may be extremely dynamic, requiring a change in number and types of VNF instances, re-dimensioning of links and changes in the offered features of the service very frequently. An example of such a service is an intelligent network service, like televoting, that is in TV reality shows. Different TV reality shows may require different features and the number of voters may swing unpredictably during the voting window. If the CSP only offers largely static placement with reactive and relatively slow modifications, then the programs' requirements may not be met.

The bottom line is that both, the dynamic and static services, require the CSP to scale VNF capacities or links, albeit at a different rate. Dynamic services may be more demanding in terms of types and number of instances of VNFs and link resources and may even require migration of VNFs from one cloud to another to be able to continuously meet the cost and end-to-end latency constraints. A dynamic placement algorithm that would monitor the SLA parameters and proactively cause changes in the amount of resources, and the combination of clouds, to meet all the requirements at all times, is still a challenging issue.

5.4.2 Optimizing the SFC Performance

When the data are high dimensional and multi-modal, optimizing placement of individual VNFs may not achieve the global minimum. Placing SFCs as a unit yields better results. The opportunity to achieve the global minimum for the parameter being optimized is available when placing the SFC. If sufficient resources are not available to implement full-service chains, then the request may be rejected or, if the policy permits, degraded service (for instance without firewall) may be provided [26] [27]. In this work, we only consider complete SFC placement. The case where the customer accepts degraded performance due to low-capacity chain placement or partial functionality due to incomplete chain placement is left as future work.

5.4.3 Meeting the Cost and Latency Constraints

From the carrier's perspective, the placement problem boils down to placing network functions in such a way as to meet the cost objectives under latency constraints. At the commencement of the VNS and during operation, the placement problem needs to be repeatedly solved to ensure that the carrier requirements are continually met. Performance criteria vary from service to service. For the carrier services like voice, broadband, and content delivery some of the common factors are jitter, packet loss, latency, and throughput. ITU standards for QoS parameters in carrier networks are available in [22]. Latency is one of the most important criteria for most services, and we have taken that as a reference performance parameter. The framework can be extended to include other criteria as well.

5.4.4 Speed and Efficiency of Placement

Carriers want short placement and reconfiguration time so that the solution can be useful in an operational network. The CSP wants the solution to have the high success of placement requests such that utilization of the virtual resources increases. When the system cannot place despite the availability of resources, it has low efficiency. In such cases, CSPs lose by way of unused resources and possible breach of SLA.

5.4.5 Interference Among VNFs

For optimizing their own cost, a CSP may instantiate a number of VMs on the same physical machine (PM) or a number of virtual links on the same physical inter- or intra-cloud links. Also, VNFs of more than one service provider may be instantiated on the same PM. In some cases, pre-instantiated VNFs may be shared among carriers. Sharing of physical and virtual resources not only cause performance concerns but could also give rise to security concerns. For our work, we have worked on the premise that VNFs of different types, belonging to a carrier are on different VMs.

5.4.6 Problems Addressed in this Dissertation

The following issues have been specifically addressed in this work:

1. Dynamic placement of the complete SFCs belonging to a VNS.
2. Meeting the specified performance and cost criteria.
3. Prediction of latency using machine learning as a basic input for the placement algorithm.
4. Refining the prediction by handling the temporal variation of traffic, unplanned short-term spikes in traffic and the time lag between planning and commissioning of SFCs.
5. Fast placement of SFCs with high success rate.

5.5 The Proposed P-ART Framework

In this section, we describe our framework and approaches for tackling the challenges discussed in Section 5.4. We also describe how the refinements mentioned were carried out to achieve the desired solutions. The methods described here can be used for carrier networks as well as in the enterprise environment. For our studies, we will consider the placement of the SFC shown in Figure 5.3.

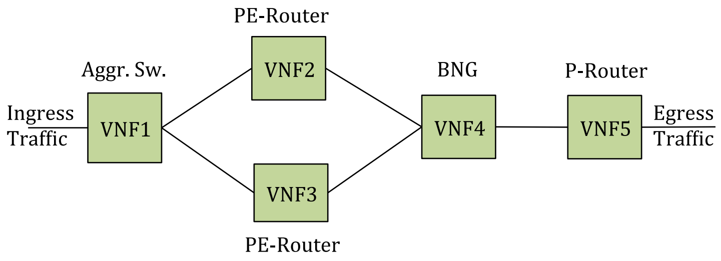


Figure 5.3. The configuration of the experimental service chain

5.5.1 Information Available from Carriers and CSPs

Carriers, who request service chain placement, provide information about the performance required from a VNS, and the number and structure of the SFCs and the VNFs to be instantiated. A VNS may have one or more SFCs. The i^{th} SFC, S_i, can be represented in terms of the constituent VNFs, i.e.,

$$S_i = \langle C_N, \mathrm{vnf}_1(i), \mathrm{vnf}_2(i), \ldots, \mathrm{vnf}_n(i), p\rangle \qquad (1)$$

Where C_N is the native cloud and p is the maximum packet rate through the chain. The native cloud is usually the point of presence (PoP) of the CSP, closest to the carrier, and provides interconnection to the carrier. The CSP may provide an option to connect to PoPs at other locations. This gives the carriers a choice to have traffic ingress points close to the customers. The design is to be carried out such that the costs of the network, as well as latency in reaching the cloud system, are kept to the minimum or below a given threshold value.

An SFC is represented as a forwarding graph of the type $G_v = (V, E)$, the nodes V being virtual network functions and edges E the virtual links among these functions. The demanded capacity of i^{th} VNF, vnf_i $(i \leq n)$ is expressed as v_i^c in the same integrated units as the cloud capacities (shown in Table 5.2). An integrated figure represents the compute capacity c_k, of a cloud k, consisting of a certain amount of processing, memory and storage components. However, there is no integer constraint on the VNF capacities. These are mapped onto resources in the available clouds represented as another graph $G_c = (C, T)$, where C represents the set of clouds with physical/virtual infrastructure and T the set of links t_{ij} among them. The state of a cloud k, at any specific time, would involve the cloud compute and link capacities –installed capacities denoted as $c_k^{(c)}$ and $t_{kj}^{(c)}$, and the corresponding used capacities are $c_k^{(u)}$ and $t_{kj}^{(u)}$. The carrier provides the maximum expected packet rate p for each request originating from a cluster of subscribers. The expected end-to-end latency is specified by the carrier in terms of a latency threshold (L_{th}). The CSP consolidates the VNF requests and packet rates required for each type of chain to allocate resources in an optimum way. Table 5.1 gives the symbols frequently used in the work.

Table 5.1 Symbols used

Symbol	**Description**	**Symbol**	**Description**	**Symbol**	**Description**
c_k	Cloud *k*	c_N	Native cloud	$c_k^{(u)}$	Used capacity of cloud *k*
C	Set of all clouds available	$v_i^{(c)}$	Capacity demand for VNF i	$t_{ij}^{(u)}$	Used capacity of the link between clouds *i* & *j*
t_{kj}	Link from cloud k to j	$c_N^{(c)}$	Equipped cap of native cloud	p	The maximum expected packet rate
T	Set of all inter-cloud links	$c_N^{(u)}$	Used cap of native cloud	m	No of clouds selected
v_i	*i*th VNF	$c_k^{(c)}$	Installed capacity of cloud k	vnf_i	The *i*th VNF in the SFC
V	Set of VNFs	$t_{ij}^{(c)}$	Capacity of link between clouds *i* & *j*	L_{th}	Latency threshold
n	Types of VNFs	$V_i^{(c)}$	Capacity demand for *i*th VNF	C_B	Cost budget

Some of the important constraints subject to which the cost optimization is carried out are:

- The number of instances of each type of VNF across all the used clouds, for any carrier, should not exceed the number of licenses for that function type paid for by the carrier.
- To place any chain, at least one instance of each type of VNF needs to be instantiated.
- The total capacity of each type of VNF placed on any cloud *k* should not exceed the capacity available in the cloud.
- At any given time the sum of the traffic flows, due to all service chain placements, between any two clouds *k* and *j* should not exceed inter-cloud link capacity $t_{kj}^{(c)}$.
- The end-to-end latency, L, of any chain should not exceed the specified threshold L_{th}.
- While the cost is optimized, the carrier may additionally specify a budget C_B for it.

The framework requires that the CSP lays down its policies regarding tariffs, integrated virtual resource capacities, clouds offered, the arrangement with other cloud providers, cloud and link capacities offered.

5.6 Predictive Adaptive Real Time Strategy

The placement solution optimizes cost and constrains the end-to-end latency below the specified threshold, L_{th}. We assume that the design for instantiation of SFCs, belonging to a VNS, is ready at time *t*, but actual placement is yet to happen. In other words, the placement problem has been solved at time *t* for the placement and activation that will actually take place at time t_1. Predictive placement is used to take care of the change of state of the clouds because of this time difference. Using prediction of the latency, as the basis for design, also takes care of the large number of infrastructure and network level parameters that interact in a complex way to decide the end-to-end latency. In addition to these, the background traffic in the network affects the latency experienced by the subscribers of the VNS being placed. Therefore, taking care of the background diurnal traffic variations in the network makes the prediction of latencies more accurate and system more adaptive to such changes [127]. Short-term surges in traffic, due to events like a football match also affect latencies during the event and should be accommodated by dimensioning and reconfiguring the SFCs. This renders the system more responsive (and near real-time) in terms of latency predictions. We have taken into account all these factors in formalizing our prediction algorithm. Latencies so predicted are then used to select a suitable subset of least-cost clouds meeting the latency constraint. The complete algorithm is given in Algorithm 5.1.

Algorithm 5.1: PLACE_SERVICE_CHAIN (client_demands, csp_data, cv_model)

```
1: Set up cloud data          // all $c_k \in$ C and $t_{k,j} \in$ T
2: Set up client data         // all $v_i \in$ V
3: Latency threshold ← $L_{th}$
4: Cost budget ← $C_B$
6: NCloud ← $c_N$   // Native Cloud
7: $v_i^c$ ← capacity demands for $vnf_i$
8: $n$ ← length of the service function chain (number of VNFs)
9: native ← true   // set native to 1 if native cloud is used else 0
10: if (native == 1)          //place as many VNFs as possible in the native cloud
11:        for $v_i$, $i$ =1, $n$
12:            if $c^c_N – c^u_N > v_i^c$ // native cloud has unused capacity
```

```
13:                pop $v_i$
14:                $c^u_N \leftarrow c^u_N + v_i^c$        // update cloud capacity
15:            else
16:                break
17:            end if
18:        end for
19: end if
20: if $V$ != 0        // for remaining vnfs
21:        call **RANDOM_SELECTION(*C*,** *cv_model*, *r_clouds*)   //get a set of lowest cost clouds
22:        sort ascending *r_clouds* on cost   //set of smallest latency clouds
23:        while $V$ != 0
24:            place vnfs        //on sorted clouds
25:            update capacity
26:            update bandwidth
27:            update vnfs_placed status
28 :        end while
29: end if
30: if all_vnf_placed & latency of chain < $L_{th}$ & cost of chain < $C_B$
31:        output placement details
32: else
33:        report failure to place
34: end if
```

The essential elements of the placement process can be understood like this: the placement process takes care of the change of state of the cloud system by predicting latencies at the time of actual activation of the SFCs. This obviates the need for drastic changes soon after placement or a reconfiguration. Prediction is, thus, an essential element of the framework. Having said that, the prediction methodology needs to be robust against traffic variations. With this, the framework becomes adaptive to placement time and traffic variations. To make the framework fast, responsive, and useful in real-time, further steps need to be taken. For this, short-term traffic variations are taken into account.

Two other important factors that need to be taken care of are *speed* and *acceptance rate* of placement. Fast placement algorithms would allow continuous optimization by making real-time changes (e.g., migration) possible, when the need arises, during the operation of the network. For dynamic scaling, a fast algorithm would be able to place hundreds or thousands of functions in sub-minute time frame. At the same time, a 100% acceptance rate implies that the algorithm totally accurate and is able to satisfy all requests for placing SFC, subject to capacity being available. This contributes to the avoidance of repeated attempts at placement and saves time and money.

Algorithm 5.1 is called for placement and reconfiguration. The cloud and client data are initialized based on the CSP resources and the client request and policies (*lines 1-5*). A separate process produces a trained model *cv_model* using the training data (**X** $\leftarrow$ feature_set and **y** $\leftarrow$ labels), which is available to the placement procedure. The placement normally begins with the native cloud (this can be overridden in *line 9* by setting *native = 0*). The algorithm accommodates as many VNFs as possible in the native cloud (*lines 10-18*). For the remaining VNFs, the chosen support vector regression module predicts the latency of various clouds. This procedure uses Algorithm 5.3 (procedure RANDOM_SELECTION) to select the set of *m* least-cost clouds that meet the latency requirements. The number *m* can be decided to start with enough capacity to place all the VNFs. For the least-cost set, the algorithm calculates the assignment of VNFs in the sequence in which they appear in the SFC. The final cost and latency are reported (line 31). If the clouds are exhausted, and placement has not completed, then failure to place is reported. If this case happens frequently, then the number *m* needs to be increased.

5.6.1 Handling Change of State of the System by Predictive Placement

The cost of placing an SFC is a function of the set of clouds C_s $(C_s \subseteq C)$, where C is the set of all available clouds), selected to place the virtual network functions and the amount of computing, storage, and networking resources consumed. End-to-end Latency (L) of the SFC depends on a number of factors prominent of which are, a) the installed and used capacities of computing, networking and storage resources in the physical servers and the links, b) the traffic pattern on the links, c) the types of network functions sharing the servers, and d) the distance between clouds. These factors together constitute the state S_t of the multi-cloud system at time *t*.

As the system operates, the number of tenants and their workloads change, causing the state to change. The amount of latency introduced in a placement by the state of the cloud, therefore, changes over time. Given the state St at the current time t can be observed or computed using assumptions about the type of traffic, e.g., Poisson, service times and the queuing discipline. The process of planning service function chains, creating virtual resources to host network functions and booting them up takes time [128]. Loading the network function software for various VNFs, chaining, acceptance testing, and commissioning need additional time. Initial placements and reconfigurations planned based on calculations at time *t*, and the state $S_{t,}$ are actually carried out at a time t_1. In due course, parameters may change and require fresh reconfiguration [129].

Figure 5.4 shows the SFC to be placed and the available clouds. Used and installed compute capacities are shown, in integrated units of Table 5.2, within the clouds, and so are the used and installed link capacities in M (Megabits) or G (Gigabits) per second. At time *t*, the assessed end-to-end latency is 20ms. When the actual placement and activation takes place at time t_1, the latency turns out to be 50ms. This increase may cause SLA violation right at the inception and trigger reconfiguration of the chain. When this happens for several service chains, it may lead to a heavy penalty to be paid by the CSP and a loss of customers and revenue for the carrier for not being able to provide service. When the states of the target clouds are known, the set of least-cost clouds, which give cost and latency below the stated thresholds, can be determined.

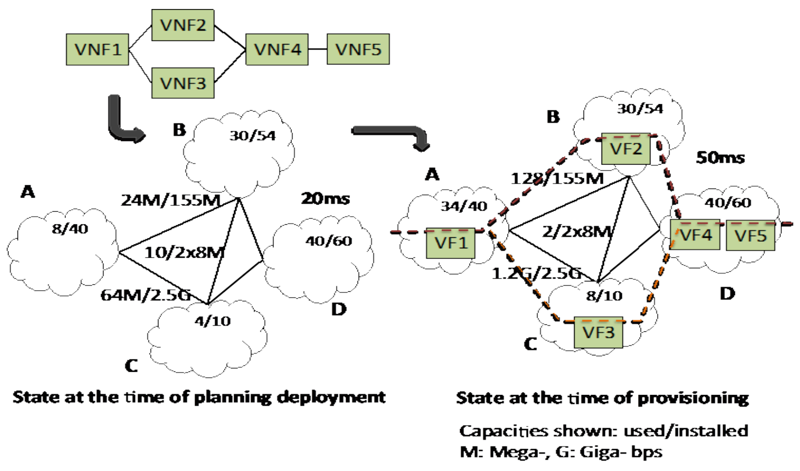


Figure 5.4. Need for predictive placement

Thus, if the state St_1 at the time t_1 can be predicted, and the placement is carried out based on this state, then the placement remains consistent with the requirements. This is demonstrated by our empirical study given in Section 5.10.

The placement procedure: In an operational CSP set-up as well as the carrier network, a large amount of useful labeled data is available. This data can be curated for use with supervised machine learning techniques. As the speed, simplicity, and accuracy are of concern, we worked on a prediction technique that could be applied repeatedly for the selection of a set of cloud consistent with the objectives of the framework. A review of the literature given in Chapter 3 shows that many supervised machine-learning techniques have been used in cloud computing settings, such as Artificial Neural Networks (ANNs), Bayesian networks, Ensemble classifiers and Support Vector Machines (SVMs). We worked with a number of methods and found interesting results with a well trained and tuned support vector regression (SVR). We discuss the results given by some well-known stock algorithms to justify the reason for our choice in Section 5.10.6. In general, SVR offers the advantage of a unique global minimum as it solves a convex optimization problem. Also, it is amenable to incremental learning. We have observed that it adapts well to multi-modal cases where the latency is time variant and needs multiple models to fully capture the actual situation. Well-tuned and trained models generalized well from training to the production environment. The results of our experimental evaluation are given in Section 5.10.

5.6.2 Incorporating Temporal Variation of Traffic in the Model

We will show, in a later section, through our empirical *analysis* that taking diurnal traffic variations into account improves prediction of latencies. In carrier networks, there is temporal and spatial variation in traffic demand because of time differences and patterns of use. The amount of traffic flowing through the virtual devices and links varies from place to place and hour to hour. This affects the latency experienced by the subscribers of the carrier's VNS. If the provider over-provisions the resources to meet the surge in traffic in the busy hour, then resources may lie unused at other times. On the other hand, if enough resources are not provisioned, in an attempt to reduce cost of the deployment, then traffic may be lost along with the associated revenue. Figures 5.5 (a) and 5.5 (b) show hourly variation of the actual traffic on a 100 Gbps link from Chicago to Seattle and 10 Gbps link from Los Angeles to San Jose [130].

The traffic that a carrier routes through the VNFs consists of streams of voice, video, and data with different probability distributions. Each of this traffic varies independently in the time domain. The aggregate traffic in the CSP's network is a composite of all the tenants' traffic and has a complex distribution. The traffic flows continuously as data streams and has properties of big data [131]. In such a dynamically changing and non-stationary environment, the data distribution changes over time, causing the phenomenon of concept drift [132]. The drift is characterized by the change in the density function that is, in turn, reflected by the change in the shape of the traffic distribution or its statistical properties like mean and variance. Thus, the joint distribution p_t of the predictor variables (X) and the labels (y) would change dynamically over time such that at time $t_0, t_1, \ldots, t_n$ the following relationship (2) holds for all X.

$$p_{t0}(\mathbf{X}, \mathbf{y}) \neq p_{t1}(\mathbf{X}, \mathbf{y}) \neq \ldots \neq p_{tn}(\mathbf{X}, \mathbf{y}) \qquad (2)$$

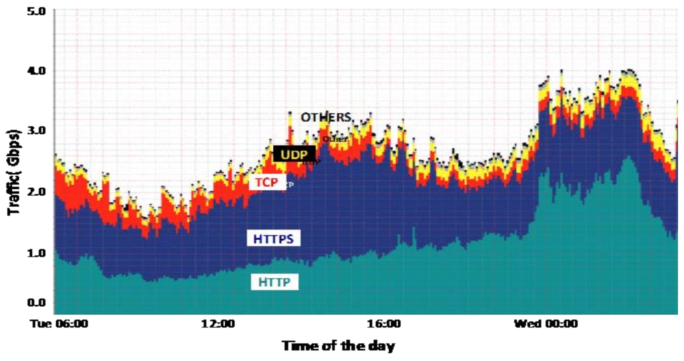


Figure 5.5. (a) Traffic variation on Chicago-Seattle link

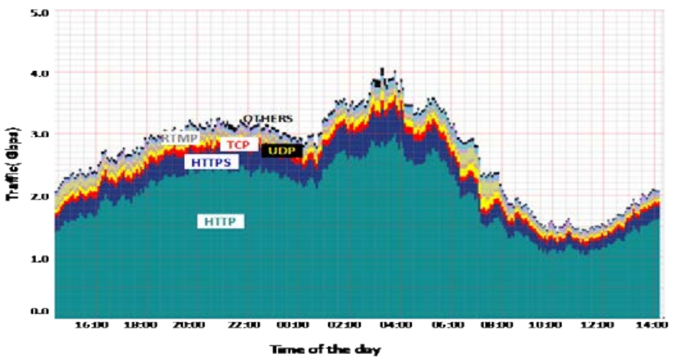


Figure 5.5 (b) Traffic variation on Los Angeles-San Jose Link

Solving the diurnal traffic variation problem: The solution evolved by us takes care of the concept drift to ensure more accurate traffic predictions. A single SVR model works well in situations where there is no sizable ambient traffic from other applications and network services. However, SVR by itself does not take care of the time-varying nature of the traffic present on the links from other voice, data, and video applications. To handle this, we incorporate time as a feature by allocating numerical codes to windows.

Researchers have experimented with both fixed and adaptive window methods to handle concept drift in real time situation. In the case of fixed windows, the data is segregated into many small windows to have lower overall generalization errors as compared to a single window situation [132]. The utility of the fixed window sizes, for topological data analysis, has been shown, under certain conditions, by the authors in [133]. A window of a certain minimal fixed size allows learning concepts because the extent of drift is appropriately limited [134]. In Adaptive Windows [135], the window size is changed so that the difference in errors (ϵ), given by a point in two neighboring windows, is bounded by a small value δ such that $\epsilon_t - \epsilon_{t-1} < \delta$.

To achieve a good compromise between prediction accuracy and complexity, our method has the simplicity of a fixed number of windows and is also flexible to include a variable number of traffic data points depending on the frequency of variations in different windows. Consequently, we call this method fixed-time variable-points (FTVP) window. SVR models are trained, one for each window, to tackle the effect of the concept drift. While even as few as two windows give an improvement in prediction, finding the right number and sizes is a matter of optimization. A larger number of small windows may give more accuracy, but would produce a larger number of models and would necessitate maintenance of all of them. This concept allows introduction of time as one of the features in the training examples. In a sense, each example carries a time-stamp, which makes it a member of a particular FTVP window. When a prediction for a new point is made, the time feature will cause the framework to use the model appropriate for the corresponding time window. In our experiments, this method gives far lower prediction root mean squared error (RMSE) and absolute error ratio (AER) than a single integrated windowless model.

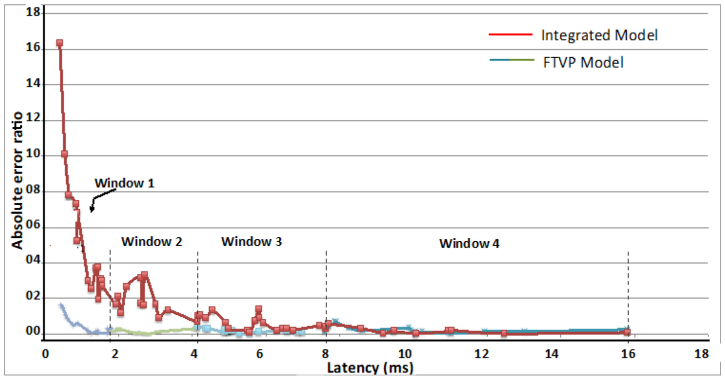


Figure 5.6. Comparison of generalization error with an integrated model and FPTV model

To validate the FTVP concept, we created a trained SVR model using a single window (full integrated dataset) and separately for each of the four selected FTVP windows. In Figure 5.6, we show a plot of the absolute error rate versus the latency for both cases. The motivation for using multiple training datasets, using time as one of the predictors, becomes amply clear. The errors, in general, remain more controlled in the FTVP case.

5.6.3 Corrections for Short-Term Traffic Variations

In an operational network, the dynamicity of the environment would render the trained predictive models obsolete, if the effects of the short-term changes in the traffic are not accounted for. Short-term variations are caused by events like festivals, game tournaments, or rallies. If the effect of short-term changes in traffic is not taken care of, latency prediction and consequent placement decisions may not be correct. Since retraining of all the models would entail prohibitive time and cost, we have used an incremental update of the models [132].

Choice of SVR for prediction makes incremental learning easier to understand. In SVR, the support vectors are the only points that determine the decision surface. They also satisfy the Karush-Kuhn-Tucker **(**KKT) conditions [136]. Each new point generated because of the change in traffic is checked for being a support vector. If it is a support vector and improves the overall model for future predictions, then it is included. If this becomes time-consuming, due to continuously generated traffic data, training in small batches speeds up the process. Support vectors can be separately found for each batch of fresh points, and they can be included in the model only if they improve it. Algorithm 5.2 gives the incremental training algorithm. We see in the next section that this contributes positively to the model empirically.

The initial training process creates a set S = {x_s, y_s} of support vectors that decide the decision surface. Algorithm 5.2 starts with the solution function f(t) at time t in terms of the initial training dataset T = {(x_i, y_i), *i* = 1, …, n} $x_i \in R^n$ and $y_i \in R$. The set of support vectors at this time are S(*t*). For the time *t+1* for which the model needs to be incrementally updated each of the new example {$x_{new}(t)$, $y_{new}(t)$} is received in the time window *(t, t+1)*, the algorithm checks if the new point is a support vector. The new support vectors are incorporated in the set S(*t+1*) if they improve the performance of the model as indicated by reduced mean squared error. Our simulations given in Section 6.6 also support this argument. The simplified algorithm is given below:

Algorithm 5.2: TRAIN_REAL_TIME (T, x_{new}, y_{new})

```
1: //Initial training set T = (($x_1$, $y_1$)...($x_n$, $y_n$))
2: f (t) = A(T)  //Training done at time t
3: f(t) : S(t)      //S(t) is the set of support vectors at time t
4: Initialize S(t+1) to S(t)
5:for all {$x_{new}$, $y_{new}$} in the window (t, t+1)
6:      if $x_{new}(t)$ : $x_s$ and $y_{new}(t)$ : $y_s$  // new point is a support vector
7:           S(t+1) = S(t+1) ) $\cup$ ($x_{new}$, $y_{new}$)
8:      endif
9: endfor
10: output f(t+1) : S(t+1)            //updated model at t+1
```

The removal of support vectors when the short-term traffic condition that created them has passed is left as a future work.

5.7 Cost optimization

5.7.1 Optimization of Cloud Selection by Random Search

An important part of the solution is to select the set of clouds that would be used for placing the VNFs of an SFC such that the total placement cost is the lowest possible, within the budget CB specified by the carrier, and is consistent with the latency constraints, i.e., $\sum_i l_i \leq L_{th}$ where l_i is the latency within ith cloud, and its link to the next cloud and L_{th} is the threshold given in the SLA. Following Occam's razor, we looked for an algorithm that would be simple and yet effective in meeting the real-time requirements. Algorithms like A-Star are efficient in finding a low-cost walking path from one node to another. Even with one parameter, i.e., the length of the path, its time complexity can degenerate to exponential.

A naïve approach is to search for *m* lowest cost clouds (enough to meet the capacity requirements), one at a time out of total *n* ($m \leq n$) such that the total cost (in terms of cloud resources and links) is minimized and the latency remains below the given threshold. In large networks, a systematic search like this for the global minimum becomes impractical [137]. The worst case time complexity of this algorithm can be assessed as follows: the search for each next lowest cost cloud requires approximately *n* lookups, searching *m* clouds would have the complexity O (mn). Again in the worst case, we would need to look through all the remaining (n-m) clouds to make sure the latency is below the threshold. Thus the complexity is O($(n\text{-}m).mn$) or O($n^2m - nm^2$). Selecting just five clouds out of a *hundred* would require 47,500 iterations. In Section 5.10.11 we compare the randomized cloud search with a modified sequential baseline method to show the usefulness of the adopted technique.

We find that the application of the general theory of optimization by random search gives us good results in the multi-cloud environment. The mathematical treatment of this technique is given in [138]. We have adapted this model to multimodal cases in the presence of constraints [137]. This category of algorithms is useful and efficient for large-scale ill-structured global optimization problems. In contrast with the deterministic methods, like branch and bound, which guarantee asymptotic convergence to the optimum at the high computational effort, random search algorithms find a relatively good solution quickly and easily. It has been shown that a global optimum can be found with random optimization even if the objective function is multi-modal [139]. Deterministic methods for global optimization are NP-hard, a random search method may be executed in polynomial time [140]. Many of the global random search (GRS) algorithms have the following desirable features because of which they are popular (i) the algorithms are usually easy to construct with guarantee of convergence, even if the objective function is multi-modal [140]; ii) they are insensitive to noise in the objective function; iii) they are insensitive to the shape of the feasible reason; (iv) they are insensitive to the growth in the dimensionality of the feature set. In these cases, it is relatively easier to construct GRS algorithms guaranteeing theoretical convergence. The theoretical basis of general random search is given below. The implementation is shown in Algorithm 5.3, and the convergence is proven empirically in Section 5.10.11.

According to [141], the general problem of minimization can be stated in terms of minimization of the objective function f(x) in the feasible region $x \in \mathbf{X}$, if x* is the global minimizer of f(x) or $f(x^*) = \min_{x \in \mathbf{X}} f(x)$. A global minimization algorithm constructs a set of points x_i i=1…n, from the region X, such that the sequence of labels $y_{i=1\ldots n} = \min_{i=1\ldots n} f(x_i)$ approaches the minimum f(x*) as n increases.

To establish the convergence of a global random search, we assume that if x is randomly chosen from within the region X, then f(x*) is a result of some stochastic process. We are presuming a generalized construction of the algorithm where the next point can be chosen from the entire space. Thus, if $X \subseteq R^d$ and $0<X<\infty$, $\sum_{j=1\ldots\infty} \inf P_j(B(x, \varepsilon)) = \infty$ for all all $x \in \mathbf{X}$ and $\varepsilon > 0$, where $B(x, \varepsilon) = \{y \in X : \|y-x\|_2 \leq \varepsilon\}$ and the infimum is over all possible previous points $x_{1\ldots(j-1)}$ and the result of the evaluation of the objective function at these points. P_j are the probability distribution of x_j. Then with probability one, the sequence of points x_1, x_2, ... falls infinitely often into any fixed neighborhood of any global minimizer. In other words, if the algorithm is allowed to converge to a global optimum in a finite number of iterations within an acceptance probability, then it will converge with probability one [141] [142]. The authors in [138] prove that as long as random sampling does not ignore any region, then the algorithm converges with probability one.

As even for large chains, the number of clouds from which resources are to be taken is not very large; we apply random selection to our problem by selecting at each step a unique set of the desired number of clouds randomly. Accordingly, we repeatedly choose, with replacement, a set M of m clouds from a space N of n clouds (such that $m \leq n$) with replacement. If the total cost of the last set is less than the set examined in the last iteration, and the latency is still less than the prescribed threshold, then the algorithm remembers this set. The cost includes that of cloud resources and inter-cloud links. The link costs are usually much larger and ensure locality of clouds while selecting clouds for placement. When the random selection no longer changes the achieved least cost, the process terminates, and the resulting least cost cloud-set is used for placement of the SFC in Algorithm 5.1. Alternatively, to ensure graceful stop, if the difference between the last two costs falls below a given value, the process can be terminated.

It is appropriate to mention that the total cost and latency of the selected cloud-set places an upper bound on the final figures as eventually more than one VNF may be placed on the same cloud, and all the clouds in the selected set may not be used. As the algorithm iterates over the available clouds, the set M clusters around the minimum. The algorithm converges to the global minimum, with probability one, even in a multimodal case, as long as it does not consistently ignore any of the clouds in the space N. These conditions are met in our implementation. Algorithm 5.3 gives the details of random selection. The procedure PREDICT_LATENCY has not been separately elaborated as it is based on the SVR model(s) refined for concept drift and short-term changes in traffic as already discussed above.

Algorithm 5.3: **RANDOM_SELECTION** (***C***, L_{th}, *cv_model*, *r_clouds*)

```
1: //C: a set of available clouds, cv_model: trained model
2: init small        //contains the sum of costs of the current smallest     cost clouds
3: init lat                     // lat: latency
4: init iter                    //set iterations large enough for convergence
5: while (iter)
6: init r_clouds    // r-cloud array holds final min cost set of clouds
7: //find a set of m unique clouds
8: while (m_clouds not unique)
9:    m_clouds ←a random set of m clouds from set C
10: end while
11: //test set r_clouds still has the lowest cost and lat ≤ threshold
12: call PREDICT_LATENCY     //uses trained and refined models
13: for k = 1, m
14:  $lat = lat + lat_k$                    //initial assessment of total latency
15:  $cost = cost + cost_k$
16: end for
17: if $cost < small$ and $lat \leq L_{th}$
18:  small = cost
19:  r_clouds ← m_clouds
20: end if
21: end while
```

Algorithm 5.3 expects CSP data like the available clouds *C* and a trained prediction model *cv_model* and produces a set of 'm' minimum cost clouds to be used for placement by Algorithm 5.1. The variable *small* represents the smallest total cost of the selected clouds. In line 8-10 a set of *m* unique clouds is selected. Line 12 calls the procedure that predicts latencies for the selected set of clouds. The total cost of the selected clouds is checked against the current minimum cost, and if found to be lower then the vector *r_clouds* is updated with the new set of clouds and *small* with the new lower cost.

5.8 Increasing Speed and Acceptance Rate of Placement

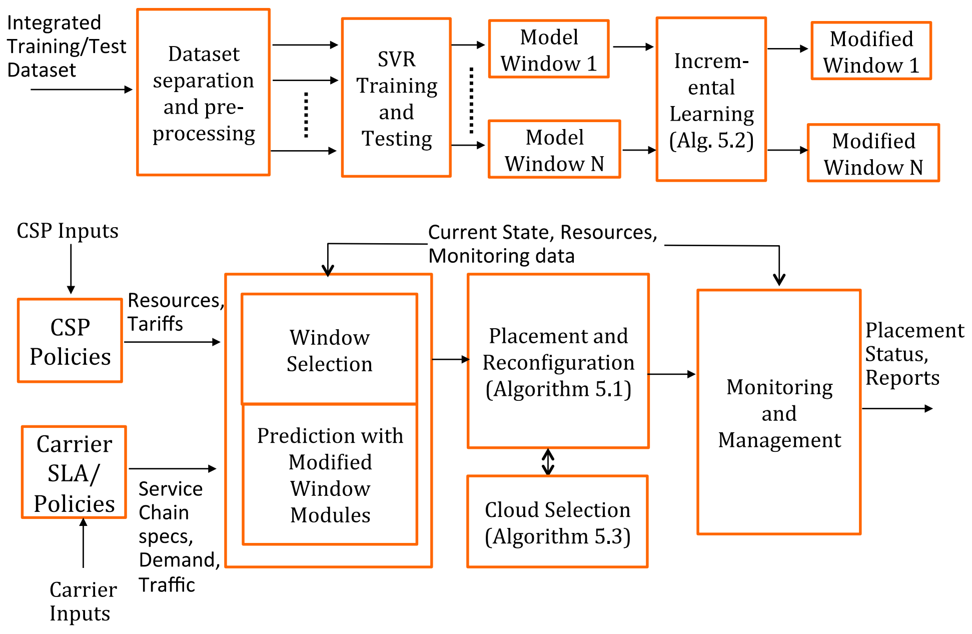
These requirements arise from the dual necessity of real-time usage and agility of the service deployment.

a. *Speed for real-time usage* In an operational virtual network service, the cloud service provider needs to monitor latency continuously for avoiding a breach of SLA requirements. Not only the latency and other QoS requirements should be met on initial placement, but also during operation of the service. If the end-to-end latency goes over the stipulated threshold, then the change of placement of VNFs and reconfiguration of the SFC is required. This necessitates the algorithm to be fast in giving optimum SFC placement, migration, and scaling (increasing or reducing the number of instances) decisions so that the network can be dynamically managed. As reported in the literature, ILP based solutions for the placement problem may take a long time (of the order of hours) to converge to the optimum solution [59] making them unsuitable in many situations of dynamic placement.

b. *Efficiency of placement* The efficiency of placement refers to successful placement rate (also called the acceptance rate) and reconfiguration of chains consistent with SLA requirements. It is important for this rate to be high since frequent failure to place and reconfigure chains according to the requirement may lead to the carrier not being able to handle customer requests.

5.9 Combining the Elements of the Framework

The placement strategy described above has been implemented in a placement framework called the P-ART framework. The main modules of P-ART are as shown in Figure 5.7 along with the relationship with the algorithms discussed.



Integrated Training/Test Dataset
Dataset separation and pre-processing
SVR Training and Testing
Model Window 1
Model Window N
Incremental Learning (Alg. 5.2)
Modified Window 1
Modified Window N
CSP Inputs
CSP Policies
Resources, Tariffs
Current State, Resources, Monitoring data
Window Selection
Prediction with Modified Window Modules
Placement and Reconfiguration (Algorithm 5.1)
Cloud Selection (Algorithm 5.3)
Monitoring and Management
Placement Status, Reports
Carrier SLA/ Policies
Service Chain specs, Demand, Traffic
Carrier Inputs

Figure 5.7 The P-ART placement framework

The framework allows CSP and carrier policies to be stored as well as the means for them to communicate with the framework. The instant state of a cloud consists of the used capacities of virtual compute, storage and networking resources. For each placement request, the management and monitoring module produces a success or a failure report. A brief description of the modules is as follows:

Training and Windowing: This part takes the integrated dataset and breaks it into a separate dataset for each of the specified windows. It then trains one model for each window applying the FTVP methodology discussed above. Short-term changes are incorporated through incremental training.

The prediction module uses these predictions to given an assessment of the latencies at the time of placement. Currently SVM algorithm has been implemented for each window.

CSP Policies: Through this module, the cloud service provider (or a multi-cloud broker) enters the cloud configuration data, installed and used cloud capacities, installed and used link capacities as well as tariffs for resources.

Carrier Policies: This module accepts client's requests for changes in service chain placements, types of virtual functions and inter-function traffic rates. Operative parts of the tenants' SLAs, including latency, threshold, and cost budgets are also stored. Carrier privileges are also recorded in the database.

Prediction module: The prediction module uses the correct model for prediction of latencies at the time of activation of the chain. It predicts the latencies among clouds at the time an SFC would be actually placed and activated.

Placement and Reconfiguration Module: This module carries out placement, scaling, and adaptation to the changed state of the environment. Heuristics for placement has been devised to work fast and converge to a set of clouds close to the minimum cost and latency below the threshold. If a placement is successful, it gives the end-to-end latency and cost.

Monitoring and Management Module: This module keeps an inventory of the resources used, the status of performance parameters and the state of the cloud environment. If placement is successful, it gives the end-to-end latency and the cost. Online monitoring reports are part of the future extension.

5.10 Evaluation of the Framework

We evaluated the P-ART framework to confirm the validity of all the sub-systems incorporated, viz., model training and generalization, prediction and its refinement, cloud selection for placement, speed and acceptance ratio of placement. To keep evaluation close to reality and to cross-verify results, datasets used for training and testing were generated as elaborated in Section 5.10.3.

5.10.1 The Experimental Set-up for Evaluation

In our experiments, we use multiple instances of the VNS having one SFC with 5 VNFs introduced in Section 5.1 (Figure 5.3). As we shall see in Section 5.10.11, the method scales well for bigger chains with thousands of virtual functions. The traffic entering the aggregation switch (VNF1) is divided into two streams, one going to one of the Provider Edge (PE)-routers (VNF2 or VNF3) depending on the carrier's traffic routing policies. For instance, the policy may route traffic from different geographical areas through different paths. All the traffic passes through one of the instances of BNG (VNF4) where in practice, the flow accounting will take place for billing purposes. The traffic is then routed to P-Router on route to the destination. The end-to-end latency of the chain would be the greater of the latencies of the two paths VNF1-VNF2-VNF4-VNF5 and VNF1-VNF3-VNF4-VNF5.

In the experiments that we have discussed in this dissertation, the CSP domain consists of 10 clouds. However, we also tested the random selection algorithm for a larger number of clouds, and the results have been discussed in Section 5.10.8. Without the loss of generality, we generate the link capacities randomly from the chosen set of realistic capacities. In our experiments, we choose from the set L= [0.016, 0.064, 0.100, 0.155, 0.622, 2.5] (in Gbps). All links are presumed to be bi-directional.

The compute capacities of the VMs hosting VNFs have been taken as a single consolidated figure for processor, memory, and storage. An example of such a usage is Amazon EC2 where, for instance, t_2, the medium virtual machine provides two virtual CPUs, 4 GB storage and elastic storage. In our experiments, the categories defined are as shown in Table 5.2.

Table 5.2 Categorization of server resources

Integrated capacity	**vCPUs**	**Memory**	**Storage**
1	1	1GB	Flexible
2	2	2GB	Flexible
4	4	4GB	Flexible
6	4	8GB	Flexible
8	8	8GB	Flexible
10	8	16GB	Flexible

5.10.2 Selection of Features for Training the Prediction Models

Considering the importance of the selection of predictor variables, due attention was given to this aspect. Too many features can make prediction models too complex, increase the training time and make test errors worse. Further, selecting a good set of features, out of all the features generated, improves the accuracy of prediction and speed of processing. Cross-validation error has been used to guide feature selection for our prediction models in SVR. Features that do not give an improvement in terms of lower overall errors (indicating better prediction) were removed from the initial feature set. We finalized the set of features given in Table 5.3. Further analysis, to include other variables that are not highly correlated with the existing ones, but may reduce the cross-validation error, is left as future work.

As seen in Table 5.3, the feature space is represented by $\boldsymbol{X} = [x_1, x_2, x_3, x_4, x_6, x_7, x_8]^T$ and corresponding labels $\boldsymbol{y}$. The equipped physical compute, and storage capacities of a server govern the number of VMs that can be created on it and correspondingly the number VNFs that can be hosted. VMs on the same PM may cause interference in each other’s operation because of shared resources which may lead to delays. As far as the links are concerned, each additional Gbps of equipped capacity does not give the same increase in traffic carrying capacity. The amount of traffic that can actually be carried depends on the grade of service required. Total ingress traffic depends on the number of served subscriber clusters. The end-to-end latency depends on the traffic, requiring this feature to be included. We have seen in Section 5.6 that traffic is dependent on the time of the day. We discussed the number of windows and its relationship with the complexity of the model. The increasing window number is indicative of the increasing time of the day. While the number of windows is a parameter in the evaluation, we obtained good compensation of concept drift with four windows as indicated by the results.

Table 5.3 Predictor variables and the output label

Predictor variables		**Label (output)**
x_1	Origin cloud compute installed capacity	y: Latency (ms)
x_2	Destination cloud compute installed capacity	
x_3	Link installed capacity (Gbps)	
x_4	Link used capacity (Gbps)	
x_5	Origin cloud compute capacity used	
x_6	Destination cloud compute used capacity	
x_7	Window #	
x_8	The distance between the origin and destination clouds	

5.10.3 Obtaining Training Datasets

We were cognizant of the fact that if a model has been trained with the adequate, realistic dataset, it will generalize well in the production environment. For a more thorough evaluation of the model, we use two methods for generating datasets. One dataset was obtained through *simulation* using a queuing-theoretic model–of inter-VNF traffic flows and the other through actual *implementation* of the service chain on CloudLab. The details of these are given in the next two sub-sections.

5.10.4 Inter-VNF Traffic Flow Simulation

Carrier networks carry all kinds of traffic: voice, data, and video. Some of these applications are real-time, and their packets have higher priorities. When queues build up at link or router buffers, the higher priority traffic may pre-empt lower priority traffic. It follows that different types of traffic will experience different delays. The delay model shown in Figure 5.8 takes care of all the important delays. Queuing delay in the links is the variable part of the end-to-end delay and depends on the network load. Propagation delay is the time required by the signal to travel on the link from one VNF to another. This delay depends on the media and is proportional to the length of the link, approximated by the distance between clouds. The other prominent delays are processing delay in the clouds, queueing delay in the virtual machines, and transmission queueing delays on the link. Intercloud simulation was carried out covering all significant delays.

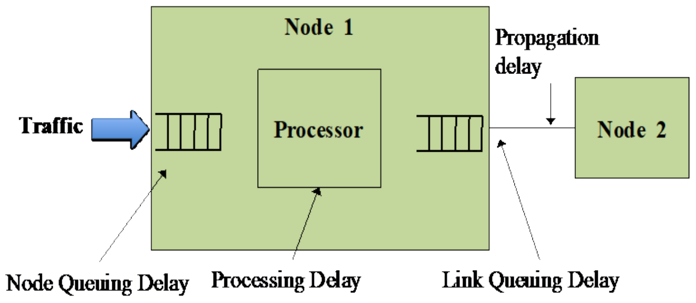


Figure 5.8 Traffic Delay Model for Data Generation

The total time spent by voice and data packets in the network can follow any distribution. Following the conclusion in [143] [144], we have assumed an M/G/1 queueing system of infinite capacity with non-preemptive priority. The traffic load is varied to imitate the pattern of the actual traffic. A C++ routine generates the dataset that incorporates all the parameters described above. The dataset was normalized to keep the numbers comparable. This will prevent any feature from overpowering others in the model and avoid biases.

5.10.5 CloudLab Implementation

CloudLab is a “meta-cloud” that has been implemented by the University of Utah, Clemson University, the University of Wisconsin, Madison, the University of Massachusetts Amherst, Raytheon BBN Technologies, and the US Ignite for researchers to build their own clouds for experimentation [144]. The software stack that manages CloudLab is based on Emulab. The infrastructure at Utah, Wisconsin and South Carolina is interconnected with nationwide and international infrastructure from Internet2, so it has been possible to extend, software-defined networks right to every host. The CloudLab set up created for this study is shown in Figure 5.9.

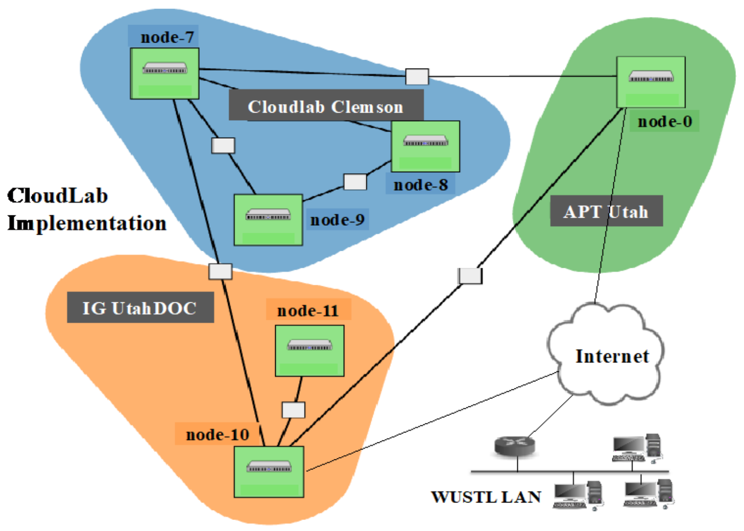
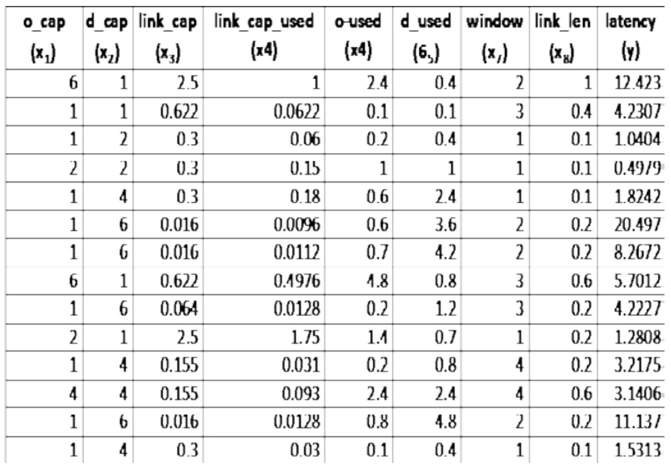


Figure 5.9. The CloudLab Implementation

The data collection process involves traffic being routed from a host on the WUSTL (Washington University in St. Louis) LAN through the Internet to the CloudLab nodes. Thus the test traffic goes with the live traffic on the Internet and provides real-life traffic conditions. Nodes 0, 7 and 10 are the transit points for traffic at APT Utah, Clemson University and IG Utah DDC (InstaGENI Rack in Downtown Data Center) clouds, respectively. The distance from the host at Washington University in St Louis to each of these were IG Utah DDC (800 miles), Clemson University (1950 miles) and APT Utah (800 miles). The VNFs are presumed to be hosted as follows: VNF1 on node11, VNF2, and VNF3 on Node 10, VNF4 on Node 7 and VNF5 on Node 9. Delays on the link from WUSTL to the CloudLab depended on the traffic on the Internet. Within CloudLab, the delays were varied by loading the links with different amounts of traffic. Various delays were recorded as part of the training data. A snapshot of part of one of the training sets is shown in Table 5.4.

Table 5.4 An extract of the integrated training dataset



o_cap (x_1)	d_cap (x_2)	link_cap (x_3)	link_cap_used (x4)	o-used (x4)	d_used (6_5)	window (x_7)	link_len (x_8)	latency (y)
6	1	2.5	1	2.4	0.4	2	1	12.423
1	1	0.622	0.0622	0.1	0.1	3	0.4	4.2307
1	2	0.3	0.06	0.2	0.4	1	0.1	1.0404
2	2	0.3	0.15	1	1	1	0.1	0.4979
1	4	0.3	0.18	0.6	2.4	1	0.1	1.8242
1	6	0.016	0.0096	0.6	3.6	2	0.2	20.497
1	6	0.016	0.0112	0.7	4.2	2	0.2	8.2672
6	1	0.622	0.4976	4.8	0.8	3	0.6	5.7012
1	6	0.064	0.0128	0.2	1.2	3	0.2	4.2227
2	1	2.5	1.75	1.4	0.7	1	0.2	1.2808
1	4	0.155	0.031	0.2	0.8	4	0.2	3.2175
4	4	0.155	0.093	2.4	2.4	4	0.6	3.1406
1	6	0.016	0.0128	0.8	4.8	2	0.2	11.137
1	4	0.3	0.03	0.1	0.4	1	0.1	1.5313

Legend

o_cap	Origin cloud installed capacity
d_cap	Destination cloud installed capacity
link_cap	Capacity of the link
link_cap_used	Used up link capacity
o_used	Origin cloud used capacity
d_used	Destination cloud used capacity
window	The time window to which traffic data pertains
link_len	Length of the link between origin and destination clouds
latency	Latency observed

5.10.6 Selection of the Machine Learning Model

There are quite a few AI techniques, involving machine learning, that are potentially applicable to the problem of detection and localization of fault and performance anomalies. Models with a single layer of non-linearity, e.g., a neural network with one hidden layer, are referred to as shallow structures or shallow machine learning architectures and those with more than one layer of non-linearity as deep structures or deep learning architectures. Shallow models with linear hypothesis may have O(n) prediction time complexity and training time of O(l^2+n^3) where l denotes the size and n the degree of the dataset, but approximation errors are large for the high dimensional and large volume of data that are usually associated with FP problem. With non-linear hypothesis space and kernel trick, the approximation errors may be smaller at the cost of higher complexity of the training time which is O(l^3 + l^2n) and prediction speed of O(ln). Of the prevalent shallow machine learning architectures, Support Vector Machines (SVM) and Random Forest (RF) are considered useful for diagnostic applications [145]. Another supervisory technique, Bayesian Network (BN), has been applied to fault management in the industrial settings. We will discuss below the analysis that was carried out to finalize the model [147].

Size of Training Dataset: The size of the available training dataset governs the choice of the machine-learning algorithm. How much data is enough depends on the number of features and the non-linearity in the relationship of features and labels among others. If the dataset is small, one may choose high bias and low variance classifiers like Naïve Bayes as compared to the low bias and high variance classifies like kNN to avoid overfitting. When the training dataset size is large, low bias and high variance classifiers give a lower asymptotic error.

Number of Parameters: Most machine learning algorithms are associated with some parameters and hyperparameters. Parameters of an algorithm are internal to it and their values affect how the algorithm behaves. They are usually learned at the time of training of the model. The value chosen for these parameters may affect the accuracy with which the model predicts. Support vectors of the SVM algorithm are an example of a model parameter. Hyperparameters are normally external to the algorithm. They need careful tuning to get good accuracy from the model. An example is the C hyperparameter in SVM. Even though having many parameters or hyperparameters typically provides greater flexibility, training time and accuracy of the algorithm can sometimes be quite sensitive to getting just the right settings.

Number of Features: If the number of features is large then the dataset is said to be high dimensional. With high dimensional dataset, we need more data to train the model. Increase in size of the dataset affects different algorithms differently. The complexity of some machine learning algorithms may rise exponentially in such cases. The training time may become too long for the model to be used in real-time applications.

Learning Process: The learning process of a model may be supervised or unsupervised based on whether labels are available or not. Since the labels indicate the ground truth, we know how our trained model should behave. In unsupervised learning, the data is unlabeled, so the model learns the inherent structure in the data. If there is some labeled data and a lot of unlabeled data, then we may use semi-supervised learning in which the labeled data can be used to improve the accuracy of the model built using unlabeled data. Another thing to note is that we are predicting latency values which vary in a continuous range. This would, therefore, call for a regression model as against a classification model.

The requirements were studied carefully to pick the right algorithm for the application. In our case, it is important that the model works in real-time or near real-time. This is possible if the placements and reconfigurations are fast. The model should be fast to train and update with real-time information. This requires models to be generally simple, with controlled dimensionality and a manageable number of hyperparameters to tune. Additionally, some models may not be suitable for online training.

Keeping the above in view, we compared a few suitable stock methods to decide on the one that we would include in our model. The models were created and tested on Weka [147]. In each case, the models were tuned for good parameter values, and a 13-fold cross validation was used. We discuss the methods briefly followed by a comparison of their performance in Table 5.5.

Random Forest is a supervised method which is robust yet simple to use. It provides good results in many situations. It does not have many hyperparameters to tune, the useful ones being the number of trees and the maximum number of features to be tried in each tree. Despite their flexibility, random forest does not support online learning. Retraining by rebuilding the trees when new examples are introduced takes time. The maximum depth of each tree has been set at unlimited. The number of iterations or number of trees is set as 100.

Table 5.5 Comparative study of machine learning algorithms

	Corr. Coeff.	Mean Absolute Error	RMS Error	Relative absolute error (%)	Root relative squared error (%)
Random Forest	0.8639	1.1881	2.4219	33.6077	50.3668
SVR	0.8610	1.2426	2.5048	35.4465	52.8385
KNN	0.8007	1.469	2.9681	41.9043	61.7248
MLP	0.8015	1.9317	2.9405	55.103	61.1514
Gaussian	0.5714	2.7523	3.9340	78.5130	81.8128

Support Vector Machine (SVM) is a supervised learning algorithm. The regression version of SVM, which is designated SVR or Support Vector Machine for Regression (SMOReg), gives good accuracy and can work with high dimensional data, which is not linearly separable. Parameter values that obtained for good results are C=200, γ=0.01, $\in$= 10E-8, RBF Kernel.

K-Means is an unsupervised model and has been included for comparison here. In this, k data points are chosen, and data is divided into clusters with each example going with the nearest data-point. Then, centers of the clusters are converted, and the process repeats until convergence. The result depends on the initial choice of the points, and the global minimum is not guaranteed.

Multi-layer Perceptron (MLP) are neural networks with at least three layers of neurons – an input, a hidden and an output layer. These layers are connected in the form of a directed graph between the input and the output layers. It is also called a feed forward network. An MLP uses backpropagation as a supervised learning technique. Some of the parameters include N (the number of epochs for training) taken as 500, E (the number of consecutive increases of errors allowed for validation before terminating the training) fixed at the default of 20 and L (the learning rate) taken as 0.3.

Gaussian processes are a supervised learning technique and generalization of Gaussian probability distribution. Gaussian distributions are governed by stochastic processes and describe random variables. A Gaussian distribution is fully specified by its mean and covariance matrix. In a similar manner, a Gaussian process is specified by a mean and a covariance function. Some of the parameters are L (the level of Gaussian noise) taken at the default value of 1 and K (the Kernel to use) taken as PolyKernel.

Using the root mean square error as a good indication of the appropriateness of the algorithm for the datasets used we see that Random Forest gives the lower error followed by SVR. Taking into account our requirement of online updates, we chose to implement SVR.

5.10.7 Tuning and Testing of the Predictive Model

In the SVR models, three hyper-parameters, viz., $\in$, C, Υ need attention. Tuning these hyper-parameters is one of the main challenges in improving the predictive accuracy of an SVR model. The Υ parameter can be seen as the inverse of the radius of influence of samples selected by the model as support vectors. With a small Υ, the model cannot capture the complexity or "shape" of the data. If Υ is too large, the radius of the area of influence of the support vectors only includes the support vector itself, and no amount of regularization with *C* will be able to prevent overfitting. The constant *C* determines the tradeoff between the flatness of *f* and the amount of error allowed above ϵ. A low *C* makes the decision surface smooth; a high *C* aims at classifying all training examples correctly by giving the model freedom to select more samples as support vectors. Most researchers have followed a standard procedure in using a grid search [123] to determine the appropriate values. A number of runs narrowed down the parameters to C = 1×10^{-2} and $\Upsilon = 1$. The cross-validation error for this combination was the lowest at 7.84295×10^{3}. It is worth mentioning that with system decided settings when the built-in tuning feature is allowed to choose the parameters; the loss is higher at 2.21345×10^{4}. The grid search has, in this case, resulted in better hyper-parameter values.

The basic idea of using latency prediction is to improve the placement of virtual functions at a future time. This will only work if the predictive model produces good predictions of latency. With the Weka tool, SVR with RBF Kernel with the hyper-parameters set at C=10, $\in$=0.4 and 20% hold-out for cross-validation, we get the errors shown in Tables 5.6 and 5.7. It can be seen that both the training and test RMSEs are low indicating good performance. In the classical case, test errors would be slightly higher than the training errors. A lower test error may indicate overfitting or biases in the dataset. These can be overcome by curating the training dataset.

Table 5.6 Training Error

=== Evaluation on training set ===	
=== Summary ===	
Correlation coefficient	0.861
Mean absolute error	1.2426
Root mean square error	2.5408
Relative absolute error	35.4465%
Root relative squared error	52.8385%

Table 5.7 Test Error

=== Evaluation on training set ===	
=== Summary ===	
Correlation coefficient	0.7304
Mean absolute error	1.8895
Root mean squared error	2.5469
Relative absolute error	63.5334 %
Root relative squared error	71.5849 %

A comparative plot of training and test error ratios (defined as *prediction_error/acutal_latency)* is given in Figure 5.10. It can be seen that the model training errors are low and it generalizes well with the test data.

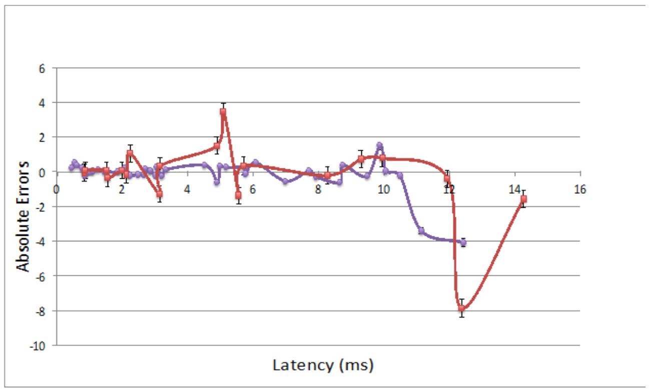
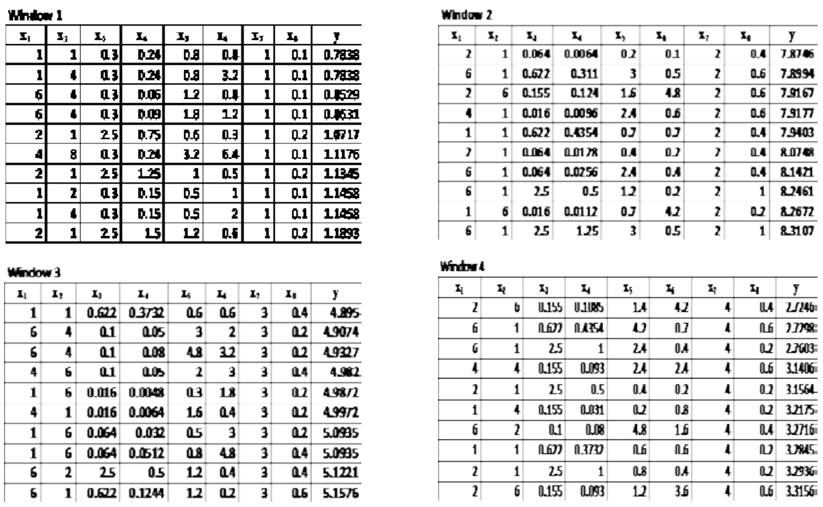


Figure 5.10 Training and test error ratios (with standard error bars)

5.10.8 Refinement of Latency Prediction by Compensating Concept Drift

The FTVP method for handling the concept drift in telecommunication traffic was presented in Section 5.6.2. This method brings in the sense of time in the datasets. Most researchers working with predictive model do not include time as a feature. In our experience, including time as a feature affects the predictions positively. We divided the data into windows of equal time blocks, which give variable data ranges. The window# is the feature (x_7) in the training dataset and has a direct relation with the time as increasing number relates to increasing time. All the time-related observations were divided into four windows. A sample from each of these is given in Figure 5.11.



Window 1

x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	y
1	1	0.3	0.24	0.8	0.8	1	0.1	0.7838
1	4	0.3	0.24	0.8	3.2	1	0.1	0.7838
6	4	0.3	0.06	1.2	0.8	1	0.1	0.8529
6	4	0.3	0.09	1.8	1.2	1	0.1	0.8531
2	1	2.5	0.75	0.6	0.3	1	0.2	1.0717
4	8	0.3	0.24	3.2	6.4	1	0.1	1.1176
2	1	2.5	1.25	1	0.5	1	0.2	1.1345
1	2	0.3	0.15	0.5	1	1	0.1	1.1458
1	4	0.3	0.15	0.5	2	1	0.1	1.1458
2	1	2.5	1.5	1.2	0.6	1	0.2	1.1893

Window 2

x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	y
2	1	0.064	0.0064	0.2	0.1	2	0.4	7.8746
6	1	0.622	0.311	3	0.5	2	0.6	7.8994
2	6	0.155	0.124	1.6	4.8	2	0.6	7.9167
4	1	0.016	0.0096	2.4	0.6	2	0.6	7.9177
1	1	0.622	0.4354	0.7	0.7	2	0.4	7.9403
2	1	0.064	0.0178	0.4	0.7	2	0.4	8.0748
6	1	0.064	0.0256	2.4	0.4	2	0.4	8.1421
6	1	2.5	0.5	1.2	0.2	2	1	8.2461
1	6	0.016	0.0112	0.7	4.2	2	0.2	8.2672
6	1	2.5	1.25	3	0.5	2	1	8.3107

Window 3

x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	y
1	1	0.622	0.3732	0.6	0.6	3	0.4	4.895
6	4	0.1	0.05	3	2	3	0.2	4.9074
6	4	0.1	0.08	4.8	3.2	3	0.2	4.9327
4	6	0.1	0.05	2	3	3	0.4	4.982
1	6	0.016	0.0048	0.3	1.8	3	0.2	4.9872
4	1	0.016	0.0064	1.6	0.4	3	0.2	4.9972
1	6	0.064	0.032	0.5	3	3	0.2	5.0935
1	6	0.064	0.0512	0.8	4.8	3	0.4	5.0935
6	2	2.5	0.5	1.2	0.4	3	0.4	5.1221
6	1	0.622	0.1244	1.2	0.2	3	0.6	5.1576

Window 4

x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	y
2	6	0.155	0.1085	1.4	4.2	4	0.4	2.7246
6	1	0.622	0.4354	4.2	0.7	4	0.6	2.7798
6	1	2.5	1	2.4	0.4	4	0.2	2.7603
4	4	0.155	0.093	2.4	2.4	4	0.6	3.1406
2	1	2.5	0.5	0.4	0.2	4	0.2	3.1564
1	4	0.155	0.031	0.2	0.8	4	0.2	3.2175
6	2	0.1	0.08	4.8	1.6	4	0.4	3.2716
1	1	0.622	0.3732	0.6	0.6	4	0.2	3.2845
2	1	2.5	1	0.8	0.4	4	0.2	3.2936
2	6	0.155	0.093	1.2	3.6	4	0.6	3.3156

Figure 5.11 Extract of FTVP windows

The data in different windows have different characteristics as shown by the mean and standard deviation in Table 5.8:

Table 5.8 Probability distribution parameters in different windows

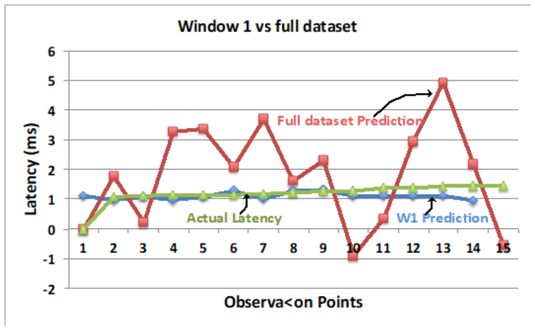
Window	1	2	3	4
Latency range	(1.824-0.422)	(27.683-7.452)	(7.317-4.131)	(4.216-1.869)
Mean	1.083	11.834	5.366	2.773
Standard deviation	0.425	4.848	0.797	0.588

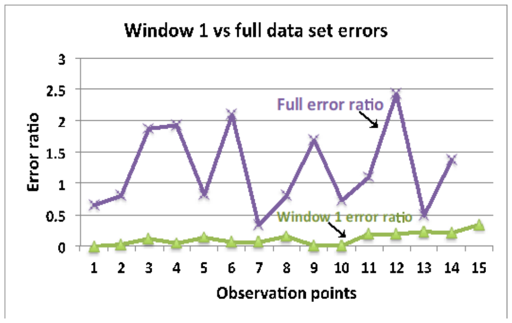
SVR with separate window models gives much better predictions on new data-points falling in those windows. Comparison of latency prediction and error ratios for each window and full dataset is given in Figure 5.12 (a) through (h).

Table 5.9 Errors with integrated and multiple models

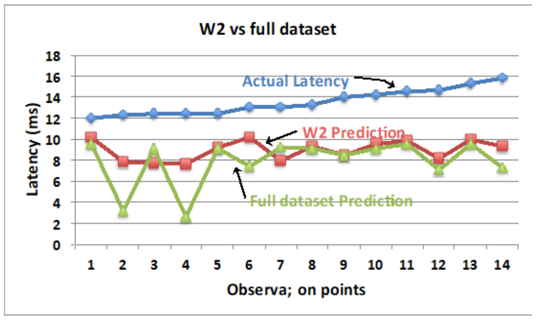
	Full dataset	Window 1	Window 2	Window 3	Window 4
Mean absolute error	3.2279	0.3698	0.4613	0.7342	2.5248
Root mean squared error	4.5869	0.4283	0.5515	0.9102	2.9353

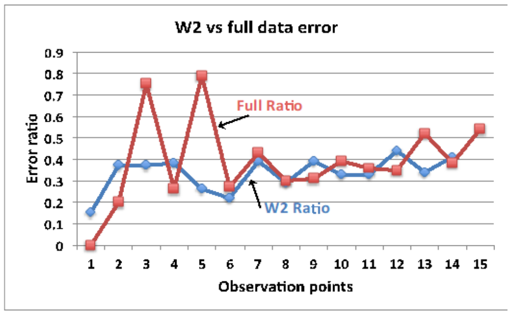
Table 5.9 summarizes the mean absolute errors and RMSE for the full (integrated) dataset and the window-based model. In the integrated model validation was done with 20% of the data points separated as a test set. For each window model also cross-validation was done with separate test sets. It can be seen that errors are less in a separate model for each widow compared to predictions made using integrated dataset.



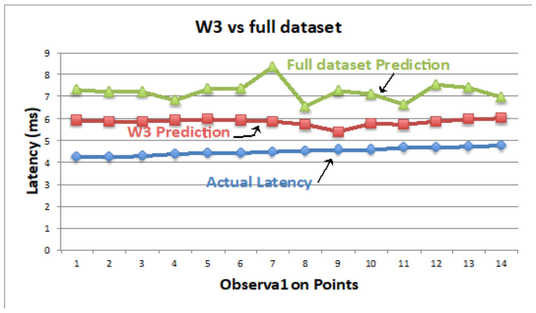


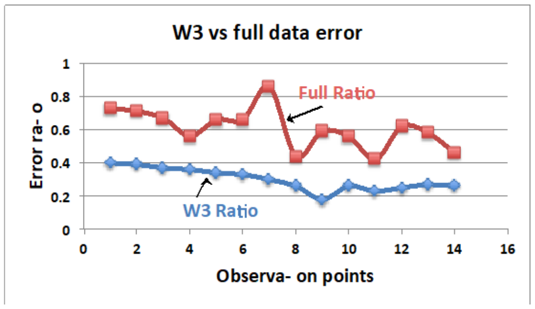
(a) Comparative Window 1 and full dataset performance (b) Error ratios for Window 1 and the full dataset
Window 1 RMSE =0.06, full model RMSE =0.47



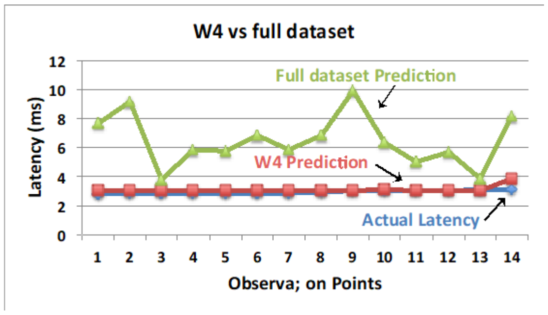


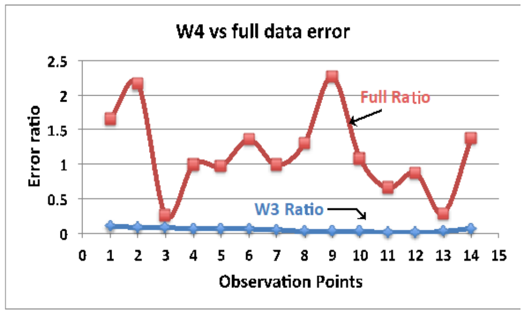
(c) Comparative Window 2 and full dataset performance (d) Error ratios for Window 2 and the full dataset
Window 2 RMSE=1.27, full model RMSE = 1.62





(e) Comparative Window 3 and full dataset performance (f) Error ratios for Window 3 and the full dataset
(e) Comparative Window 3 and full dataset performance





(e) Comparative Window 3 and full dataset performance (f) Error ratios for Window 3 and the full dataset
Window 3 RMSE=0.36 Full model RMSE = 0.74
Figure 5.12 Comparison of performance window-based integrated models

5.10.9 Compensating Shot Term Traffic Variations

We tested an incremental update of the trained models, with support vectors generated during VNS operation, while the trained model was in use. The result of initial training is given in Table 5.10, and after the introduction of separately generated support vectors, the results improved as shown in Table 5.11. We can see that both the mean absolute error and the RMSE decrease when new support vector points are learned online. Before the addition of new support vectors, the RMSE was 1.74; while after addition, it reduced to 1.68, which along with other measures of errors show an improved model.

Table 5.10 Performance before online updation

Support vectors before online updation				Performance before online updation	
SV#	Actual Latency	Predicted Latency	Error	=== Evaluation on test set ===	
50	5.713	5.379	-0.334	Correlation coefficient	0.8742
51	7.452	5.233	-2.219	Mean absolute error	1.2677
52	3.111	3.152	0.041	Root mean squared error	1.7366
53	1.531	2.785	1.254	Relative absolute error	47.3488
54	5.572	4.625	-0.947	Root relative squared error	49.2994
55	5.771	5.298	-0.473	Total Number of Instances	55

Table 5.11 Performance after online updation

Support vectors after online updation				Performance after online updation	
SV#	Actual Latency	Predicted Latency	Error	=== Evaluation on test set ===	
50	5.713	5.379	-0.334	Correlation coefficient	0.8816
51	7.452	5.233	-2.219	Mean absolute error	1.2014
52	3.111	3.152	0.041	Root mean squared error	1.6797
53	1.531	2.785	1.254	Relative absolute error	44.5651
54	5.572	4.625	-0.947	Root relative squared error	47.9109
55	5.771	5.298	-0.473	Total Number of Instances	60
56	3.111	3.374	0.264		
57	0.605	2.424	1.820		
58	3.345	3.190	-0.155		
59	3.315	3.579	0.064		
60	10.259	10.199	-0.060		

5.10.11 Cloud Optimization with Iterative Random Selection

The principle and methodology of random selection of clouds for placement of VNFs have been discussed in Section 5.7. In one trial, a total of 50 experiments were conducted with 1500 and

1700 iterations each. The minimum possible cost was 51 units, and latency threshold was set at 150 ms. In the former case, 98% of times the minimum cost of 51 units was reached (Figure 5.13 (a)) with a latency of 137 ms. In the 1700 iteration case, the minimum cost clouds were selected with the latency below the threshold in all cases (Figure 5.13 (b)).

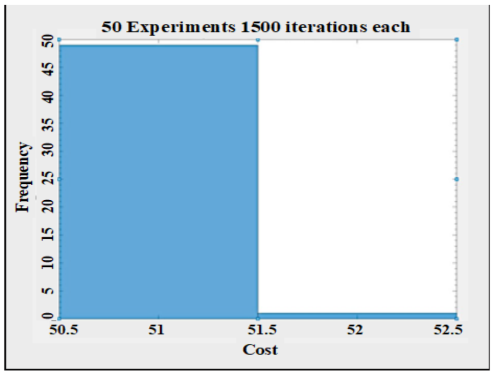


Figure 5.13. (a) 50 experiments with1500 iterations each

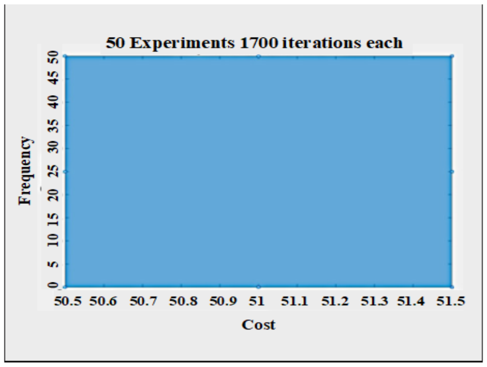


Figure 5.13. (b) 50 experiments with 1700 iterations each

In another trial of 5000 experiments, 50 each with the number of clouds increasing from 10 to 100 in steps of 10 and iterations from 500 to 2000, the convergence rate is as shown in Figure 5.14. Somewhere between 1500 and 2000 iterations, the algorithm converges to the minimum cost in 100% cases. This is an order of magnitude improvement over the exhaustive search described above.

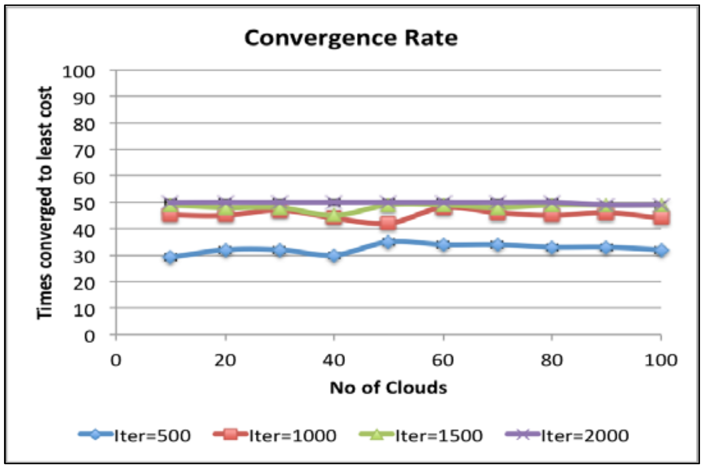


Figure 5.14. Number of convergences in 50 experiments

We implemented, as the baseline, a variation of the sequential method, which we call modified-sequential (M-sequential). In this method, the first set of lowest cost clouds were sequentially selected from a set of 100 clouds without replacement. This ensures the lowest cost. However, if the total latency of the selected cloud was more than the given latency threshold, then the highest latency cloud was removed from the selected set, and a search was made for the next lowest cost cloud. The search stopped when a set of lowest cost with latency below the given threshold was found.

Figure 5.15 shows the number of iterations required to achieve the target latencies (from 100 to 160ms) for both the randomized and M-sequential algorithms. We see that the M-Sequential takes from 34% to about 67% more iterations than randomized. Figure 5.16 gives the final latencies achieved in the number of iterations for which the algorithm was run (as shown in Figure 5.15). From these, we can conclude that the randomized algorithm performs better than the baseline both in terms of the number of iterations and latencies achieved in selecting the required set of clouds for placement.

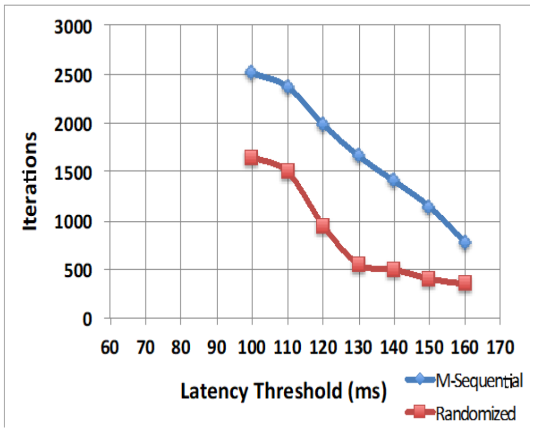


Figure 5.15. Number of iterations required by randomized and M-Sequential to achieve latency

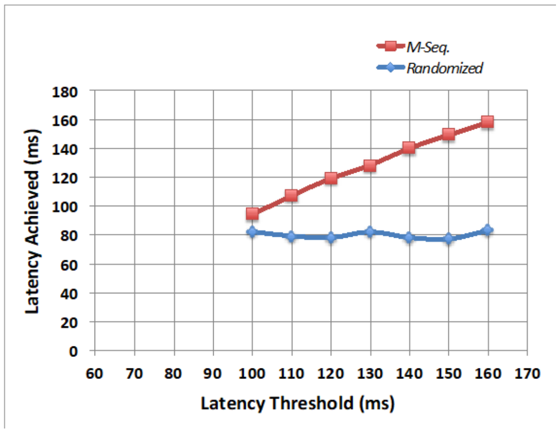


Figure 5.16. Latencies achieved by randomized and M-Sequential in the number of iterations

below the threshold shown in Figure 5.15

5.10.12 Speed and Efficiency

It is important for dynamic rescaling that the designed placement strategy is able to carry out a large number of placements within an acceptable time period. A slow placement algorithm would not be able to respond fast to the changing network situation or a tenant's new request. Changes made too late may not be suitable, and may actually be detrimental to the health of the network, as by that time the situation would have changed. On the other hand, if at a future time, maintaining the required performance does not need all the resources that have been deployed, not descaling would use up a higher amount of resources leading to higher expenses. For the training time of SVR, various assessments of complexity in the range $O(n^2)$ to $O(n^3)$ are available in the literature. According to [129] the complexity is $O(\max(n, d) \min(n, d)^2)$ where d is the size of the feature set. If n is much larger than d, then it can be approximated to $O(nd^2)$. However, the time complexity of the search is linear. It took about 1.19 s to train with 2720 sample points in Weka and 0.76 s in MATLAB. For speed of placement, we tested with 10 clusters, each requesting 10 to 100 SFCs of 5 VNFs each. Thus, the number of VNFs was varied from 500 to 5,000. We observe that the algorithm is able to place up to 3,000 VNFs in about 1 minute (Figure 5.17).

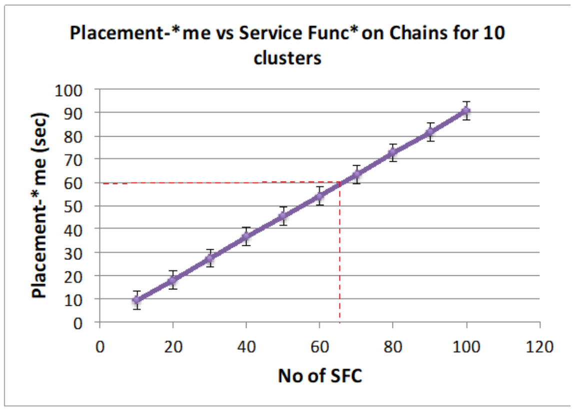


Figure 5.17 Placement time Vs. No of SFCs

To see how the speed of the proposed method compares with the placement speeds obtained in other works we examined the work done in [148]. The two methods have been performed under different conditions and are thus not strictly comparable. However, comparison does give a general idea of the behavior of the methods. From Figure 5.18, we see that in case of up to 20 SFCs, the ILP method is able to find a solution but the author reported average time is 8 minutes and 41 seconds and that of the suggested heuristic 1 minute and 21 seconds. For the case of 60 SFCs, the ILP model takes unduly longer times (>48 hours for ≥18 SFCs). The heuristic was able to give a solution in less than 30 minutes. For small instances, 40 SFC requests (with 75 network functions per request or a total of 3000 functions) take about 1000 seconds.

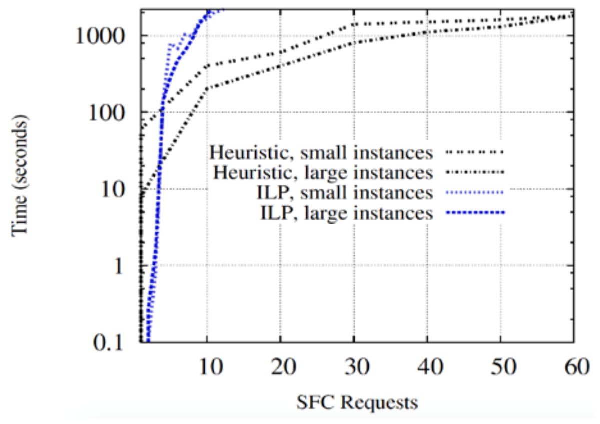


Figure 5.18 Placement time reported in [148]

A comparison has also been made with results obtained by a completely different technique presented in [149]. The authors have carried out joint optimization of resource allocation in NFV (JoraNFV). The authors assume that the number of VNFs can be 3, 4 or 5. Taking the example of a 5 VNF SFC and medium traffic, the authors conclude that their method works faster than CoordVNF described in [150] and a simulated annealing (SA) approach described in [151. The coordinated NFV-RA is formulated as mixed-integer linear programming (MILP) with a heuristic based two-stage approach to get the near optimal solution. For ten units of traffic, the number of instances deployed is about 7 for JoraNFV, 10.5 for CoordVNF and 7 for the SA method. For a 90 node network, the JoraNFV and CoorNFV take 10 seconds to place an SFC while SA takes about 2000 sec. Even if we assume a linear increase in time taken, for 3000 functions/instances JoraNFV will take 4285 seconds (Figure 5.19).

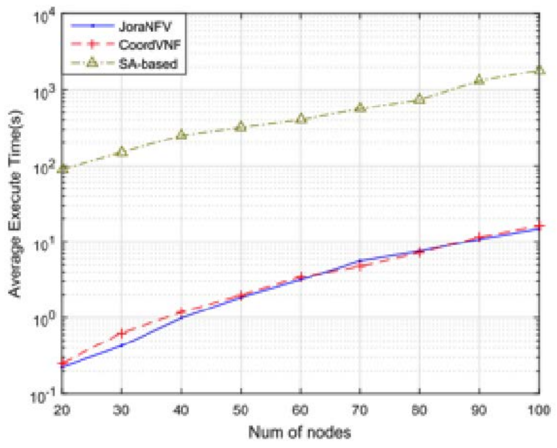


Figure 5.19. Average placement time reported in [149]

ILP based solutions for a large number of VNFs are slow, even with efficient solvers. Researchers in [152] and [127] have carried out VNF placement of different configurations using ILP method. In [127], the authors have reported that ILP takes 2.3, 4.0 and 7.2 hours for 10, 30 and 50 functions. In [152], the authors have tried to solve ILP for large networks (60 SFC with 4 VNFs and 30 instances, each, i.e., 7200 VNF instances) but for more than 18 SFCs the time taken is more than 48 hours. The authors have suggested heuristics to find an acceptable solution within reasonable time limits. Thus, [128] suggests using Genetic Algorithm with which 200-700 functions are placed in 8-13 seconds. In [152], the heuristics involve guiding the ILP solution by reducing the solution search space using binary search. With this for 7200 VNF instances, the time taken is 30 minutes. In [18], MILP based algorithm takes 500 seconds for 3,000 VNFs. We have shown above that with our framework we are able to place up to 3,000 VNF instances in less than 60 seconds. It needs to be appreciated that the results are not exactly comparable because of different experimental environments, but do give a sense of improvement with predictive algorithms.

The acceptance rate of the heuristic is an important parameter that often gets ignored. In the ongoing operations, whether we are looking at new placements or reconfiguration or migration of existing chains, it is important for the placement engine to be able to place SFCs every time a request is made subject to resources being available. If a large number of requests cannot be placed despite adequate capacities being available, then the acceptance rate is low, and we do not have a good algorithm. Failure to place SFCs would mean the loss of business for cloud service providers and may affect the requesting carriers revenue. For a medium-sized placement request, viz. 100 SFCs or 500 functions, the acceptance rate with our algorithm turns out to be 100% (Figure 5.20).

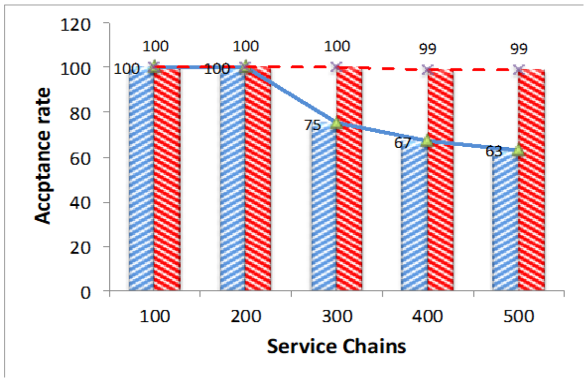


Figure 5.20 Acceptance rate Vs. Number of SFCs

As the number of service chains increases, the acceptance rate may fall because of a lack of capacity to place the complete service chains. When corrected for capacity, the acceptance rate for our algorithm remains above 98% up to the tested configuration of 500 SFCs or 2,500 VNFs.

We compare this with the real-time placement presented in [153]. The authors propose an ILP model to provide an optimal solution for placement and chaining VNFs based on minimizing the resources allocation and the deployment (mapping) delay while meeting the real-time condition. They also propose a heuristic solution named Degree Based Heuristic (DBH) to minimize the end-to-end delay and resources allocation cost. A comparison of successful requests is given in Figure 5.21.

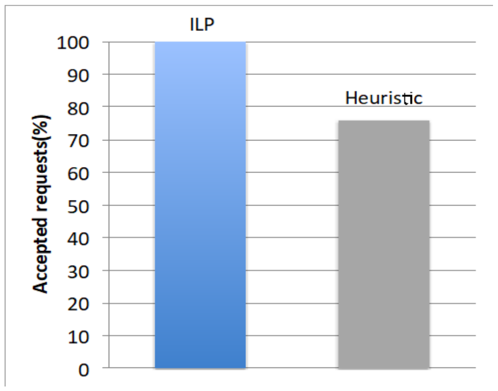


Figure 5.21. Acceptance % reported in [153]

The authors in [154] claim that with 500 VNFs the acceptance rate is 85%. In comparison, for our solution, the acceptance rate is 100% for up to 100 SFCs or 500 VNFs. Above this, the acceptance rate drops to 98% for up to 2500 VNFs.

5.11 Summary and Future Work

Innovative strategies are required to extract carrier-grade performance from SFCs that use resources from multiple clouds. Our strategy consists of techniques based on a predictive approach to performance optimization. Complex performance indicators, like end-to-end latency of a service chain at activation time, depend on far too many deterministic and probabilistic factors, to be modeled accurately by deterministic techniques. We have shown that a carefully designed predictive approach combined, with heuristics to select low-latency clouds, can help us in keeping the performance consistent with the SLA and costs within the carrier's budget. To make latency predictions more accurate, we have worked with time-based windows and an incremental update of the models used for prediction. Making use of the predicted latencies is an iteratively convergent randomized search heuristic used to select low latency clouds for successive placement of VNFs. Not only the proposed strategy produces results with low error, but it also executes fast so that the results can be used to take corrective actions. A comprehensive empirical evaluation has been carried out and reported in this work. The proposed P-ART framework has been built from all the techniques that have been described in this work.

A number of research directions are foreseen in this project. When enough resources are not available, carriers may accept under-dimensioned service chains. The service has to be functional, even though not meeting the performance criteria. Another important issue to be worked upon is the security aspect of VNSs in the multi-cloud environment.

Chapter 6

Fault and Performance Management

In this chapter we discuss the techniques that have been evolved to address the challenges that inhibit large-scale deployment of virtual network services using NFV (NFV) over multi-cloud systems. Our analysis shows that the gaps in inter-platform interface specifications and the lack of a reliable fault and performance management system as the key factors for the inability to achieve five nines availability that carriers cannot compromise on. We have developed a framework that takes into account the information available with the platforms and assists in reliable fault detection and localization and a means to increasing availability. Because of the lack of exhaustive identification of markers consisting of alarms, notifications, counters and measurements, we have defined some of the relevant one for fixed, mobile and broadband virtualized services, going beyond those that have been specified by the standards bodies like ETSI [25]. This helps in understanding the fault categories for designing detection and localization functions.

Taking a path different from that of the rule based systems in traditional networks, machine-learning solutions have been explored for detection of faults in the multi-cloud-NFV environments. The aim is to detect both manifested and impending faults and performance issues. For localization a deductive-predictive methodology has been developed. A major contribution of the research is use of innovative deep learning techniques to predict severity of impending faults through a combination of unsupervised and supervised training method. The manifesting faults, on the other hand, are handled with supervised techniques to correlated distributed alarms and notifications from various physical and virtual layers. This has been published in [156] [163] and also submitted to [157].

6.1 Introduction and Motivation

Using VNFs as the building blocks for creating VNSs, carriers can change the way the network services are provided. They are prepared to bear the problems of making this major change, so that they can reap the cost and agility benefits that have been described before [31]. Additionally, use of multi-clouds position the carriers at the advantage of competitive pricing, wider presence and a network that is more robust against failures [32] [158]. The virtual resources (e.g., virtual machines and virtual networking) for building these services can be obtained from the in-house datacenter, carrier-cloud owned by carriers themselves or public clouds owned by Cloud Service Providers (CSPs).

Despite many advantages, there are several challenges in providing large-scale deployments of NFV-based VNSs, which motivate this part of the work. It is important to identify and address these challenges so that they can be met and this promising technology does not disappear into oblivion. Some of the main challenges are listed here:

a) Performance and availability of VNSs are inferior to those of the traditional networks. The traditional networks have five nines availability (99.999%), which translates to 5 minutes and

15 seconds of downtime in a year. Cloud information technology applications have been working more on three nines (99.9%) availability, which go up to 8.76 hours of downtime in a year.

b) Traditional networks are built to the stringent quality of service (QoS) norms defined by Fault, Configuration, Accounting, Performance and Security (FCAPS) standards like ISO Common Management Information Protocol (CMIP) and ITU Telecommunications Management Network (TMN) M.3010 and M.3400 recommendations [28] [35] [36] [40]. Such norms are still to be fully defined and met for the VNS deployments.

c) In NFV-based VNSs, faults may occur for many more reasons compared to traditional physical networks. The cloud infrastructure consists of virtual resources such as virtual machines, virtual storage, and virtual network links. These virtual resources are created on shared physical resources like server hardware, system software or network links, using virtualization software (e.g., Hypervisors). Virtual resources may fail because of the failure of physical resources. Even if the physical resources are operational, the virtual resources may themselves fail [44] for reasons like operating system crash or hypervisor failure. Taking this argument a little further, even if both physical and virtual resources are healthy, the VNFs, like routers, instantiated on these virtual resources can develop algorithmic problems causing VNSs to malfunction or tot ally break down. The myriad levels of dysfunctions make handling of fault and performance (FP) issues in NFV over clouds more abstract and complex.

d) Internet Engineering Task Force (IETF) has recently identified ensuring performance and guaranteeing the QoS as open research areas and technology gaps in NFV [6]. Without a robust mechanism for handling these issues, carriers would find it very difficult to meet the quality, reliability and availability norms. This calls for vigorous research efforts so that NFV deployments acquire carrier-grade performance and availability [6] [159] [160]. Considering the importance of this area, ITU has included the fault management of the cloud-based NFV in their standardization agenda based on our paper [161]. The National Science Foundation (NSF) is supporting our research work in this important area, which can potentially change the way the telecommunication services are delivered [162].

6.2 Contributions

This part of our research deals with the complexities of fault and performance issues in an NFV multi-cloud environment as they prevent availability from reaching close to five nines. I then develop an innovative framework for detection and localization of these faults based machine and deep learning techniques. The specific contributions are as described below:

a) An elaborate discussion on the architecture, creation and management of VNSs from the viewpoint of fault and performance management. This is illustrated with a real-life example to elucidate clearly the fault and performance (FP) management problems and their complexities.

b) Discuss reasons for the traditional methods not performing well in the cloud-based NFV environments and how AI techniques like machine learning and deep learning can help.

c) Describe the AI based FP management framework that we have evolved to detect and localize faults.

d) Perform a thorough evaluation of the framework and discuss, in detail, the use of a hybrid shallow and a deep learning model to detect and localize some important aspects of fault and performance issues [163].

6.3 Structure of VNSs

In this section, we will describe the structure of a VNS, using NFV over a multi-cloud system, with the help of an example of a carrier network service. Additionally, we shall see the complexities and deficiencies in its management setup, which make a strong case for a predictive fault and performance management framework. A basic discussion on the constituents of VNS was done in Section 5.3. The discussion here is to help in the discussion of the fault and performance aspects of these services.

Figure 6.1 illustrates a carrier's implementation of the broadband Internet service consisting of access, aggregation core and the edge network. The *Access Network* consists of various

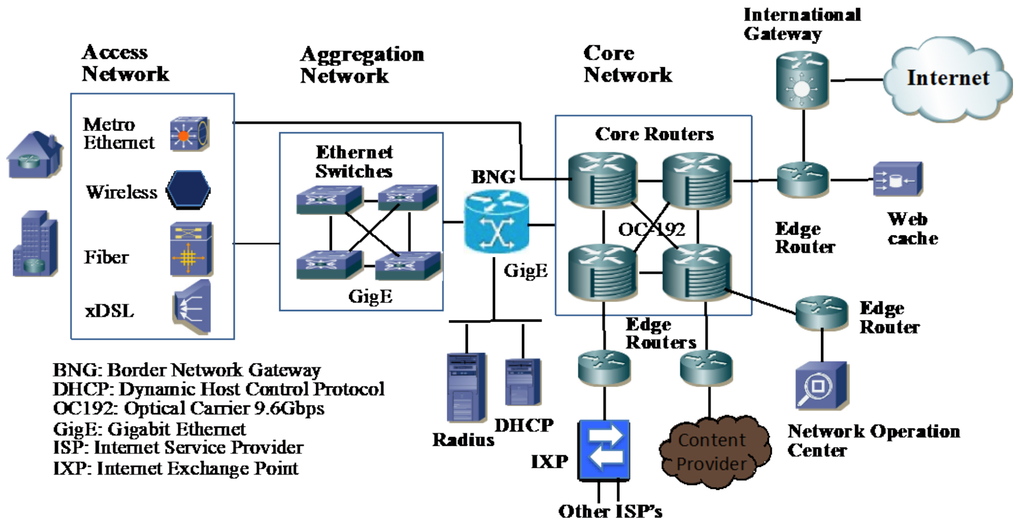


Figure 6.1 A carrier's broadband service network

technologies through which home and business customers access the Internet and the related services. The *Aggregation Network* collects various streams of traffic and concentrates them on higher capacity links to the core network. The Border Network Gateway (BNG) is situated at the border of the core and provides Layer-2 and Layer-3 connectivity, policy injection, QoS and accounting of user sessions and traffic flows. The *Core Network* contains core routers that transport traffic. The edge network consists primarily of the edge-routers which provider connectivity to the Internet and other services like content delivery. The core also connects to the Internet Exchange Points (IXPs) for exchanging traffic with other local ISPs without using the expensive international bandwidth.

6.3.1 A Top-down Discussion of VNS

A VNS can be defined in terms of layers of physical and virtual infrastructure. These layers are illustrated in Figure 6.2. We will see the relationship among these layers and discuss their functions in a top-down manner.

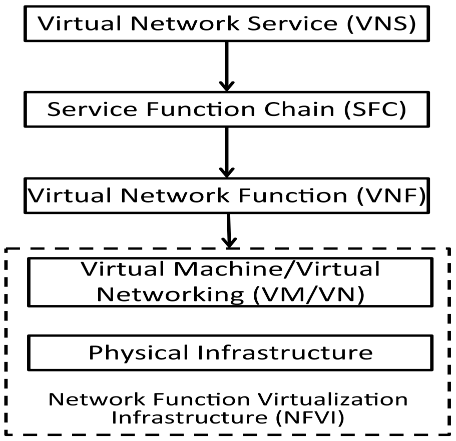

Virtual Network Service (VNS)
Service Function Chain (SFC)
Virtual Network Function (VNF)
Virtual Machine/Virtual Networking (VM/VN)
Physical Infrastructure
Network Function Virtualization Infrastructure (NFVI)

Figure 6.2. The virtualization hierarchy

a) Virtual Network Service (VNS)

From the illustration of a complex network service in Figure 6.1, we abstract a subset to represent a simpler VNS that we can use as an example. Figure 6.3 shows this VNS being composed of 8 VNFs realized as VNF1 to VNF8. The figure also shows that the carrier has retained DSLAMs as Physical Network Function (PNF) from their legacy network, as these functions might not have reached their end-of-life. VNFs of a service may belong to different vendors, owned by different operators, managed by different platforms and even unaware of each other. In such a case, the service is a multi-domain VNS [164] .

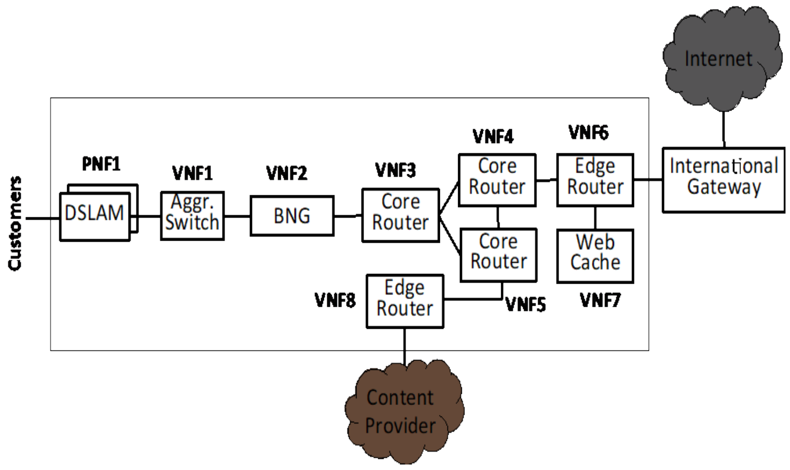


Figure 6.3 Virtual broadband service

b) Service Function Chains (SFCs)

A carrier may obtain resources from multiple cloud service providers for the reasons discussed before. VNFs are instantiated on these cloud resources and linked using virtual networking resources to form one or more SFCs [165] [166].

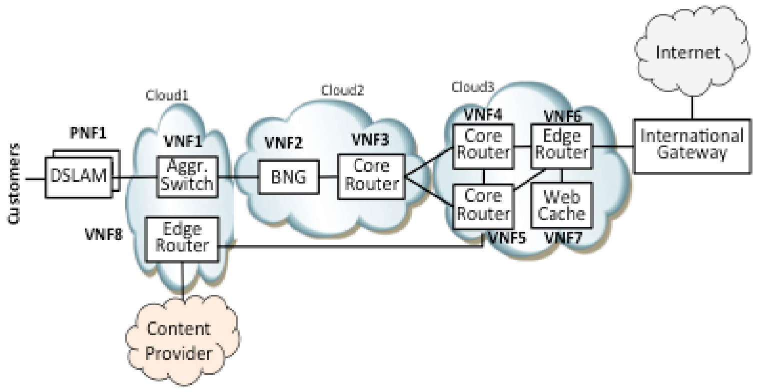


Figure 6.4 An SFC created on multiple clouds

Figure 6.4 shows an SFC with two paths, i.e., PNF1-VNF1-VNF2-VNF3-VNF4-VNF6 for the Internet access and PNF1-VNF1-VNF2-VNF3-VNF5-VNF8 for the content services. There are multiple paths through meshed core routers through which traffic can be routed if the selected link fails or if there is congestion on the selected link. Some VNFs are dimensioned with multiple instances to handle the volume of the expected traffic.

c) Virtual Network Functions (VNFs)

A VNF is the virtual counterpart of a network appliance or a middlebox implemented by running software over commercial off-the-shelf general purpose servers. Each VNF has a well-defined functional behavior and interfaces for interconnection with other VNFs or PNFs. Figure 6.5 shows a VNF with its Element Management System (EMS) and interfaces to the rest of the network [6] [29].

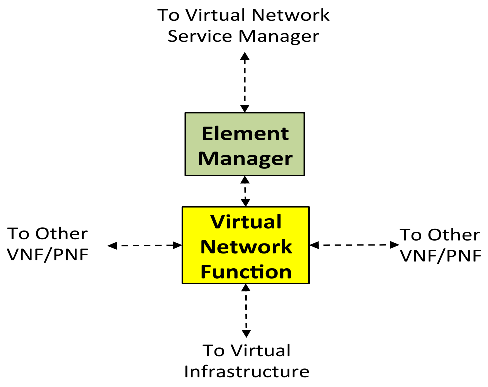


Figure 6.5. A VNF implementation

Some examples of pre-programmed VNFs are given in Table 6.1.

Table 6.1 Examples of commercially available VNFs

VNF	Function	OEM
ISRv	Integrated Services Router	Cisco Systems
vSphere	Distributed Switch	VMware
SRX	Firewall	Juniper Networks
440Vx	Load Balancer	Barracuda
SBC SWe	Session Border Controller	Ribbon Communications
Nexus 1000V	Virtual Switch	Cisco Systems
Steelhead CX 555V	WAN Accelerator	Riverbed Technology

d) Virtual Machines (VMs) and Network Function Virtualization Infrastructure (NFVI)

Network Function Virtualization Infrastructure (NFVI) consists of all the hardware and software used to deploy VNFs. This is also referred to as NFVI Point of Presence (NFVI-PoP). The virtual compute and storage resources, in an NFVI-PoP, are interconnected to form a network of virtual resources that can host carrier services. Figure 6.6 shows an example of a system with three

VNFs, their EMSs, hosted on two NFVIs of two cloud service providers. External connectivity may be possible through the designated switching and routing devices [165].

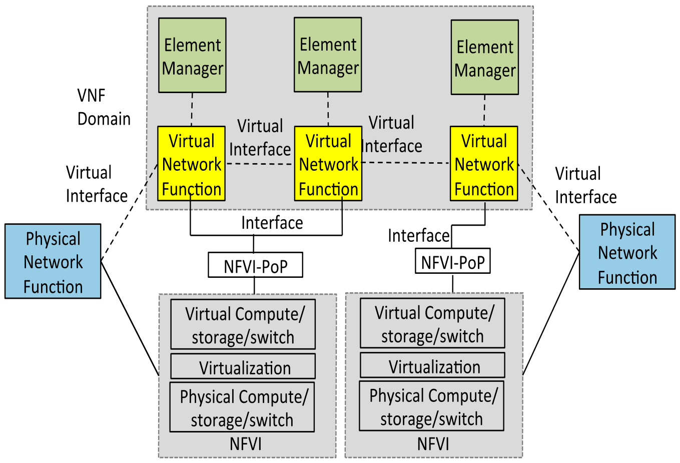


Figure 6.6. Network function virtualization infrastructure

e) Routing Protocols for SFC

Protocols for routing of traffic through an SFC are under development at the standards organizations. For example, two protocols - Segment Routing (SR) and the Network Service Header (NSH) - are under development in the IETF [167]. SR is a modified version of source routing. In SR the IPv6 header is extended to include the Segment Routing Header (SRH), which decides the path of the traffic packets. A segment is a path through a carrier network. The internals of the segment may not be exposed to the users. For example, it may be a Multi-Protocol Label Switching (MPLS) tunnel or may be a sequence of IP routers. Each segment has an ID and may contain information about the treatment of the traffic on that segment. A Software Defined Networking (SDN) controller may utilize the Path Computation Element Protocol (PCEP) to find the most appropriate segments and instruct the classifier to direct the traffic flow accordingly. NSH, on the other hand, can work with IPv4, IPv6, and Ethernet. NSH is an 8-byte header followed by a number of optional variable length context headers containing some metadata to be used by NSH-aware devices. Implementation of service function chains with NSH capabilities requires NSH-aware virtual switches and a central controller.

6.4 Management of VNSs from an FP Management Perspective

Figure 6.7 shows the management set-up of a VNS. Management platforms belonging to carrier, cloud service provider and the NFV operator interact to make VNSs work. NFV Management and Orchestration (NFV-MANO) and its subsystems use the virtual infrastructure provided by the Multi-cloud Management and Control Platform (MMCP) of the CSP to create and manage VNFs, SFCs, and VNSs [168]. MANO has the responsibility of performance measurement, event reporting, correlation and assistance in fault management of the VNSs and their constituents. The MMCP creates virtual machines, virtual storage, and virtual networking links. It also manages the placement and migration of these virtual resources over the available clouds [29]. The Operation Support System (OSS) of the carrier, with its Network Management System (NMS), manages the deployment and operation of the VNSs. The OSS carries out the network management by providing support for the provisioning of services, management of fault and performance and maintaining an inventory of the resources used.

In view of what has been said above, the fault management function becomes a shared responsibility. The relative distribution of responsibilities among various platforms and their interactions are yet to be fully defined. In order to understand the fault and performance management of VNSs, we need to discuss the sub-systems of MANO and their interactions in some more detail [169].

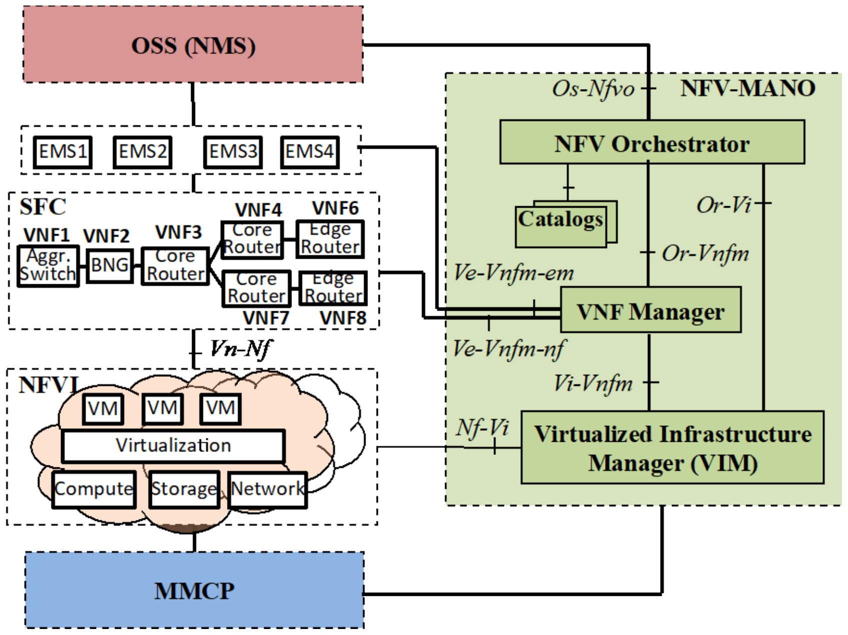


Figure 6.7. Orchestration and management of VNS

6.4.1 Virtual Infrastructure Manager (VIM)

VIM manages all the virtual and physical resources in NFVI to enable the higher layers of MANO to do their tasks of creating VNFs and SFCs. VIM manages the lifecycle of all the virtual resources in one NFVI domain (one infrastructure provider's domain) and applies security policies on them. VIM collects information about the performance events and measurements from NFVI over the Nf-Vi reference point and forwards them to NFV Orchestrator (NFVO) through its northbound reference point (Or-Vi). In the cloud environment, VIM interacts with the cloud management platform for obtaining virtual resources. In a multi-cloud or a multi-carrier service, there may be multiple VIMs managing the resources.

6.4.2 Virtual Network Function Manager (VNFM)

The VNFM instantiates and configures VNFs with resources obtained through the VIM. During the lifetime of a network service, VNFM manages the complete lifecycle of the VNFs (scaling, descaling and eventually terminating when they are no longer required). It is entrusted with the important functions of FP management of VNFs. For this, VNFM interacts with the EMSs of the VNFs to obtain fault and performance markers. The EMS (not a part of the MANO) collects device statistics, logs notifications, alarms and events, and performance statistics [168]. As shown in Figure 6.7, VNFM shares this information with the NFVO over the Or-Vi reference point. Since a VNS may have VNFs from multiple providers, it is important that NFVO can interact with them through the standard reference points.

6.4.3 NFV Orchestrator (NFVO)

NFVO is at the heart of the MANO architecture. It carries out two of its main functions: resource orchestration and service orchestration. Using its resource orchestration function, NFVO coordinates the acquisition and release of the NFVI resources by interfacing with the VIMs. NFVO instantiates the VNF Manager, which in turn manages VNFs as explained above. Service orchestration functionality deals with onboarding new network services using the information from descriptor files within various catalogs. For fault and performance issues, NFVO has to coordinate with the carrier’s OSS and multi-cloud management platform.

6.4.4 Catalogs and Repositories

MANO has several catalogs and repositories containing descriptor files, which NFVO uses to carry out the orchestration functions [170]. For example, there is a catalog for service onboarding templates and another for requirements for creation and operation of the VNFs. There is an NFV repository for storing all instances of network services and yet another for the available and used NVFI resources.

6.4.5 MANO Reference Points

All exchanges among the sub-systems of MANO and between them and external entities, including those pertaining to fault and performance status, take place through the defined reference points. Table 6.2 contains a brief description of these reference points and what fault and performance related information they carry [171]. Interactions between MANO and MMCP have not been defined in the NFV specifications. This has to be taken into consideration in a VNS fault and performance management solution.

Table 6.2 Fault And performance information over NFV reference points

Reference Point	**Endpoints**	**Functions**
OS-Nfvo	OSS and NFVO	1. Carries information related to VNS requirements from OSS to NFVO 2. NFVO creates VNS and applies carrier policies 3. Carries usage, accounting, fault and performance events for all VNS, VNF and NFVI resources.
Or-Vnfm	NFVO and VNFM	1. VNF and NFVO exchange information related to the creation and management of VNFs. 2. Forwards events related to VNF to the NFVO
Vnfm-Vi	VNFM and VIM	1. Carries information about NFVI requests from VIM.
Or-Vi	NFVO and VIM	1. Reserve NFVI resources for VNS 2. Coordinating scaling and release
Nf-Vi	VIM and NFVI	1. Creating/Obtaining virtual resources for creating VNS 2. Failure event, measurement results, and configuration information to VIM
Vn-Nf	VNF and NFVI	1. Physical and virtual resource information to VNFM for ensuring creation scaling and performance and portability of VNFs.
Ve-Vnfm-Vnf	VNFM and VNF	1. Event reporting by VNF to VNFM 2. Communication from VNFM to VNF regarding configuration and events 3. VNF aliveness check
Ve-Vnfm-em	VNFM and EMS	1. Same functions as Ve-Vnfm-Vnf for virtualization-aware EMS

6.5 Comparison of Competitive Network Service Orchestration Offerings

Since MANO components are important parts of the FP management of VNSs, we present some of the well-known ones in this section and compare their features relating to the management of FP problems that threaten the availability and reliability of these services. We include multi-cloud and multi-carrier domain support and interaction with the OSS, which are the important considerations for our discussions. Most MANO implementations are in initial releases and under active development. The idea, therefore, is to be representative and not comprehensive.

Table 6.3 Comparison of some competitive network service orchestration solutions from FP perspective

Platform / Criteria	ETSI	Linux Foundation	Open Networking Foundation	Gigaspace	Cisco	Netcracker (NEC)	Oracle
NFVO solution nomenclature	Open Source MANO (OSM)	OPEN-O/ONAP	XOS/CORD + ONOS[1]	Cloudify	Network Service Orchestrator	RT MANO Network Orchestration	Network Service Orchestration
Inception date	Launched 2016, Spearheaded by Carriers	Launched 2016	Launched 2015	Launched 2014		Launched 2015	Launched 2015
Current Release	Rel 4, 2018	Rel 2, 2018	Rel 1.10, May 2017	Rel 4.5, July 2018		Rel 12 May 2017	
Whether carrier-grade		Planned	Field Trials	Yes		Yes	Yes
Open Solution	Yes	Yes	Use case of open source ONOS	Yes, TOSCA based			Partly
End-to-End service	Planned	For defined use cases	For carrier use	Yes, may require plug-ins for underlay	Yes	Yes	Yes
Fault/Performance Management Sophisticaton[2]	Level 1	Level 3	Level 1		Requires extension with Crosswork Network Automation	Level 2	
Support for Multiple VNF /VIM	Yes	OpenStack VIM+ generic VNFM	OpenStack VIM+ VNFM like functions	Yes	Yes	Yes	Yes
Cloud platform neutral/Multiple Clouds	Yes	Planned	Multi-access edge cloud	Yes			
Integrates with BSS/OSS	Yes		Yes	Yes	OSS		Yes[3]

[1] ONOS is under Linux Foundation. CORD is under Open Networking Foundation.
[2] Level 1: e.g., log-based correlation; Level 2: includes a detection mechanism and root cause analysis; Level 3: predictive detection/localization
[3] Proprietary APIs

Table 6.3 gives a comparison of the Network Service Orchestration platforms. The following criteria have been used for classification [42][172]. A blank cell indicates that sufficient information is not available to adjudge the product on the corresponding criterion.

- Orchestration of end-to-end service is important from the carrier's point of view. In the absence of this feature, manual configuration and a large amount of scripting may be required to orchestrate complete services.
- Maintains the carrier-grade performance, which is more stringent than enterprise or IT application performance.

- Sophistication level of fault and performance management functions, we see that many of the platforms are yet to achieve the required level of sophistication of fault and performance management.
- Handling of multiple VNFM and VIM support allows management of SFCs across multiple carrier domains.
- Whether it can interact with different cloud and multi-cloud platforms.
- Coordinate with the OSS for fault and management functions.

This study helps to motivate our work and highlights its relevance in relation to the state-of-the-art.

6.6 Fault and Performance (FP) Problem Description

FP issues may range from simple single point failures to complex faults with many devices involved. A fault may appear because of some hardware or algorithmic error in the system. If the error is due to a malfunction or a deviation of the system from the accepted normal performance, then a fault would result. Additionally, one faulty entity may affect other neighboring entities and faults may propagate. In such a case, the faulty or other connected devices may give out notifications. The variety of FP issues that can affect the carrier networks is large and difficult to detect, diagnose and localize [41] [42] [43]. When we add to this the virtualization and the cloud computing layers, the number of ways faults can affect the virtualized network far exceeds that of their physical counterparts. In this setup, when faults traverse through the physical and virtual layers they change their presentation and produce a different set of markers in different layers, making it even more difficult to correlate an observed system disorder with the original fault [73].

Traditional failure detection mechanisms are ineffective or inapplicable in NFV environments. Traditional methods depend on probing or running tests on hardware, which are not accessible to the carriers who deploy services on virtual resources. Too much of probing or software testing may overload the VMs that have been optimized for the network function hosted on them. Attempts to apply other traditional methods, like rule-based approaches involving direct correlation of the markers with the faults, get mired in complexity and prove to be inadequate.

New methods are required to deal with faults in VMs or VNFs, which cause the VNSs to behave abnormally, even if the underlying hardware is fault-free. VMs are managed by cloud service providers, and VNFs by the network service providers making it difficult for the traditional systems to deal with problems in virtualized services. Consider a situation where the virtual private networks (VPNs) of many customers do not work. In this situation, FP detection and localization would require investigation all the VMs, on which virtual core router is hosted, the VNF that is working as the core router, the virtual network interface controller (vNIC) with fast Ethernet or Gigabit Ethernet ports and even the physical machines. Many alarms and other markers would result, which have to be correlated.

The fault detection mechanism should be able to separate out the error conditions that do not constitute a fault from the ones that do. The fault conditions have to be further classified into manifested or impending so that further action can be accordingly taken. As the name suggests, the manifested faults are those that have already occurred and have affected the system in some way. The impending problems may not have manifested as faults yet, but may soon materialize with varying degree of severity. After having detected and broadly classified the fault as manifested or impending, the system has to localize each of them to a level, which will be helpful, the operational staff to attend to the problems. We discuss in this section how faults are classified according to their criticality and see in detail the sharing of FP responsibilities among different platforms and enunciate the FP problem that this work solves.

6.6.1 FP Issues and their Criticality

As far as the VNFs and their interconnections are concerned, faults would happen due to algorithmic causes in the system software or in the application software. Faults in the application software affect the network functions or the links while those in the system software affect the VMs on which the VNFs are implemented. In the multi-domain scenario (multiple carriers), besides the usual faults occurring in the carriers' networks, there would be issues due to the interconnection of networks. For example, non-provision of a sufficient number of inter-carrier interconnections at the Points of Interconnect (POI) would lead to congestion and failure of calls from one network to the other.

Some events that cause alarms may not always be errors. For example, degradation in service can happen with some devices underperforming or because of being underprovisioned. Since, in such cases, the devices may still not be faulty, there may be no alarm or just a minor alarm. The degradation of a service can be detected through notifications, counters or meters set up to count events at the virtual function or the service level. Many of these markers would be routine warnings. At the same time, some alarms may be automatically taken care of by the network's resilience features, i.e., by using the redundant units instead of the one not performing properly. Some of these alarms may be coded to indicate the severity of the events. The confusion does not end here. There could also be problems with the management platforms themselves – multi-cloud platform, MANO, or the OSS/BSS. In this thesis, we confine ourselves to the faults of VNFs or of SFCs that affect the performance of VNSs.

ITU recommendation X.733 classifies the alarm events into the following four severity classes: Critical, Major, Minor, and Warning [35]. Critical alarms are caused when service to one or more users is totally stopped. If the service is highly degraded, but not stopped, then a major alarm may result indicating a condition that is preventing the service to be given as contracted. A minor event does not indicate present degradation, but if the condition is not corrected, it may cause a major fault to develop. A warning may be the most benign, but usually indicates an impending fault or performance issue which could eventually turn into a major fault. In addition to detection and localization functionalities, the predictive capabilities of the fault and performance monitoring system is able to indicate the faults that may develop and their severity levels. Impending faults are, thus, an important source of concern. It would be very helpful to the carriers if they can identify which performance deviations or impending faults may potentially result in an FP problem that would require personnel and material to resolve.

6.6.2 Shared FP Responsibilities

The fault and performance related responsibilities are jointly exercised by the MANO, the MMCP, and the OSS. Their interrelationship in the context of VNSs was illustrated in Figure 6.7. Table 6.4 summarizes the fault and performance related responsibilities of these management systems. As can be seen from the description, the functions of many systems overlap. For example, OSS and NFVO may both obtain information from the EMSs for knowing

the status of VNFs. Similarly, the marker collection functions of VNFM and EMS overlap. The precise distribution of FP related functionalities would, therefore, have to be done in the implementations. Standardized reference points among the management systems would help with interoperability of management functions of different carrier networks. Some of the reference points have, either not been defined, or not completely defined. These issues make the fault detection and localization problems more difficult to handle as complete information is not available with any system. Our framework uses information from various management platforms to improve FP management.

Table 6.4 Shared FP responsibilities of different management entities

Management Block	**Fault and Performance Functions**
1. MANO	
1.1 NFVO	NFVO orchestrates services and monitors parameters required to meet SLAs. It manages the lifecycle of VNSs and uses available resources or requests additional resources to maintain the required performance. For handling FP issues, it gets VNF level alarms from VNFM and NFVI level alarms from VIM. It interacts with OSS to share measurement results and notifications regarding network services. Its functions overlap carrier OSS function.
1.2 VNFM	VNFM interacts with VNF instances to obtain VNF related FP information like software inter-module communication failure. It also collects VNF-instance related NFVI information. It sends intelligence to NFVO for fault detection and localization. VNFM functionality overlaps with EMS functionality as both collect network function information.
1.3 VIM	VIM collects alarms related to physical and virtual resources contained in NFVI. It forwards FP alarms to VNFM and NFVO for broader correlation and root cause analysis. The fault information may include VM crashes, virtual port malfunction, storage failure, resource unavailability, etc.
2. **MMCP**	MMCP keeps an inventory of and monitors all virtual compute, storage and networking resources from different CSPs. It logs analytics for VM related faults. It adjusts resources to changing workloads and maintains the required performance level. The division of FP responsibilities among MMCP, OSS, and MANO is still to be finalized.
3. OSS	OSS monitors network services and resources and detects anomalous conditions. It interacts with EMSs to obtain the status of network elements. In the virtual network service environment, it may directly or through NFVO get information about VNFs. It correlates alarms from various sources to localize faults and performance conditions. Its functions spread from VNS down up to the VNF level.
4. EMS	Each network function/device is monitored and managed by an EMS. They collect operational status and alarms from VNSs and forward them to the OSS and VNFM.

6.6.3 The FP Problem Statement

The FP problem of the carrier networks can be defined as follows:

1) Detection of any condition that has already led to or could lead to degraded performance or failure:

The reasons could be manifested faults, hidden faults or inconspicuous deviations. The goal of FP issue detection is to sense and notify impending or actual fault and performance issues.

2) Identification and localization of manifested and impending faults:

The goal of FP issue localization is to first determine the broad category of manifested fault and then carry out a more fine grain classification. For impending faults it predicts the severity with which the fault may occur in future.

6.7 Methods for FP Management

During their operation, carrier networks produce large volumes of high dimensional data in the form of markers like alarms, notifications, observed behavior, warnings, counter values and measurement of performance indicators. These are discussed in some more detail in Section 6.8.1. The markers used by carriers are predominantly at the service and the virtual network function levels. Any FP management system should take into account all the relevant markers to carry out the required functions. Traditionally handling FP issues as part of FCAPS has been considered a difficult problem as abnormal behavior has to be interpreted from large amounts of high dimensional and noisy data [34]. There are three broad categories of methods FP management being explored by researchers lately: 1) NFVI level diagnostics 2) Causality inference based methods, and 3) statistical methods including AI based. We'll discuss each of these briefly here and take up a more detailed study of the selected method in the next section.

6.7.1 NFVI Level Diagnostics

We have seen previously that in VNSs, NFVI relates to the totality of hardware resources and the virtual compute, storage and networking resources created over these. The hardware component of the NFVI is in the domain of the CSP and generally inaccessible to the carriers. The methods in this category would rely on VM level alarms and metrics such as compute load or memory leak. These techniques thus rely on the monitoring and diagnostic techniques for cloud computing resources used for IT applications. An explicit or implicit assumption would usually be that the higher level alarms and other markers, e.g., those at network function and network service level, would usually have corresponding host level alarms which can be correlated to detect and possibly localize network function and service level manifested and impending issues. A correlation between telemetry information from the CSP and the higher level alarms in the domain of the carrier would have to be built up for diagnosing faults in the VNSs. Correlation of metrics with anomalies at the virtual layer has been applied by authors in [88]. The applicability of these techniques in a large distributed network needs to be studied.

6.7.2 Causal Inference Based Methods

These methods are also normally applied on VM level alarms like high CPU load and insufficient memory availability. The expectation here is that determining the causal relationship among them would help to get to the root cause of FP issues at the network function and service levels. The process involves looking for anomalous behavior based on VM level alarms, correlate alarms in pairs or clusters, determine causality, i.e., the effect of one alarm on the others and attempt to build causality templates that could be used for future alarms. The complex architecture and dynamics of NFV pose significant challenges from the point of view of causality inference. For instance, in [89], the authors carry out analysis of uncorrelated alarms in order to recover the pairwise causal relationship between them. To take care of the fact that higher-level faults (e.g., VNF or VNS levels) do not only depend on the pairwise relationship among VM level alarms, the authors propose clustering to infer multi-way causality templates. The patent documentation at [90] goes a step further and uses alarm data from different layers (e.g., NFVI and VNF). It takes into account the temporal proximity and the order of the alarm types in the clusters to make causality templates.

6.7.3 Statistical and AI-based Methods

The large volume of operational data generated in an operational telecommunications network could emanate from within one layer or across multiple layers and possibly contain many different types related and unrelated markers. In such a complex environment, it would be difficult to analyze the available data to produce information that can be used to manage FP issues. This situation, thus, creates a perfect set up for removing humans from the loop and resorting to machine intelligence. In this category, there are methods based on machine learning and deep learning that could be used for the detection and localization of FP issues.

There has been extensive work on performance modeling systems for distributed Internet applications of the pre-NFV era, notably TIPME (2000) [83], Pinpoint (2002) [84] and Magpie (2003) [85]. TIPME helps in identifying and eliminating causes of long response times. Pinpoint uses data mining to correlate the behavior of each active user request with the past failures and successes to determine failed components. Magpie works on individual user requests and compares the observed behavior, with saved normal models, to identify anomalous requests and malfunctioning components. Recently, the ‘mPlane’ consortium of European telecom companies and academic institutions, has worked on developing a measurement plane for Internet and CDN (2013-2016). The core of the project is ‘mpAD-Reasoner,’ which uses machine learning to detect anomalies involving multiple flows or users. It compares the current distribution with stored average distributions [166].

Researchers’ interest in AI-based machine intelligence for the identification of FP issues dates back to the era of expert systems [91] [92] [93] [94]. During the intervening decades, the carrier networks have undergone changes in technology and form, but the interest in intelligent fault handling has persisted. We look at AI as a way to empower machines to mimic and outperform human intelligence. Machine learning is a subset of AI, chiefly consisting of statistical techniques that allow machines to exhibit behavior that improves with learning. Deep learning is a way to implement machine learning using neural networks with more than one level of non-linearity. When using neural networks for difficult tasks, complex relationship among variables modeled with several levels of non-linearity improves the generalization process [95] [96] [97].

It has been shown that learning methods provide a way to relatively easily learn structure in the data and draw inferences [99]. Shallow machine learning algorithms, characterized by a single convolution stage, are suitable for cases where a large amount of labeled training data, including normal and fault cases, are available. They can derive intelligence from data and do not depend on experts to build complex interacting rules to derive patterns or models. Even dependencies, which cannot otherwise be explicitly modeled, can be learned. These advantages make them attractive for handling FP problems. In FP applications, machine learning methods can not only be trained with historical fault and performance data but can also be made to improve themselves as they operate and encounter new situations. This makes the machine learning systems, adaptive and intelligent and when they have been adequately trained, as they can generalize well from the training environment to the real-life situations. Use of different algorithms has been reported for detection and localization. We shall see more about this method in the next section.

VNSs are a new development and their deployment over multi-cloud is still to be explored fully. Many of the AI methods developed for intrusion detection have been tried, with varying degrees of success, for managing the FP issues. Some researchers have applied AI methods directly to the fault detection and, to a lesser extent, to fault localization. A very important reason for exploring AI for the problem of FP management for cloud-based NFV is the intractability introduced by the known gaps in the NFV specifications. We have seen in the earlier sections that interactions among multiple domains, especially between the legacy OSS and the MANO and the legacy OSS and the MMCP have not been fully defined [29]. ETSI supported proof of concepts (POC) have also resulted in highlighting the gaps in the NFV framework and carved out research work for the future. The present NFV framework, rather simplistically, assumes that VNFM will be primarily responsible for fault management actions. In real implementations, there will be multiple layers of cooperating fault managers. The OSS tackles customer fault reporting and management, which interacts with the EMS and NFV-MANO for the element level and VNFM level inputs, respectively. Besides, state change events for fault management actions have not been defined which are required for avoiding conflicting multi-layer actions and also an escalation from lower to higher layers. In this situation the learning methods of AI make the best use of the features learned from the available markers and can assist in FP management.

The authors in [39] use Artificial Neural Networks (ANN) for one and two alarms simulated scenarios. They show that in a simulated environment ANN provides better performance in comparison with the other implemented methods. The researchers in [101] propose a system for fault analysis and prediction in the telecommunications access network for the Rijeka area of Croatia. The Authors in [102] have used temporal decision trees for fault prediction in telecommunications networks. As per findings in [103], fuzzy cluster means can be used to classify network faults. The current research indicates the possibility of advancing the state-of-the-art in FP management through deep learning structures.

Random Forest machine learning method has been used in [104] to detect performance degradations in the VNFs. However, these researchers have chosen to rely on virtual resource layer level features data like CPU consumption, disk I/O, and free memory based on their suitability to computing systems. Evaluation has been carried out in a centralized IMS system.

Application of the proposed method to a highly distributed multi-domain network has not been reported. In a similar vein the authors in [105] have worked on the premise that underlying all the VNF failures are the NFVI level failures like disk I/O or memory usage. They propose Self Organizing Map (SOM), a type of unsupervised learning neural network, for clustering the statistical data and analyzing them to detect the faults. In [106], the author mentions that machine-learning algorithms are expected to detect invisible failures and anomalies. However, more work is required to validate them.

We now discuss in more detail the architecture and design of the HYPERVINES FP management framework for NFV deployment in the multi-cloud scenario.

6.8 Description of the Proposed FP Management Framework

The architecture and life cycle management in the cloud-based VNS made it VNS management is a collaborative process, among the elements constituting the VNS and the management systems involved, in creating and managing the service. VNSs impose new requirements on the FP management system. Gaps in the 'NFV on clouds' specifications in relation to the FP management, render rule-based approach inapplicable, and forced us to look at innovative techniques. The data generated by an operational system is large and high dimensional. In such a case, it would be very difficult to capture the intricate relationships among the features (e.g., the location of the fault, resources involved, markers produced, etc.) and the corresponding labels (faulty, non-faulty, impending fault, manifested, fault-severity, etc.) through traditional methods. Involvement of multiple layers of virtualization, generation of large amount of high dimensional operational data and unclear causal relationship between alarms and faults dictated the choices that we made in developing the framework.

6.8.1 Markers and Metrics for Fault Detection and Localization

We have introduced markers before as indicators produced by an operational network and measurements taken by the operations staff. There are a large number of markers that are directly or indirectly related to the occurrence of an FP issue. These markers become important features in our datasets. Events, that produce these markers, relate to communication, QoS, processing, equipment, and environment. Of course, not only each FP issue would usually have multiple markers, but also many of the markers would appear in more than one type of issue. Also, at any given time the markers produced may be a result of more than one FP issue. Thus, there is a complex relationship between the markers and the FP issues. This would usually mean that when using machine learning for fault detection and localization, feature engineering, i.e., selection of appropriate markers would be required to get better results. However, deep-learning models, are able to extract relevant features automatically, without human intervention. Some of the markers related to mobile, fixed and broadband networks are given in Table 6.5.

Table 6.5 List of markers for different carrier services

Broadband	**Mobile Network**	**Fixed Network**
Intermittent connection	Handoff alarm	Earth on a limb
Low data rate	BTS power alarm	No dial tone
NPOT	Packet loss counter	Loop resistance
Repeated training	Backhaul congestion	Line card port faulty
LAN lamp off	RX noise floor	Permanent ground alarm
Line noisy	Frequency error	Distribution cable fault
Port mismatch	Antenna tilt	DP fault
No ping	C/I ratio	Insulation measurement
ADSL lamp flashes	Signal strength	MDF fuse blown
No line sync	Radio link failure	Handset fault
Browsing issues	Cell site failure	Dis on one limb
Micro-Filter Faulty	Interference level	No incoming calls
No Communication	CQ indicator	Drop wire fault
Dropouts	Virtual eNB capacity	Ringtone fault
No authentication	Hypervisor alarm	Message fault
vRouter failure	Registration failure	Delayed dial tone
BTS: Base Transceiver Station, C/I: Carrier to Interference, CSSR: Call Set-up Success Rate, MDF: Main Distribution Frame, MU: Multi-User, eNB: eNodeB, NPOT: No Power in Optical Network Terminal, XCOA: Contact with AC, CQ Indicator: Channel Quality Indicator		

The metrics used by carriers to measure the health of the network provide important information about the FP problems at the macro level. Use of these as features in the dataset would help learning algorithms to narrow down the scope of localization effort. According to ITU Recommendation regarding the QoS criteria and parameters, a number of basic aspects have to be considered while identifying measurable metrics of service availability [173]. ETSI documents on service availability [79] and on service quality mention metrics that need to be collected and analyzed [80]. The ETSI group specification on service quality metrics recognizes that it is important to have an objective and quantitative metrics to assist in identifying problems when they arise and provide good service to the consumers. Examples of metrics, and their realistic values (where applicable), from an actual network [174] are given in Table 6.6.

Table 6.6 Metric for network availability and resiliency

Metric	Typical Value	Metric	Typical Value
Broadband Network		POI congestion	<0 .5%
Packet loss	< 1%	Assistance response	> 95%
Customer PoP to Internet exchange latency	<120ms	**Mobile Network**	
Peak international bandwidth utilization	< 90%	BTS total downtime	≤ 2%
Connection data rate availability	> 80%	Traffic Channel Congestion (TCH)	≤ 2%
Average throughput for packet data	> 90%	Call Drop Rate (CDR)	≤ 2%
Latency (audio)	<150ms	Call Set up Success Rate (CSSR)	≥95%
Fixed Network		Paging channel congestion	≤ 1%
Fault incidences	< 5%	Signal strength in vehicle	≥ 85dbms
Call completion rate	> 55%		
PoP: Point of Presence, BTS: Base Transceiver Station, POI: Point of Interconnection			

6.8.2 AI Techniques used in the Framework

A number of AI techniques, involving machine learning and deep learning, were are tested or finally used in the HYPERVINES FP management framework for the problem of detection and localization of FP anomalies. Following the practice of applied machine learning researchers, we designate models with a single layer of non-linearity, e.g., Support Vector Machine (SVM) and neural network (NN) with one hidden layer, as shallow structures or shallow machine learning architectures and the models with more than one layer of non-linearity, e.g., stacked autoencoders are referred to as deep structures or deep learning architectures [173] [176] [177]. We mentioned in the last chapter that it is common for shallow models with a linear hypothesis to have O(n) prediction time complexity and the training time complexity of O(l^2+n^3) where l denotes the size and n is the number of features in the dataset used. However, with such models, approximation errors are large for the high dimensional and large volume of data that are usually associated with the FP problem. Thus, if the data is not linearly separable then kernels could be used to map data into a higher dimension where it shows linear properties. This implies that linear models like SVM could be applied to the new space. This kernel trick reduces the approximation errors at the cost of higher complexity of the training time which is O(l^3 + l^2n) and prediction speed of O(ln). Of the prevalent shallow machine learning architectures, supervised methods (where each training example consists of the feature vector as well as a label) such as SVM and Random Forest (RF) are considered useful for diagnostic applications [145]. Another supervised learning technique, Bayesian Network (BN), has been applied to FP management in the industrial settings. Our preliminary exploration of these methods with small datasets has shown that SVM and Alternating Decision Tree (ADT) produce comparable and encouraging results for the detection problem. We will discuss the evaluation results in the next section.

In deep learning, increasingly improved features are learned as the hidden layers are traversed. Learning of complex features and structure in the data can be broken down into simpler tasks performed at many levels. This way, deep learning can achieve low generalization errors, even for functions otherwise difficult to represent [177]. Lately, better results than SVM have been achieved with deep neural networks in a number of important applications [175] [178]. A key advantage of deep learning over shallow learning is the automatic extraction of high-level features. Each algorithm that we have used is briefly described here. Some of these might have been described in a different context before; the focus here is not their applicability to the FP problem. For more details, readers may consult the references mentioned.

1) Support Vector Machine (SVM): Geographically dispersed elements of the network may generate similar or different markers at different locations, for example, at the carrier's OSS location or the NFV provider's MANO location. The information contained in these markers is non-unique across the domain of faults and performance issues. The SVM classifier can analyze the data and learn inherent patterns, which are otherwise not evident to the human senses. It works by finding optimal hyperplanes that separate different classes in a given labeled dataset. Once trained, it can classify unseen data. References at [136] give a more detailed description of SVM. As we have use SVM in our framework, we mention some more details of parameter C, Υ and ϵ that require careful selection to minimize prediction errors. As the exact solution is impractical, precision ϵ is used to indicate the error insensitive tube around the decision boundary in which the errors are ignored. The aim is to minimize $||\omega||^2$ which is equivalent to maximization of the margin between the classes. The constant C determines the tradeoff between the flatness of function learned and the amount of error allowed above ϵ. A low C makes the decision surface smooth; a high C aims at classifying all training examples correctly by giving the model freedom to select more samples as support vectors. We choose how significantly the misclassifications should be treated and how large the insensitive loss region should be, by selecting suitable values for the parameters C and ϵ. The data $\boldsymbol{X}$ is projected to a higher dimension using function $\phi(\boldsymbol{X})$. Poor generalization and computational complexity that may result from projecting data to higher dimensions can be avoided by the use of a kernel function that maps the input feature space of dimension *d* to a higher dimensional space in which the relationship becomes linear. In our studies, we have found that the performance of the Radial Basis Function (RBF) kernel performs better than others. The RBF kernel has the form given below. Here, $\mathbf{x_i}$ and $\mathbf{x_j}$ are two sample feature vectors, and Υ is the parameter that sets the spread of the kernel.

$$K(xi, xj)=\exp(-\Upsilon\|xi-xj\|) \tag{6.1}$$

In all cases where SVM had been used, these parameters had been arrived at by a grid search.

2) Alternating Decision Trees *(ADT):* This method combines Decision Trees with Boosting. The ADT is different from normal decision trees as it has predictor and test nodes alternately, while the normal decision tree has just test nodes with each branch representing an outcome of the test. Another difference is that while each leaf can only be split once into two, in ADT each part can be split multiple times. This increases the accuracy of classification/regression. The splitting criterion could be impurity based like information gain or Gini index or based on a statistical test like chi-square. Boosting, on the other hand, brings in performance-enhancing capabilities. However, it adds more test and predictor nodes. The complexity is quadratic in boosting iterations, but can be reduced by using a suitable heuristic [180].

3) Random Forest: Among supervised learning algorithms of its class, the Random Forest (RF) is a classifier that is likely to give more accurate results. It proves to be efficient and robust in many use cases with large databases. It can help in feature selection by estimating the relative importance of the predictor variables. This is done by selecting an impurity measure like entropy and measuring the contribution of each feature. Another very useful feature is that it does not need separate test data or any cross-validation. The Out-of-bag error (OOB-error) gives an unbiased estimate of test or classification error [181].

4) Deep Learning using Stacked Sparse Autoencoder: An autoencoder is a neural network, which has an input layer, an output layer, and one or more hidden layers. It learns the feature of a dataset in an unsupervised manner (i.e., the training examples are just feature vectors with no labels). Such a model reconstructs the input values at the output with accuracy depending on how well the features are represented by the hidden layer(s). A sparse autoencoder (SAE) contains a hidden layer with a smaller number of neurons than the inputs. Thus, the high dimensional inputs are mapped to a lower dimension forcing them to learn the best representations of the given features. Extraction of features takes place according to their relative importance. More than one sparse autoencoders can be put in tandem to construct a stacked sparse autoencoder (SSAE). Training of the stacked autoencoder is done in a layerwise greedy manner. The first layer is trained with the input data $\mathbf{x}$ to obtain weights ω and bias b for the hidden units such that the output $k(f(\mathbf{x}))$ is as close to the input as possible, i.e., minimizes the loss function $Ø(x, k(f(\mathbf{x}))$ [182]. The L2 norm (mean square error) is often used as the loss function. The primary feature activations of the first hidden layer are then used as input to the second hidden layer and so on. Since the L2 norm may not reduce the error to zero, a sparsity penalty term is added to constrain the neurons to be mostly close to zero. The training criterion can be written as $Ø(x, k(f(x))) + \Omega(h)$, where $\Omega(h)$ is the sparsity penalty.

If we consider an SSAE with n layers then the weight and bias parameters for the mth autoencoder can be written as $\omega^{(m, 1)}$, $\omega^{(m, 2)}$, $b^{(m, 1)}$, $b^{(m, 2)}$. The encoding step in the feed-forward direction for each layer k of the stacked autoencoder is given by:

$$h^{(k)} = f(x^{(k)}) \tag{6.2}$$

$$x^{(k+1)} = \omega^{(k,1)}h^{(k)} + b^{(k,1)} \tag{6.3}$$

The decoding stack of each autoencoder is run in the reverse order

$$h^{(n+m)} = f(x^{(n+m)}) \tag{6.4}$$

$$x^{(n+m+1)} = \omega^{(n-m,\ 2)} h^{(n+m)} + b^{(n-m,\ 2)} \tag{6.5}$$

Then, as the layer-wise training proceeds, each successive layer learns increasingly more and more useful features with the innermost layer $h^{(n)}$ giving a representation of the input in terms of the most compressed and useful features for the input of higher dimension. With appropriate settings of the parameters, the compressed layer reconstructs the original input with good accuracy. Good reconstruction performance helps in achieving good prediction. For prediction of fault classes or severity of impending faults, a layer of Softmax classifier replaces the decoder layers with $h^{(n)}$ forming the input to this layer. Softmax regression can be used for multi-class classification as it gives probabilities of output being close to the target value in the range 0 to 1 with the sum of probabilities being 1.

Table 6.7 summarizes the machine learning and deep learning techniques useful for NFV-Cloud FP problems.

Table 6.7 Machine/Deep learning algorithms for the FP problem

Algorithm	Advantages	Watch out for
Support Vector Machine	• Works well for the detection problem. • Works with linearly separable as well as non-linear feature space (with RBF kernel).	• Select kernel function and fine-tuning of parameters. • Select the cross-validation method carefully. • Long training time with the big dataset.
Random Forest	• Works well for the detection problem and localization of manifested faults. • Works for binary as well as multi-class classification. • Less prone to overfitting. • Handles non-linearity. • Handles categorical features. • Handles high dimensional spaces and a large number of examples.	• Fine tuning of parameters like the number of features in any tree, number of trees in the ensemble and leaf size. • Watch out for classification time and complexity of the model.
ADT	• Works well for the detection problem. • It has the speed of a decision tree and is robust to noise and missing values. • It can be used for mixed categorical and numerical data. • It helps in finding significant features.	• Must be used carefully to avoid overfitting. • Keep control of parameters like depth and number of features to split on.

Autoencoder/Stacked sparse autoencoder	• Useful for localization of impending problems. • It gives better control over quality. • With the appropriate number of layers and neurons, it performs better than the shallow algorithms	• Sensitive to number and size of layers. • Careful fine-tuning of sparsity and regularization parameters is required.
SoftMax	• Used as the last stage of stacked autoencoder in the localization problem. • Trained in a supervised manner. • It can do binary as well as multi-class classification. • It can be used for prediction of faults, severity, etc.	• Watch for bias due to the distribution of data. • If sufficient labeled data are available fine-tuning by backpropagation may improve results.

6.8.3 Design and Implementation of the HYPERVINES Framework

The operational basis of the HYPER-VINES framework is to consume markers from large volumes of multi-source and high dimensional operational data to accurately detect and localize faults and performance issues of VNSs in a carrier's environment. Figure 6.8 gives a simplified illustration of the process.

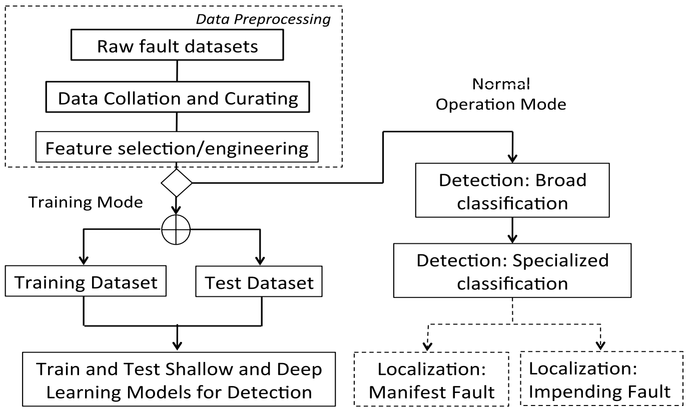

Data Preprocessing
Raw fault datasets
Data Collation and Curating
Feature selection/engineering
Normal Operation Mode
Training Mode
Training Dataset
Test Dataset
Train and Test Shallow and Deep Learning Models for Detection
Detection: Broad classification
Detection: Specialized classification
Localization: Manifest Fault
Localization: Impending Fault

Figure 6.8 HYPER-VINES Fault and performance management mechanism

In the training mode, the data is curated and partitioned into training and test datasets. Shallow machine learning and deep learning models, used in various stages of detection and localization, were trained and tested. The models were fine-tuned till the mean square error of classification or prediction is optimized. The framework is called hybrid as it involves both machine leaning and deep learning models. During operation, the generated markers are pre-processed and run through the trained models for detection and localization. We discuss more details of the process in the next sub-section.

The relationship of the HYPER-VINES framework with its environment is shown in Figure 6.9. It collects performance markers primarily from MANO, OSS and the cloud management platforms over standard interfaces. Additional information about operational statuses of individual VNFs comes from the EMS via OSS or is pulled directly by HYPER-VINES. Since HYPER-VINES obtains markers from multiple sources, it mitigates the problem of overlapping responsibilities and ill-defined interfaces of the management platforms.

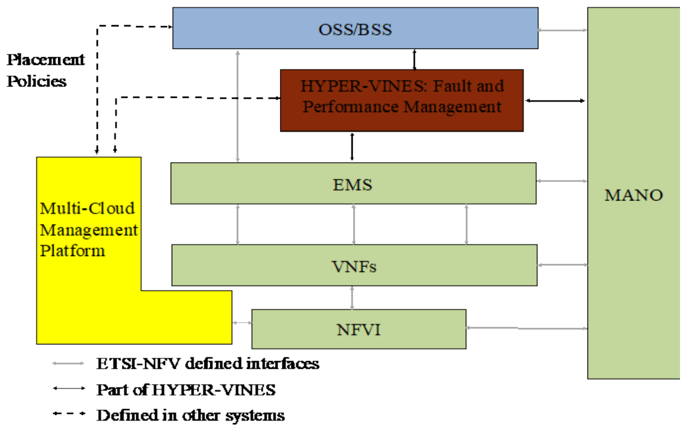

OSS/BSS
Placement Policies
HYPER-VINES: Fault and Performance Management
MANO
EMS
Multi-Cloud Management Platform
VNFs
NFVI
ETSI-NFV defined interfaces
Part of HYPER-VINES
Defined in other systems

Figure 6.9 The environment of HYPER-VINES

The internal architecture of HYPER-VINES is shown in Figure 6.10. The main sub-systems are the Detection and Localization functions with an optional inclusion of the data pre-processing module. We discuss the important details of the framework below.

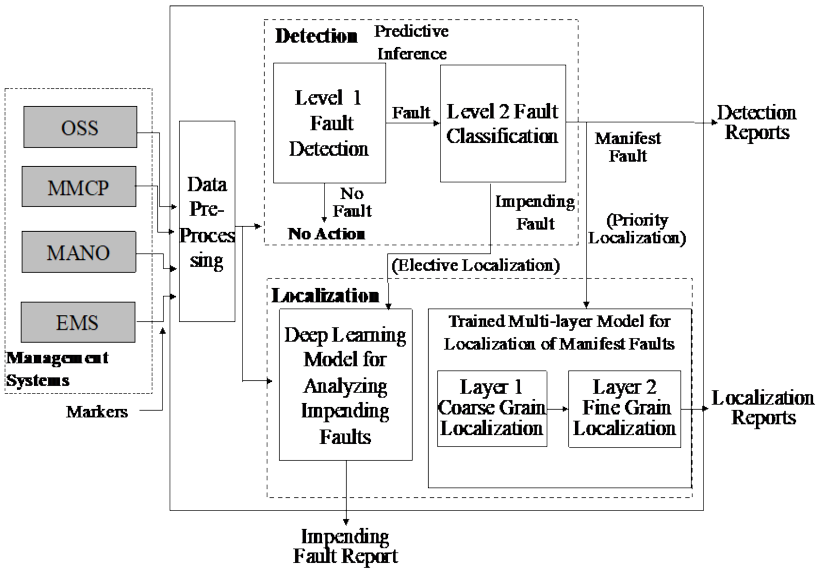


Figure 6.10 The Architecture of HYPER-VINES Framework

Markers available from the management platforms, viz., runtime monitoring and measurements, alarms, notifications and warnings, configuration changes, measurements and environmental factors are used along with machine learning models trained with historical data to draw inferences about the manifested performance and fault issues. Additionally, the capability of deep learning to map the intricate relationship among the features has been used to predict the impending faults. The framework shown in Figure 6.10 consists of three main sub-systems: Data pre-processing, Detection and Localization.

6.8.4 Data Pre-processing

The marker data, obtained from various sources, are collated and normalized to remove biases. We have also tested reduction of features based on some criterion like correlation with the labels. Tools like Weka have been used to select features based on correlation with the labels [33]. With the dataset used in this study, inclusion of features up to a correlation threshold of 23% improved accuracy. In the training mode, the available dataset is split into training and test datasets, which are used to train and test all the models. During operation, the marker data is run through the framework to detect and localize problems. We shall see more details of the actual datasets in the next section.

6.8.5 The FP Detection Subsystem

The first part of the FP problem, i.e., detection is essentially a two-stage binary classification problem that first classifies the outcome as 'normal performance' or 'abnormal performance' or alternatively as 'fault' or 'no fault' classes. Then for the 'fault' or 'abnormal performance' cases, it decides whether the problem is *manifested or* impending. We shall see in the next section, why a two-stage model is better in this case. It is important for the detection models to have good accuracy, as manpower and material resources are committed for rectification of detected and localized faults. This is particularly important, as the presence of alarms does not always indicate a fault.

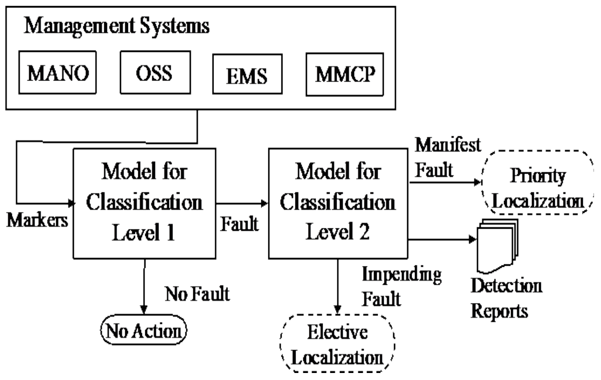


Figure 6.11 The detection subsystem

The detection sub-system of the FP management framework is shown with more details in Figure 6.11. In the two-stage implementation for detections, both the levels use the shallow machine-learning models. As mentioned before, these models are trained on historical data consisting of FP events, present markers including the severity levels and the fault clearance description that the maintenance staff has entered after rectifying the fault. The cases, where no action is required or the fault is transient and corrects itself, are labeled as 'no-fault.' In the case of an actual fault, the nature of the fault and its actual clearance is indicated. The trained model can then take markers resulting from new events as inputs to decide at Level 1 whether the conglomeration of markers constitutes a fault. If it does, then the model at Level 2 uses the available information to decide whether the fault is impending or manifested. The use of markers from many management platforms may introduce redundancies, as a good amount of similar information may be available from OSS and MANO. However, redundancy is good for our framework and makes up for the gaps in communication among various management platforms. The occurrence of multiple faults, the overlap of markers among faults and conflicting markers render the task of detection difficult. If our detection sub-system is effective and can correctly segregate the conditions, then localization has better chances of succeeding. A two-level model for detection helps in filtering out a large number of ‘no-fault’ cases at level 1 so that level 2 is largely applied to the ‘fault cases.’ This makes classification better and faster.

Algorithm 6.1 describes the process of detection. X is the vector of predictor variables. Hyper-parameters {pd} and {pd'} pertain to detection models at the two layers, {ps} and {pn} are for models at the Localization Layers 1 and 2 and {pi} are for deep learning model for impending faults.

The procedure detect_level1 at line 1 takes the feature vector of a new event and populates the hyper-parameters (line 3). The trained machine learning model is used to predict labels. If it is ‘fault’ condition then detect_level2 is invoked (line 8) which uses another trained model to classify the fault as ‘manifested’ or ‘impending’. Thereafter, the appropriate localization module is called (line 13 or line 15) to handle the manfiested fault localization or the impending fault localization. Use of X' and X" indicates the possibility of curating the feature vector used with the corresponding model. This algorithm also outputs the detection report, which includes fault cases as well as the type of faults.

Algorithm 6.1: Detection Levels 1 & 2

```
1: procedure detect_level1 (X)
2: #fault/no-fault classification
3: {$p_d$} ← values of hyper-parameters for the chosen model
4: use trained model for detect_level1 with X, {$p_d$})
5: if ‘fault’ is true
6:    call detect_level2 (X')
7: produce detection report
8: procedure detect_level2 (X')
9: # classify as manifested/impending and call localization
10: {$p_d$'} ← values of hyper-parameters for the chosen model
11: use trained model for detect_level2 with X,Y, {$p_d$'}
12: if manifested is true
13:   call manifested_localization (X") #defined in Algorithm 6.2
14: elseif impending is true
15:  call impending_localization (X") #defined in Algorithm2
```

6.8.6 The FP Localization Subsystem

The second part of the FP problem is the localization of the detected faults. Many of the manifested faults, that have made themselves evident, could be major or critical threatening to seriously cripple the network service from which they originate. Localization of manifested faults is, therefore, taken up on priority while immediate localization of impeding is elective, nevertheless important.

Since many faults may propagate and produce secondary markers at other places in the network, the localization process has to cut across layers and domains to identify the faulty devices, links, and software correctly. Our framework localizes manifested faults using a multi-class, two-layered model shown in Figure 6.12 (extracted from Figure 6.10). We now describe some more details of the manifested and impending fault localization.

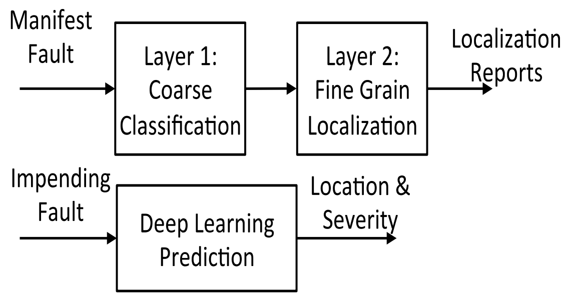


Figure 6.12. The localization process

Manifested Faults: During the operation, all the FP issues classified as 'manifested' pass through the two layers of localization. At Layer-1, the model works as a multi-class classification model that classifies the faults into one of the several broad categories of FP issues. Table 6.8 gives examples of three such categories, 'Network Performance,' 'Security' and 'Virtual Resource.' The model at Layer-2 is also a multi-class classification algorithm that localizes the FP issue at a finer granularity (e.g., a device, interface, or link) within the broad category predicted at Layer 1. The localization sub-system produces localization reports that can be used by the maintenance staff to carry out the rectification work. For the multi-class classification with SVM, we chose to work with simple models like One vs. One (OvO) and One vs. All (OvA) [33]. We eventually selected OvA since it provided more accuracy and was comparable to OvO in training and actual operations. In the OvA approach, for the i^th^ classifier f_i, the examples can be classified with $f(x) = \arg\max_i f_i(x)$, i.e., choose the class that classifies the example with the maximum margin

Table 6.8 Coarse (Layer1) and fine (Layer 2) categorization of faults

Layer 1 Fault	Layer 2 Fault	Markers
Network Performance	Traffic and Beacon Channel plan	Bad receive quality, Call drop at the cell boundary, Link degradation
	Handoff parameters setting	
	Bit error rate	
	High paging discard rate	
Security	Denial of Service attack	Client Authentication failure, Call initiation failure
	Home Subscriber Server Failure	
Virtual Resource	VM Fault	Network function failure alarm
	Hypervisor fault	

The procedure manifested_localization (line 1) uses procedure localize_layer1 (line 4) to determine the broad category of manifested fault. Depending on the category determined, it calls the localize_layer2 with corresponding parameters. For each category at Layer 1, the Layer 2 may have a specifically trained model. For Impending fault localization the procedure impending_localization (line 17) calls the deep learning model with the required parameters. Let us discuss a little more about the manifested and impending faults.

Algorithm 6.2: Localization Layers 1 & 2

```
 1: procedure manifested_localization (X)
 2: # Coarse grain localization
 3: {p_s} ← values of hyper-parameters for the chosen model
 4: call localize_layer1( X,{p_s})
 5: # fine grain localization with the appropriate model
 6: if class_category ==1
 7:   {p_1} ← hyper-parameters class_category 1
 8:   call localize_layer2(X",{p_1})
      …
 9: if class_category==7
10:   {p_7} ← hyper-parameters class_category 7
11:   call localize_layer2(X",{p_7})
12: produce localization report
13: procedure localize_layer1(X,{p_s})
14: use trained model localize_layer1 with (X,{p_s})
15: procedure localize_layer2(X",{p_n})
16: use trained model localize_layer2 with (X, {p_n})
17: procedure impending_localization (X)
18:{p_i} ← parameters neurons, sparsity parameters
19: use deep_learning_model (X,{p_d})
20: produce impending fault report
```

Impending Faults: In traditional systems, in the absence of predictive analysis, preventive maintenance is relied on to catch issues early. In our framework, localization of impending faults consists of predicting the severity and location of the fault. In a stable operational network, most of the markers would constitute normal data with markers indicating anomalous conditions interspersed sporadically. While our data has more than 800 features, any anomalous condition would present <5% of these! In other words, the data are quite sparse. Impending faults may also contain previously unseen faults. Thus, while manifested faults are manageable with shallow models, impending faults have been tackled with deep learning. We have used Stacked Sparse

Autoencoder (SSAE) (a type of deep neural network). A single SAE contains an input, an output, and a hidden layer. With an under complete hidden layer, the autoencoder learns the most useful individual features as well as creates composite features. The advantage can be accentuated with stacking a number of autoencoders and carefully designing the hidden layers.

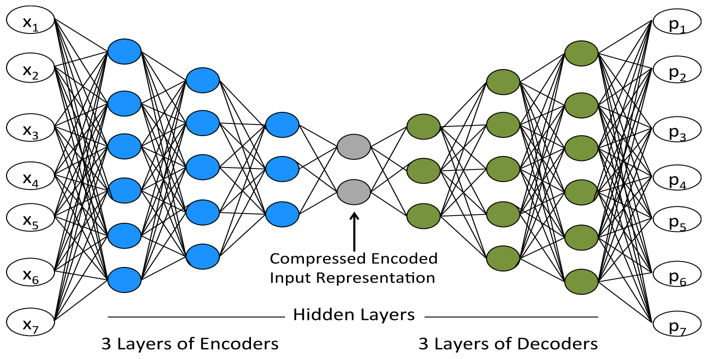


Figure 6.13 The stacked encoder used for prediction

Figure 6.13 shows the stack of three sparse autoencoders used in this work: the input layer (x), an output layer (p) and three hidden layers consisting of paired encoders and decoders. The colored neurons show three corresponding pairs of encoders and decoders. By reducing the size of hidden layers, the output is made reliant on increasingly lesser but richer features. Such a network can be trained in an unsupervised mode to reconstruct input data at the output with good accuracy. These networks can be tuned well for sparse data by using parameters like sparsity regularization and sparsity proportion as discussed in the evaluation section.

We train our model to have a good reconstruction of the input at the output (decided by the L2-norm), with unsupervised data, in a layer-wise greedy method (one hidden layer at a time). A model that reconstructs well also gives good predictions [172]. During training, features (z) learned by each hidden layer are input to the next layer. Pairs of {weights, biases}, viz., (ω1, b1), (ω2, b2) and (ω3, b3), are learned in achieving good reconstruction.

$$\{\omega_k, b_k, \omega_{k'}, b_{k'}\} = \begin{cases} \text{argmin}\{\text{L2\_norm}(x, x'), k=1 \\ \text{argmin}\{\text{L2\_norm}(z_{k-l}, z_{k-l'}\}, k>1 \end{cases} \tag{6.6}$$

$z_1 = f(\omega_1, x)$ (6.7)

$z_k = f(\omega_k, z_{k-1}), k>1$ (6.8)

After achieving good reconstruction of the input, the decoders are removed, and a prediction layer is added in tandem with the encoded representation layer (Figure 6.14). Softmax assigns decimal probabilities to each class in a binary or multi-class problem. These decimal probabilities must add up to 1. This additional constraint helps training converge more quickly than it otherwise would. In simple terms, the Softmax function can be written as

$$F(y_i) = \exp(y_i)/\sum_{j=1,k} \exp(y_j), \quad i=1, 2, \ldots, k \qquad (6.9)$$

Softmax uses the rich features from the encoded layer of the stacked autoencoder to learn its weights ω_4 and biases b_4. Training of Softmax is done in a supervised manner using the labeled examples available. ω_4 are the weights for minimum prediction mean square error (MSE). It produces predictions y' for the given labels y. Thus, for labels y and its prediction y' we have,

$$\{\omega_1, \omega_2, \omega_3, \omega_4\}=\text{argmin}\{L2\_norm(y, y')\} \qquad (6.10)$$

After the Softmax classifier has been trained in a supervised manner, the whole model is fine-tuned using back-propagation and simultaneous adjustment of weights of all the layers to minimize the mean square error in the labeled test datasets [29].

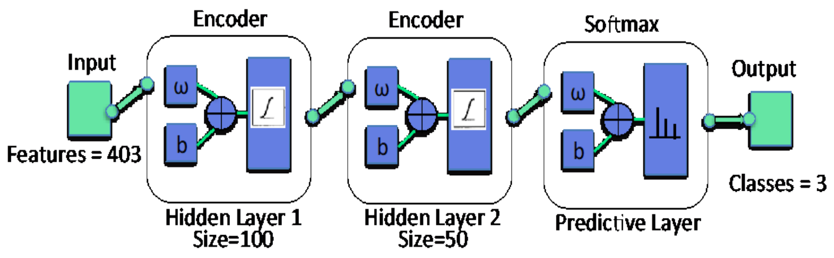


Figure 6.14 Stacked sparse autoencoders

6.9 Evaluation of the Model

In this section, we will discuss how our FP detection and localization framework that we have described in the sub-section 6.8 has been evaluated. More specifically, we will see the training dataset used, curation of data, and the performance of the trained models for the unseen events. Curating may involve one or more of the following activities to improve the outcomes: feature pre-selection using some kind of technique to correlate features with the labels, cleansing of data, pruning or integration, synthesis or analysis of features. The performance of the HYPER-VINES framework is demonstrated by good accuracies achieved by the detection and the localization subsystem.

6.9.1 Analysis of the Datasets Used

Having access to good quality datasets is important for proper training of the learning models and their predictive performances [183]. Records like fault dockets, switch room logs, outdoors logs, personal records of maintenance staff and fault closure reports contain a vast amount of information about complaints, faults, test results and restoration details of telecommunication networks. However, assembling a useful dataset from these primary data is not an easy task. Since network fault and performance datasets are not easily available, researchers commonly resort to either proprietary datasets that are not publicly available or generate synthetic datasets [101] [103]. We have used in our studies the real network FP dataset pertaining to faults and disruptions in telecommunication carrier Telstra's network drawn from real fault-logs [184]. The dataset, as available, is split into a number of sub-datasets, each containing different information derived from the logs. These sub-datasets give *event_type, log_feature, resource_type* and *severity_type*. They are related through the *"id"* column that acts as the key field and also conveys the timing information. It can be used in innovative ways to improve predictions based on the dataset. The *event_type* is the type of fault or performance incident. Any anomalous situation may have up to 5 different events associated with it. The *resource_type* gives the affected virtual resources. The feature *fault_severity* is given in terms of the number of faults: many faults (2), a few faults (1) and no faults (0). The *'log-feature'* file identifies features or markers like alarms and notifications by their numbers. There can be up to 386 features associated with an anomalous event. The *severity_type* rates the warning conditions in terms of their seriousness (on a scale 1 to 5 with 5 being the most serious).

The training dataset contains *"id,"* the location of the incidence and the severity of the fault. The rest of the fields can be extracted from the other sub-datasets to make a complete dataset for training detection models. In the case of localization, the available sub-datasets as collated with the training dataset such that the localization model gives a good prediction of severity of faults. An extract from the training and test datasets are given in Table 6.9 (a) and 9 (b) respectively. The test dataset has *"id"* and the *location* for which severity has to be predicted.

Table 6.9 (a) Training dataset | Table 6.9 (b) Test dataset

id	**location**	**fault_severityity**	**id**	**location**
4757	location 508	0	13484	location 922
16358	location 257	1	12392	location 184
11810	location 116	0	2322	location 1019
7274	location 830	1	567	location 734
4311	location 704	2	4436	location 236
12261	location 1089	2	12156	location 124
14752	location 653	0	7508	location 858
3304	location 1099	1	6184	location 707
9012	location 975	0	12213	location 763
9928	location 1019	2	6458	location 1100
10013	location 696	0	13967	location 155

The Telstra *log_feature* sub-dataset contains 58,672 examples, with events displaying the presence of different features. The *event_type* sub-dataset has 31,170 examples, the *resource_type* sub-dataset has 21,076 and *severity_type* sub-dataset has18,552 examples. The test and the training files have 11,171 and 7381 records respectively. They have not been split from a common dataset so the standard 80:20 or a similar ratio is not maintained. A dataset prepared by the consolidation of all sub-datasets has more than 800 features as shown in Table 6.10. Each fault (with a unique id) is associated with a location, up to 6 features and corresponding volumes, up to three affected resources, up to 5 events, and up to 5 severity types indicating the intensity of the warning and fault_severity ranging from 0-2 as explained before.

Table 6.10 List of features from network fault dataset

No of Features	**Feature Name**	**Explanation**
1	id	Unique id for an anomaly situation. It contains a time-stamp.
2	location	Location of the event
3-12	resource_type	Up to 10 resources may be involved
13-398	feature	There are 386 types of markers of which usually a few will be present
399-797	volume	There is volume information for each feature present
798-802	event_type	Up to 5 event_types may be associated with an anomalous situation
803	severity_type	Indicates severity of warning for the situation. The scale is 1-5 with 5 being the most severe
804	fault_severity	0 indicates no fault, 1 indicates a few faults and 2 indicates many faults

A part of the consolidated Telstra dataset is shown in Table 6.11. Only feature1 (out of the complete set of features from feature1 to feature386) is shown for compactness. As part of preprocessing of the dataset, selection of features was carried out based on the degree of correlation of each feature with the labels using the Weka tool [147]. With the dataset used in this study, a correlation threshold of 23% was found to improve accuracy.

Table 6.11 Consolidated training dataset

id	location	fault_severity	resource1	resource2	event1	event2	event3	event4	severity_type	feature 1	volume 1
8	location 243	0	resource_type 2		event_type 34	event_type 35			severity_type 2	232	3
13	location 418	0	resource_type 2		event_type 35	event_type 34			severity_type 2	232	1
19	location 644	1	resource_type 2		event_type 42	event_type 44			severity_type 1	368	2
20	location 79	0	resource_type 2		event_type 54	event_type 11			severity_type 2	55	1
23	location 257	0	resource_type 8	resource__type 2	event_type 35	event_type 34	event_type 10		severity_type 2	307	1
24	location 367	0	resource_type 2		event_type 35				severity_type 4	312	2
26	location 238	0	resource_type 2		event_type 35				severity_type 4	312	1
27	location 793	0	resource_type 8		event_type 11				severity_type 1	73	3
28	location 889	0	resource_type 8		event_type 11				severity_type 2	68	2

6.9.2 Evaluation of the detection subsystem

To prepare the data for Level-1 detection, the fault_severity has been curated to have binary values with 0 indicating 'no-fault' and 1 indicating 'fault.' The detection classification of 'fault'/'no-fault' was implemented with a number of supervised learning techniques of which SVM, ADT, and RF have been shown in Table 6.12. On the basis of accuracy, SVM and ADT perform comparatively better than RF. In each case, 10% cross-validation was used.

Table 6.12 Stage-1 detection results

Benchmark Algorithm	SVM	ADT	Random Forest
Time taken	0.01 seconds	< 0.01 seconds	0.1 seconds
Correctly classified instances	95.42%	95.00%	86.67%
Precision (Average)	95.7%	95.2%	86.9%
Mean absolute error	0.0458	0.0859	0.2509
Root mean squared error	0.2141	0.2092	0.3261
True positive for class 0	94.3%	94.3%	95.5%
False positive for class 0	2.4%	3.6%	30.1%
True positive for class 1	97.6%	96.4%	69.9%
False positive for class 1	5.7%	5.7%	4.5%

With this dataset, SVM, on the whole, performs better than ADT and RF giving ≥ 95.4% accuracy. Considering the definitions in Table 6.13, the true positive (TP) rate for 'fault' cases were the highest for SVM showing that these were correctly classified as 'fault' cases. Considering the nature of the dataset, this result indicates a good result. There were no faults and system said fault in 5.7% cases, while there were faults and system said no faults in 2.4% cases. The false positive and negative rates were the lowest in SVM and the highest in Random Forest. A desirable outcome is that besides classifying faults and faults and no faults as no faults with high accuracy, it classifies a very low percentage of faults as no-faults, thus, helping to do what is intended to do – detect performance and fault issues. SVM and RF also gave high precision indicating that they correctly classified 'no-fault' cases.

Table 6.13 Metric used

Metric	**Interpretation**
Accuracy	(TP+TN)/(TP+TN+FP+FN)
Precision	TP/(TP+FP)
Recall	TP/(TP+FN)
TP=True Positive, TN=True Negative, FP=False Positive, FN=False Negative	

To get a sense of the performance of our detection model, using SVM with RBF Kernel, we compared the results with baseline results obtained by One-R model. One-R is a simple but accurate classification algorithm, which generates one rule for each predictor and then selects the one with the smallest error. Running on our datasets, the baseline result was about 74%, which indicates that our chosen model gives a substantial improvement over the naïve baseline. The comparative baseline and the accepted model (SVM in this case) results are shown in Table 6.14 and Figure 6.15

Table 6.14 Detection Level 1 performance

Parameter	**HY-V (%)**	**ONE-R (%)**
Accuracy	95.41	74.55
Precision	95.7	78.6
Recall	95.4	73.3
No-faults as faults	5.7	11
Faults as no-faults	2.4	15

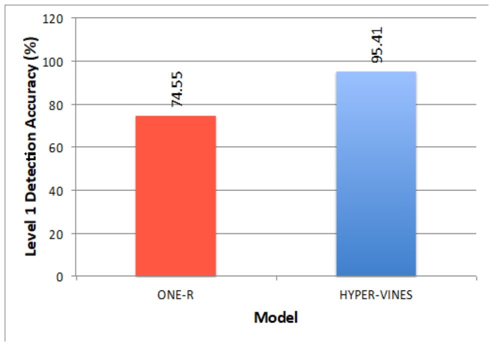


Figure 6.15 Detection Level 1 performance graph

At Level-2, the detection module classifies the fault cases as 'manifested' or 'impending.' For the Level-2 classification into manifested/impending classes again a tuned SVM with RBF Kernel works well as can be seen from Table 6.15.

Table 6.15 Detection Level 2 performance

Metric	**HYPER-VINES (%)**
Accuracy	95.1
Precision	93.0
Recall	93.0
Rate for correct prediction of impending faults	95.1
Rate for correct prediction of manifest faults	89.7

For Level-2 detection, we have chosen One-R as the baseline algorithm. The accuracy of our framework is 13.03% better for 'impending' faults and 5.97% better for 'manifested' faults, which is a significant improvement (Figure 6.16.)

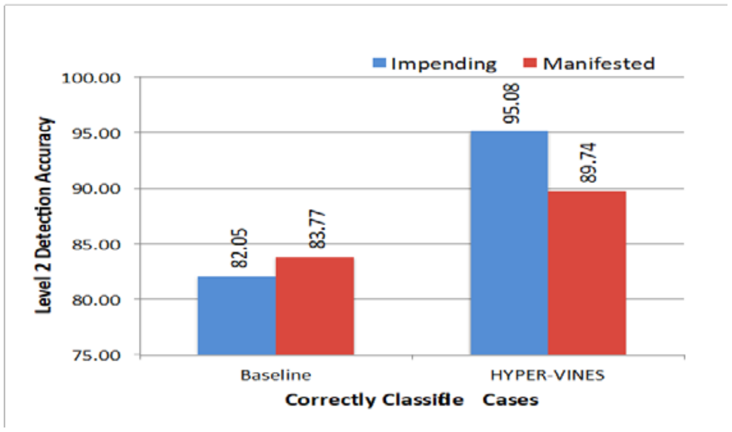


Figure 6.16. Detection Level 2 effectiveness compared to baseline

6.9.3 Evaluation of Localization Subsystem

As discussed in Section 6.8.6, for handling manifested FP issues, the localization subsystem was implemented in two layers with multi-level classification carried out at both the levels. At Layer-1, the model classifies the faults into one of several broad categories of FP issues as was shown in Table 6.8. We set up the baseline performance with OneR as shown in Table 6.16.

Table 6.16 Localization Performance

	Localization Level 1		Localization Level 2	
Parameter	**HYPERVINES (%)**	**ONE-R (%)**	**HYPERVINES (%)**	**ONE-R (%)**
Accuracy	97.03	86.14	96.04	90.10
Incorrectly classified classes	2.97	13.86	3.96	9.90
Precision	97.1	94.7	97.6	93.5
Recall	97.3	86.1	96.0	90.1
PRC Area	0.967	0.806	0.955	0.846

At Layer-1, we chose the sequential minimal optimization (SMO) multi-class support vector classifier. With SMO and RBF Kernel and parameters C = 12, gamma = 0.01 epsilon = 1×10-12, the accuracy of Layer-1 localization is 97%, which is a substantial improvement over the baseline performance of 86.14%. The performance of the model is given in Table 6.16.

Figure 6.17 gives a comparison of the performance of our Multi-Class Multi-Layer (HYPERVINES) model with the baseline. It can be seen that the accuracy of the classification of HYPERVINES is 97.03% against the baseline accuracy of 86.14%. A useful metric for comparison of classifiers is Precision-Recall Area (PRC Area), which gives the tradeoff between precision and recall. A high value indicates high precision (i.e., low false positives) and high recall (i.e., low false negatives). We can see that HYPERVINES Level 1 gives a high PRC Area of 0.967 compared to 0.806 of the baseline.

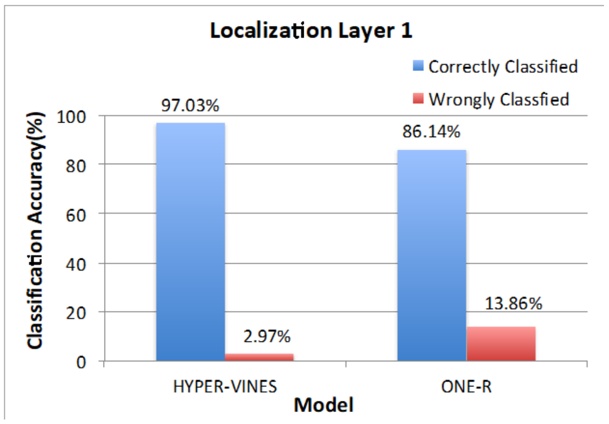


Figure 6.17 Localization Layer 1 effectiveness compared to baseline

Once a broad category has been identified, the Layer 2 model does fine grain localization for each category of manifested fault. In a dataset containing Network Performance Faults at Layer 1 and 5 different faults at Layer 2, we have the results in Table 6.16.

In the same table we can see that when compared with the baseline algorithm result, the HYPER-VINES using multi-class classification with SMO and OvO has a much superior performance, indicating the efficacy of the model. The localization accuracy of the model is 96.04% compared to 90.1% of baseline. The PRC Area of the HYPERVINES Level 2 classification is 0.955 against 0.846 of the baseline. Figure 6.18 gives the graphical comparison of Level 2 performance of the implemented model (Multi-class, Multi-layer) and the baseline. It is seen that the implemented model gives a higher percentage of correctly classified and lower percentage of wrongly classified examples.

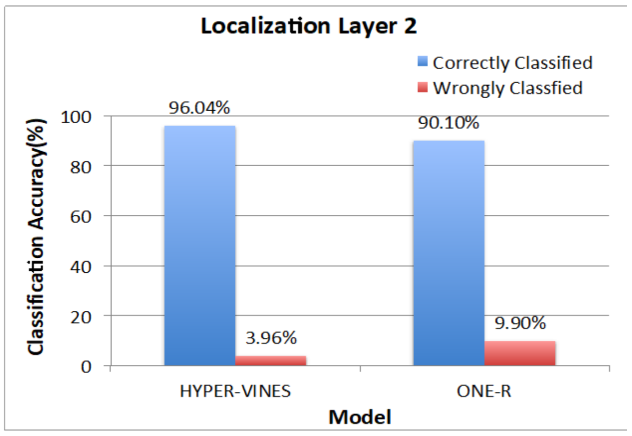
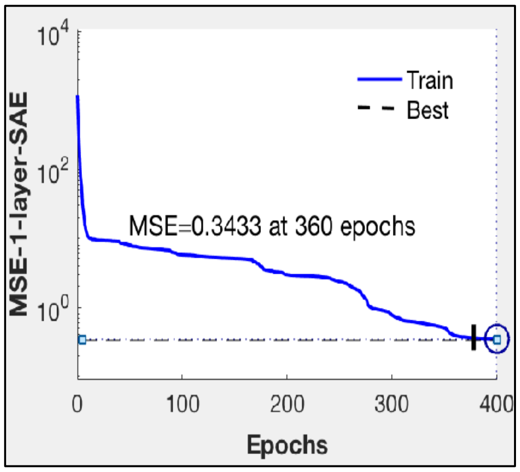


Figure 6.18 Localization Layer 2 effectiveness compared to baseline

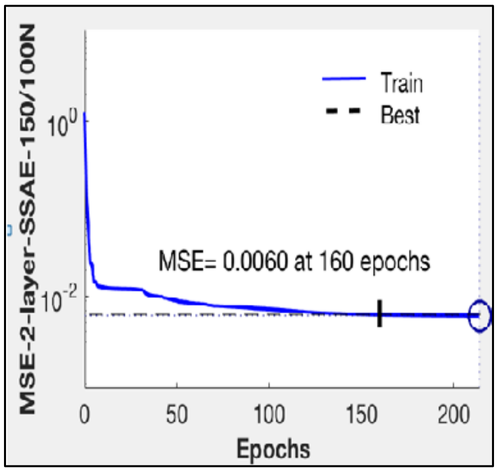
6.9.4 Localization of Impending FP Issues

One of the main concerns handled in the framework is to localize impending faults and predict their severity levels. We have seen in Section 6.5 that the localization sub-system uses stacked sparse autoencoder (SSAE) for faults detected as impending faults. While at the preprocessing stage a total of 353 features were selected, further condensation was left to the SSAE used.

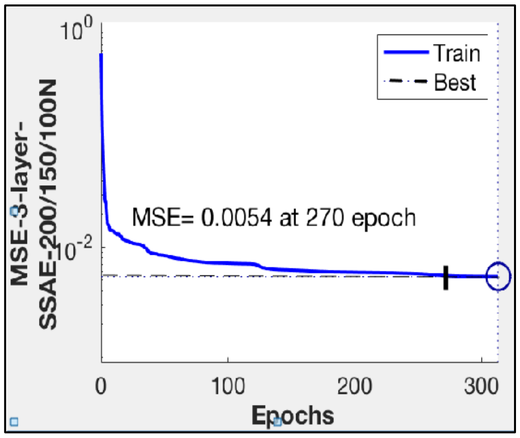
To make the deep learning model predict with high accuracy, the first step is to train the stacked autoencoders such that the output is as close a replica of the input as possible. To achieve optimum performance, the stacked autoencoder parameters like the number of hidden layers, the size of the layers, sparsity regulation (SR) and sparsity proportion need to be judiciously arrived at. An example of comparative reconstruction performance is given in Figures 6.19(a) through 6.19(d). It is seen that the model with 3 hidden layers of 200/150/100 neurons, respectively, converges quite fast to a low mean-square error. Reconstruction accuracy is important as it affects the prediction based on the trained encoders, which the model is eventually used for [169].



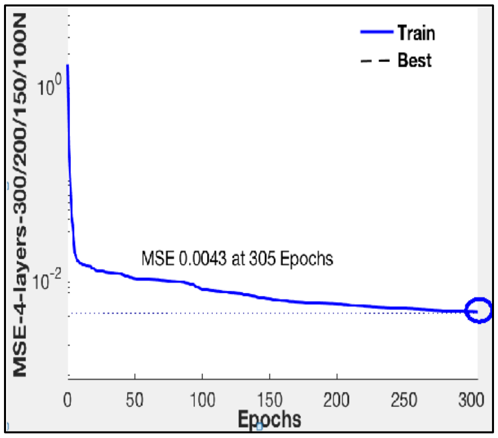
(a) Single AE



(b) 2-layer SSAE



(c) 3-layer SSAE



(d) 4-layer SSAE

Figure 6.19 Mean square error for reconstruction of the input

Sparsity in data is handled by using the Autoencoder parameters sparsity regulation (SR) and sparsity proportion (SP). SP gives the proportion of training examples a neuron reacts to. A low value of SP encourages sparsity.

Having achieved good reconstruction results with stacked autoencoders, the model was tested for prediction of the severity of impending fault and performance issues. As discussed in Section 6.8.6, a Softmax layer is added as a prediction layer. The graph in Figure 6.20 shows that the model has good generalization characteristics as MSE for the test dataset is close to that of the training dataset.

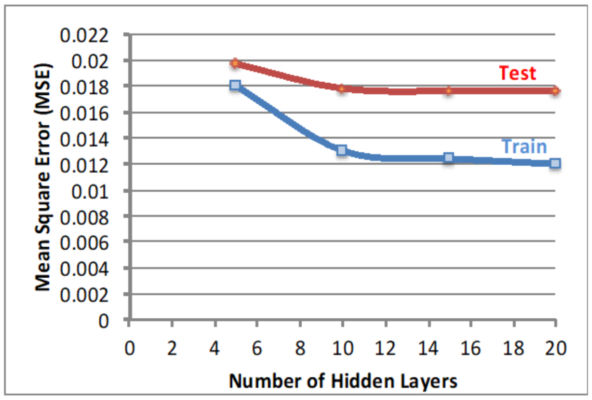


Figure 6.20 MSE in training and test dataset

Fine tuning of the model was done using backpropagation. The accuracy ranges between 72 and 85% with the abridged dataset (~1000 examples) and ~92% with the enhanced dataset (~5000 examples). Experiments were carried out for SR = 1 and SP = 0.4.

We baselined the above results with those obtained with a shallow model, viz., SVM with RBF kernel which worked very well for detection and could only obtain 73.1% accuracy in localizing impending faults. A comparison between SSAE and SVM models is shown in Figure 6.21. The deep structure thus provided a substantial improvement in terms of accuracy of prediction of the severity levels of the impending faults.

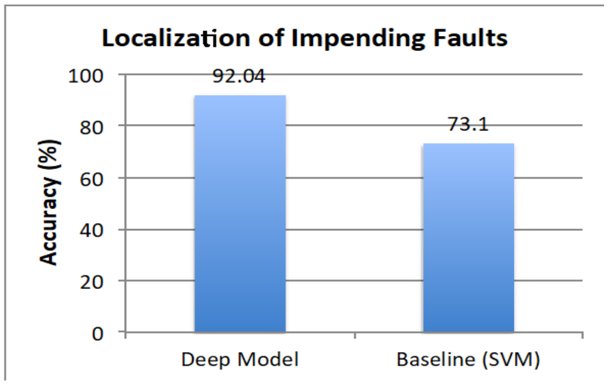


Figure 6.21 Localization of impending fault stacked autoencoder and SVM (baseline)

6.10 Summary

This work introduces the issue of availability and performance management of carrier services using NFV over a multi-cloud in a way that achieves the goals established in Subsections 6.1 and 6.2. To recapitulate briefly, the following contributions were planned: a) Discuss the architecture, creation and management of VNS, b) Elucidate the FP problem, c) Discuss usefulness of AI techniques in the cloud-based NFV environments, d) Describe the AI based FP management framework and e) Evaluate the designed framework and discuss results. To enable a holistic understanding of the fault and performance issues, we have described the design of VNSs like mobile or broadband services using SFCs. All the management platforms (MANO, OSS/BSS, and MMCP) that play important roles in fault and performance management of VNSs and their interactions have been discussed. MANO is the main component of NFV life cycle and fault management and considering its importance its constitution and functions have been discussed in detail. Responsibilities of each of the sub-systems of the MANO towards monitoring and management of fault and performance issues have been described. Interfaces that have been defined between the MANO and the multi-cloud manager (MMCP) and between the MANO and the OSS have been discussed. All these aspects cover goal a). Towards achieving goal b), a full section has been devoted to the description of the fault and performance issues wherein we also discuss the criticality of faults and the shared FP responsibilities of the management platforms. To meet goal c), explanation has been given for the importance of considering AI for achieving the goals of the FP problem. Towards achieving goal d), a generic framework for detection and localization of the FP issues has been proposed and described in detail. It has been brought out how the AI based framework would be able to go beyond the traditional models in predicting impending failures and their severity. Markers and metrics are important ingredients of any FP management system and have been given a fitting treatment. To accomplish goal e) we have discussed the results our work involving the implementation and evaluation of the detection and localization functionalities using the machine and deep learning respectively. Using an actual network fault data, we have shown how manifested and impending FP issues can be effectively handled by the detection and localization sub-systems of the FP management framework based on machine and deep learning models.

Chapter 7

Security in Next Generation Healthcare

So far we have discussed the work that we have done to place VNSs over clouds and ensure that they perform well. In using cloud infrastructure and virtual network services critical services like healthcare, security of patient data becomes an important requirement. In this chapter we discuss the threat model and the security architecture we have evolved for Internet of Thinks (IoT) based next generation healthcare conceived to be deployed in the virtualized network service and multi-cloud environment. It is clear from our study that we are dealing with unknown threats and adversaries, which limit the use of signature based detection methods. Any attack on the data in motion between domains could be a serious threat to life or well being of the patients being diagnosed, treated or transferred to a medical facility on an ambulance. We carry forward the study to the method for mitigation of attacks with special emphasis on protection of data as it moves among domains for analytics, storage or visualization. The overall aim is to develop an efficient anomaly based multi-level deep learning based intrusion detection system, which protects the data in motion, among the domains and within cloud hierarchy, from any effects of malicious attacks leading to mutilation of data flows that could lead to device resetting or capture.

We have worked on an innovative hierarchical deep learning solution that works at different levels in the clouds and protects patient data from mutilation and pilferage. Any undesirable activity on data could lead to incorrect diagnosis or patient device malfunction, putting their lives and health to risk. The contents of this chapter draw from our work on QNRF funded project ID NPRP10-0125-170250 and our paper titled, "Merged Hierarchical Model with Layer Reuse," (under preparation) for Security in Multi-cloud.

7.1 Introduction and Motivation

Among the problems of the present healthcare systems are the inadequate response to medical emergencies, delays in diagnosis of acute cases, insufficient monitoring of chronic patients, readmissions and above all too many preventable errors. Preventable errors are the third leading cause of death in the USA [185]. To reduce patient morbidity and mortality with timely and more accurate diagnostics, medical care of patients is increasingly relying on technology. The future technology directions include Internet of Things (IoT), multi-cloud systems and virtualization of network services. These developments have the potential of saving crucial minutes in the diagnosis and treatment of critical, hospitalized or ambulance bound patients, which could mean the difference between life and death or severe disability and return to health.

Cost of providing healthcare is another reason for taking recourse to the advanced technology. The US alone spends 3.5 trillion dollars annually amounting to a staggering 17.5% of the GDP [186]. Increasing use of medical IoT for monitoring vitals and other parameters, delivering medication and collecting patients' data records for current and future use is already making its presence felt with the average number of devices per bed in US reaching 13 with 4 out of these already connected to the network [187]. As a consequence of rapid upsurge in the use of IoT devices, analytics and storage is being increasingly outsourced to one or more clouds [188]. As virtual network services go mainstream, the triad of VNS, IoT and multi-cloud is expected to substantially reduce cost and improve outcomes of healthcare [189] [190].

Notwithstanding the expected gains, fast and better diagnostics, from the direction that the technology is taking in healthcare, the undesirable fallout is the increase in the attack surface of the healthcare network, making patients and hospitals susceptible to crippling malicious activities. Available data show that 94% of the US hospitals have been target of attacks with 63 percent of healthcare breaches having been caused by criminal or malicious activity [191]. The worst concerns in the recent years have been ransomware with several high impact cases involving WannaCry, Petya/NotPetya and SamSam [192]. Intrusive attacks lead to serious threats on data e.g., patient data inaccessibility, pilfering and mutilation and attacks on devices e.g., altering settings of or capture of patient devices. In a striking example of threat to critical medical devices, the U.S. Food and Drug Administration (FDA) recalled half a million pacemakers due to the firmware having vulnerabilities that could allow a hacker access to the device and let them manipulate heart rate settings and battery power [193].

The increasing attack surface with the increasing sophistication of technology and failure of the current intrusion detection systems in the evolving technical milieu, provide us with the motivation for a cloud based hierarchical model that would be able to detect any kind of intrusion, from within or outside the organization, that alters patient data in any way.

Attack and anomaly detections must consider hundreds of indicators and interaction patterns across thousands of sensors and servers on a continuous basis. This voluminous data is multidimensional and complex with patterns that do not become evident with traditional analysis. The meta-information about the flow of data assumes as much importance as the payload for examination to effectively weed out the anomalies. The proposed hierarchy of distributed deep learners examines the data being transferred to and from the clouds to identify malicious communications, data generated by malwares and any other attacks that results in changing of the value of the parameters being transferred as well as the integrity of the data flow. Such distributed deep learners grow in size and complexity as we move from the IoT domain edge through the edge cloud to the public cloud and the challenge to reduce the training time and maintaining accuracy remains. In our research we have worked on both of thes aspects.

7.2 Contributions

Towards tackling the challenges of security of the data in motion in the IoT, cloud based and VNS based healthcare, we make the following contributions:

a) Establish the reference architecture for critical healthcare application like ambulance bound critical patients.

b) Based on the attack surface presented by the flow of data in the multi-domain next-generation healthcare architecture, evolve a threat model defining clearly the threats and their mitigations.

c) Propose a novel method based on hierarchical deep neural networks to protect the patient data flowing between the IoT and edge cloud and edge clouds and public cloud. The mainstay of this proposal is a novel artificial intelligence (AI) technique involving a hierarchy of deep learners for hardening the security of healthcare system infrastructures.

d) Evolve a merged public cloud deep neural network for improvement in training time and accuracy of prediction of anomalous activity in the data flows.

e) Evaluate the proposed method and discuss the results.

7.3 The Conceptual Framework of the Next Generation Healthcare

The generic layout of the next generation healthcare service assists in understanding the flow of data and provides useful inputs for evolving the system architecture. This architecture is then used to prepare a threat model and carry out mitigation planning. We depict the overall layout of the next generation healthcare service in Figure 7.1.

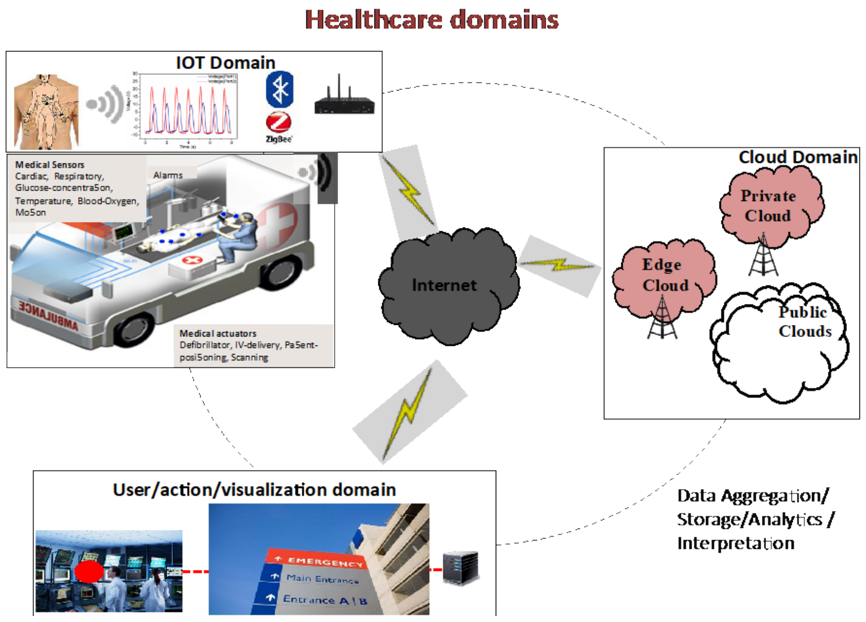

Healthcare domains
IOT Domain
ZigBee
Medical Sensors
Cardiac, Respiratory, Glucose-concentra5on, Temperature, Blood-Oxygen, Mo5on
Alarms
AMBULANCE
Medical actuators
Defibrillator, IV-delivery, Pa5ent-posi5oning, Scanning
Internet
Cloud Domain
Private Cloud
Edge Cloud
Public Clouds
User/action/visualization domain
EMERGENCY
Main Entrance
Entrance A | B
Data Aggregation/ Storage/Analytics / Interpretation

Figure 7.1 Overall layout of the IoT-Cloud based healthcare

7.3.1 The Healthcare Network Domains

As can be observed from Figure 7.1, the healthcare network consists of three domains – the IoT domain, the multi-cloud domain and the visualization domain. These domains act as source, sink or storage and analytics nodes. The IoT domain is generally a source producing large volumes of multi-dimensional patient data but also acts as sink for the commands sent to the actuators. The multi-cloud domain provides a good combination of speed, analytical sophistication and storage. The visualization domain is predominantly a sink that consumes the analyzed data in many forms. However, the clinical staff can generate small amounts of data in the form of commands, instructions or prescriptions. Given here is the brief description of the three domains:

The IoT Domain: The IoT domain consists of a heterogeneous mix of wired and wireless, wearable or implanted biosensors, actuators and other medical devices, for patient data acquisition and delivery of treatment. Each of the sensing devices performs simple tasks like monitoring pulse rate, oxygen saturation or blood pressure. Actuators, on the other hand, perform the task of delivering treatment like activating oxygen flow or injecting insulin, based on commands from medical professional or other devices. In the case of ambulance bound patients, as a result of monitoring by IoT devices, there also are auto responses in terms of alerts and suggestions to paramedic staff inside the ambulance.

IoT medical devices measure body temperature, blood pressure, heart rate, oxygen level, and other contextual information of the monitored patient and provide a real time flow of vital biofeedback. Much of the data generated by devices is communicated to other devices. For instance, in the ambulance, pulse oxymeter may start oxygen administration to the patient. Characteristically, IoT devices present specific challenges due to their resource constraints. In particular, they are characterized by small batteries, small amount of memory and low power CPUs. The majority of medical IoT devices are based on cheap micro-controllers that run a small amount of preloaded code. These devices have varying processing, storage and communication capabilities and are, in general, constrained. As a consequence, these devices depend on external storage, processing and analytics provided by the multi-cloud domain. Most of the devices connect to an IoT gateway, which acts as the interface between devices and the rest of the network. It does the task of protocol or data conversion, when required. Some of the devices may connect directly to the service providers wired or wireless virtual network services. This scenario is challenging, as despite being constrained they are required to transmit large amount of information and at the same time be fault tolerant and robust to security attacks. To complicate the situation further, the variety of communication protocols used in healthcare IoT is large and yet interoperability among the devices is a necessity. The majority of the devices use the ISM (Industrial, Scientific and Medical) frequency band (e.g., the 2.4 GHz and 5.8 GHz band). Since this band is unlicensed, the manufacturers of medical devices implement support for their own communication protocols. This results a heterogeneous scenario constituted by different modulation schemes, protocols, and hardware that represent a challenge from the security perspective. All these complexities lead to the attack surface becoming very wide, and a malicious actor can exploit any of the vulnerabilities of the system.

The Cloud Domain: in our work the cloud domain has been assumed to consist of the hierarchy of edge and public clouds as was shown in Figure 7.1. The reason for working with a hierarchy, rather than just the edge or just the public cloud can be understood like this: taking all the data from the widely spread out IoT domain, directly to the public cloud would require expensive connectivity and would increase access latency. On the other hand, keeping all the data in the edge cloud alone improves latency and connectivity cost but reduces the storage capacity and analytic sophistication. A multi-cloud infrastructure provides a combination of low latency, vast storage, optimized bandwidth cost and high analytical sophistication. While the edge cloud aids quick diagnosis in emergency and acute cases, computationally demanding inference algorithms in the larger public clouds help in determining patterns in historical data that correlate with current symptoms. This helps in differential diagnosis or discovering yet to develop ailments [8]. Finally the cloud hierarchy, allows us to think of a distributed security solution, with increasing sophistication from IoT gateway to the public cloud gateway.

The Visualization Domain: consists of mechanisms for presenting multiple streams of processed data to the concerned clinical staff. The information can be presented in graphics, tabular and other forms, to assist the doctor make a fast and accurate diagnosis. This domain also allows the medical staff to choose from several streams of incoming information. In the case of data from an ambulance, the doctor may decide to communicate with the paramedics to provide guidance for immediate patient care. The basic idea is to summarize a large amount of information – historical and current, for the clinical staff to quickly glance through, with a view to get a fairly good idea of the health of a patient, catch the exceptions and anomalies and the early signs of developing complications [194]. It is understood that improvement in visualizations can improve diagnosis and in turn outcomes of the treatments.

7.3.2 Design of Architecture for Next Generation Healthcare

From the viewpoint of this work the architecture is required to be so designed as to take care of the security requirements while ensuring that it meets the functional requirements. The functional aim of the design is to provide comprehensive data to the medical experts as well as applications that process them or controls devices used for patient care. Improperly designed system architecture can severely constrain the system's functionality of data movement among domains and its amenability to effective security. As the doctor's are mobile and they are increasingly using their own mobile devices like tablets and smartphones, it is important to take this mobility while evaluating the threat surface. A number of other requirements like image acquisition, analytics and communication, remote patient consulting and monitoring and telemedicine would govern movement of large amount of data from the patient or the IoT domain to the cloud domain for analytics and storage. This voluminous data has an impact on how the network is organized, where the data is taken for storage and analytics, the way the data will move among domains and how devices, applications and medical experts will collaborate for care of the patient. We have tried to make the architecture general so that it retains its flexibility and scalability to grow as the demand on the system grows. In general, it is advisable to use open architecture standards that would ensure compatibility of domains, avoid vendor lock-in, ensure widely accepted standards and reduce cost of the eventual system [195].

The architecture that we have evolved is shown in Figure 7.2. In consistency with the layout already discussed in Section 7.3.1, the architecture has been divided into four inter-linked zones: the sensor and actuator zone, edge cloud zone, public cloud zone and the visualization zone. This compartmentalization into zones assists in design of the security policy for the healthcare network. The organization of zones is in consonance with the flow of data that among the domains of the network. A brief discussion on the important constituent of various zones is given below:

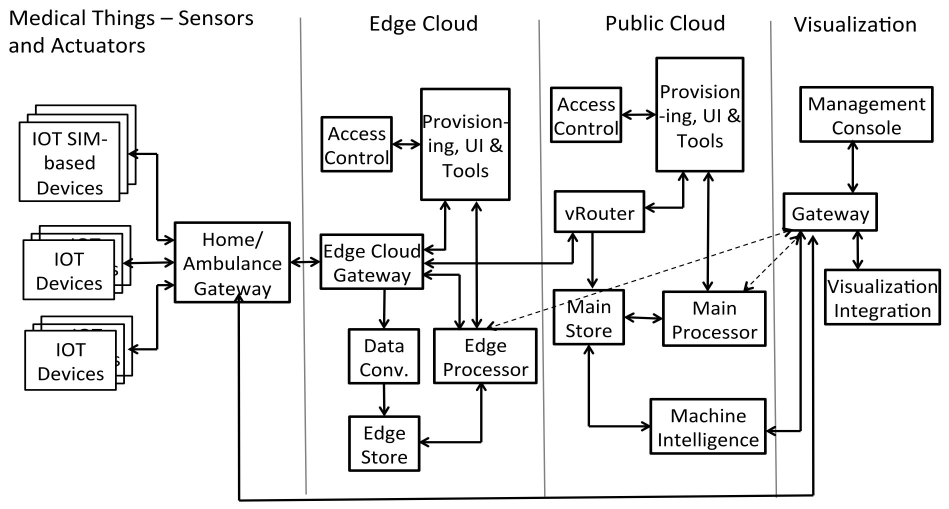


Figure 7.2 Architecture of the IoT-Cloud healthcare system

Medical Things and Hospital/Home/Ambulance Gateway: This is equivalent to the IoT gateway in the layout diagram of Figure 7.1. This zone is composed of devices like sensors, actuators, monitors and gateways. It has been mentioned before that most IoT devices have limited communication, compute, storage and power capabilities. These devices may be vested with the capability to discover other devices and mesh with them. They rely on their connectivity with the IoT gateway (designated here as home/ambulance gateway) for many functions like protocol and data conversion, and connectivity to cloud gateways for communication with the cloud domain. The gateways connect to the carrier's virtual or physical wide area network and can register with the edge clouds but also in some cases directly with the public cloud. Some of the devices that are mobile SIM based can also connect to the communication network and can securely register with the cloud for sending and receiving data. The gateways, or intelligent edge devices as they may be called, serve an active role in managing access and information flow. They may assist in device provisioning, data filtering, batching and aggregation, buffering of data, protocol translation, event rules processing, and more. Some of the devices in this zone are involved in image or video acquisition and would give rise to large amount of data to be sent to the gateway or the edge cloud for immediate analysis or the public cloud for more sophisticated analytics and storage.

The Edge and Public Cloud Gateways*:* in the edge and public cloud zone, respectively, provide connectivity to various types of devices within the zone and to the gateways in the other zones. They also manage devices and securely accept data. The cloud gateways provide secure connection to the cloud, data conversion, telemetry information, event ingestion and management of connected devices in its zone.

The Edge and Main Processors: work on the streams of data from the devices or the gateway. The edge processor works on the streams of data from the Medical Things or IoT domain, and either interfaces the streams to the cloud zones or directly with the visualization zone. The edge processor has reasonable processing capabilities and is able to run small neural network models. The edge store is primarily for temporary storage to keep the patients data while the ambulance is passing through the corresponding cell or while it is needed for analytics. The main processor can carry out more sophisticated analytics based on large neural network models using both current data and large historical datasets. It has the capability of interfacing multiple streams of heterogeneous information with the visualization zone. If can also add new data and its connections to the already stored the information.

Machine Learning Agent: in addition to the main processor, the public cloud zone may provide specialized hardware (e.g. GPUs) and software that can process historical data for intelligence like developing diseases or for predicting readmissions.

Visualization: Visualization domain consists of related tools used to visualize patient data and facilitate patient diagnosis, monitoring and management. It consolidates and synthesizes large volumes of data from multiple sources to provide key insights to the clinical staff. It makes patterns and relationships evident in large volumes of data, which are not discernable in raw data or reports [196].

7.3.3 Security Architecture for the Next Generation Healthcare

Unlike traditional healthcare network architectures where security is overlaid and involves manual provisioning, in next generation systems security is integrated into the healthcare and network architecture right from the design stage. This enables consistent security system across all domains/zones and leads to excellent end-to-end security. This not only increases the complexity of providing security but also the time and cost of such maneuvers [197]. Our aim here is to show the basic security requirements in such a system and specifically discuss the innovative methods that we have evolved for detection of malicious activity while the data flows across the zones. Proper integration of the security aspects into the architecture will not only mean high availability, patient safety, improved ambulance bound patient care but also ensure that the system, whether used for domiciliary or patients in ambulances, conforms to any regulatory and accountability compliances expected by law of the land.

In the real-world systems the data needs to be protected right from the point of generation to the clouds and onward to the visualization points. In the architecture that we have envisaged, this would mean hardening of all the zones with appropriate technologies that go beyond the state-of-the-art and thwart the attempts of adversarial actors in infiltrating the system. We describe here possible end-to-end protection techniques and indicate the ones that we are concerned with in this work. This puts the work done in this dissertation in right perspective as part of a bigger system.

Device Protection and D2D Communication Protection: This concerns protection of both the devices and device-to-device communication. It turns out that physical layer security provides efficient and effective solutions to enforce confidentiality in D2D communications. In particular, new cryptographic key-establishment protocols leveraging different radio signal features such as received signal strength and artificial noise generation will help strengthen the inter-device communication. Focus is also on denial of service robustness and in particular on jamming by considering it both as a threat and as a defense strategy for scenarios in which unauthorized radio communications should be prevented at the radio level. New solutions need to be developed for protecting the device against compromise, tampering, mutilation or theft of data being sensed and code attestation during the devices' updating process. Access to these devices needs to be strongly regulated and the device integrity needs to be tested frequently.

User Authentication*:* Patients, paramedics in the ambulances, hospital medical and support staff have all to be authenticated in a secure and robust way to protect the patient and their private data. It is proposed to study, analyze and develop novel user authentication techniques based on several biometrics such as iris, face, fingerprints and ECG. For patients in the ambulance, brain wave biometrics can be explored. Cryptographic authentication and authorization techniques would provide a high level of system access security.

Data at Rest in the Cloud: Gateways are the interfaces between the devices and the cloud system and between levels of clouds in the hierarchy. The cloud system provides a range of services from quick processing to specialized analytics to aid diagnostics and discover hidden trends. Individual patient's historical data can be used with current observations to detect any acute conditions and make treatment recommendations. Data from patients from a community can be analyzed to predict impending epidemics. Confidentiality of data stored at various points in the network can be protected through innovative encryption and encryption key management. Integrity can be taken care of by using innovative hashing algorithms.

Data in Motion: This is the part of interest for this dissertation. Data from the physiological sensors are collected by the ambulance/home gateway and transmitted over the Internet or wireless access network to the edge cloud. Low latency tasks would usually be performed here and for deeper analysis the data would be transferred to the public cloud. Processed information is sent to doctor’s screens to visualize the exceptions and make an accurate diagnosis. Our work on hierarchical deep learning is aimed at protecting the data in motion from any attack that threatens to affect the meta-information or payload of the dataflow. The main concern here is to use deep learning to find anomalies in inter-domain streams of data for any indication of malicious intent. The innovativeness of the method lies in its hierarchical implementation and use of merged edge models to create larger public models.

The security architecture that we have envisaged in the next generation network is represented in Figure 7.3 in a simplified form.

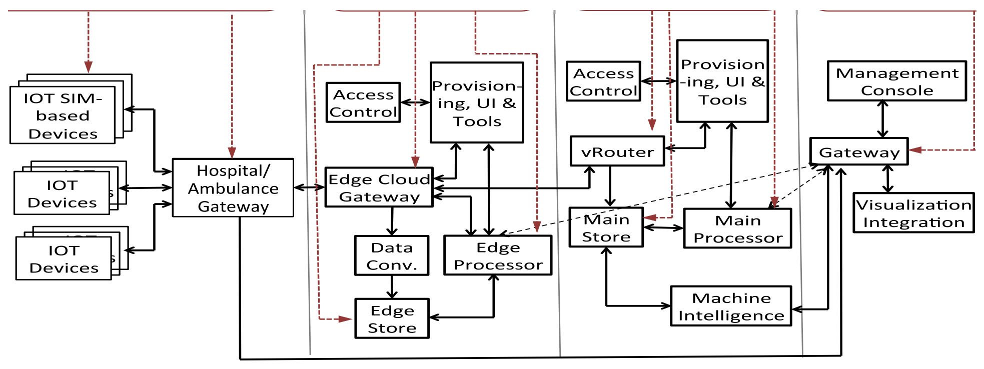


Figure 7.3 The security architecture of the next generation healthcare

In this work we have focused on the development of a hierarchical deep learning based anomaly detection system that will indicate if the dataflow between the home/ambulance gateway and the edge cloud and between the edge cloud and the public cloud is compromised.

7.4 Threat Model for the Healthcare Network

We describe in this section a threat model for the healthcare system that has been characterized above. A threat model is essential to decide on approach that we take to securitizing such a system. The threat model essentially provides an understanding of the attackers, attacks and mitigation. It has helped us integrate the essential features of the security policy while design the system. We describe in this section a threat model for the healthcare system that has been characterized above. A threat model is essential to decide on approach that we take to securitizing such a system [198].

The compartmentalization or zoning of the healthcare system that has been the distinctive feature of our architecture has helped us formulate an appropriate threat model. The zones are represented by their gateways. Thus we have the IoT gateway, the edge-cloud gateway and the public cloud gateway and associated services. A trust boundary separates each zone from the other connected zones. It represents a transition of data/information from one source to another. It is important that the data flow crossing the trust boundaries be protected against various kinds of attacks [199]. A diagrammatic representation of the zones and trust boundaries in shown in Figure 7.4

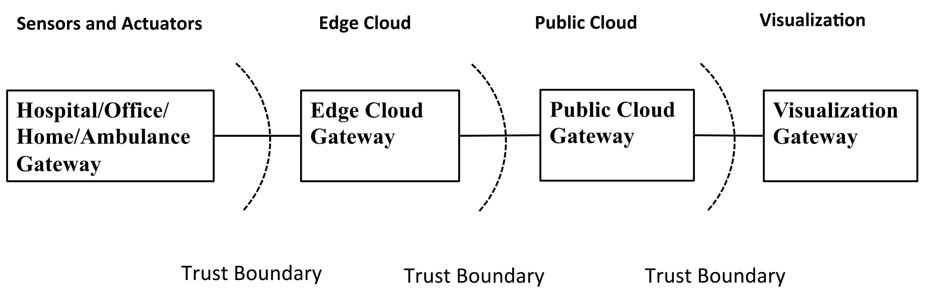


Figure 7.4 Trust boundaries between zones

7.4.1 Attack Surfaces

The attack surfaces at various gateways are described below:

The Hospital/Home/Ambulance Gateway Attack Surface: we can see from Figure 7.4 that this gateway is at the boundary of sensors and actuators zone. It combines the functionalities of a router, a processor, a protocol and data convertor, a device controller, a data flow manager and a sessions manager. It acts as an interface between the devices and the cloud domain carrying out the necessary interface activities. It enables communication between heterogamous devices and the cloud domains, which might be operating on different protocols. The gateway also routes traffic to the correct edge cloud. The sensor and actuator gateway zone identified for threat modeling is essentially the IoT domain shown in the architecture in Figure 7.1. We will use these two terms interchangeably. There are two attack surfaces associated with this gateway. One facing the IoT devices and the other facing the edge cloud zone. These gateways are often targets for intrusions. These intrusions can be serious as there is no redundancy built into these gateways. The data sent by the IoT gateway to the edge cloud could be tampered with or be made non-available.

The Edge-cloud Gateway Attack Surface: the edge cloud gateways would be physical or virtual server based systems that accept flows from one or more IoT gateways (in some cases directly from the devices) and regulate access to the facilities of control, analysis and storage in the edge clouds. Physical access to an edge-cloud is generally more regulated than the IoT Gateway. The IoT gateway would be in ambulances, hospitals, offices and homes while a mobile edge cloud gateway would be on top of the mobile tower. All the external communication with the edge cloud takes place through the edge cloud gateway. In the set up envisaged by us, the edge cloud is exposed to the public cloud. Most communication to the public cloud is also through the edge gateway. The gateway has two surface areas, one facing towards the IoT gateways and the other towards the public cloud gateway. Both edge cloud and public cloud have interconnection with the Internet making the zones vulnerable.

The Public Cloud Gateway Attack Surface: This is usually a virtualized gateway that provides access to the public cloud zone. It controls access from the home/ambulance zones coming directly or through an edge cloud gateway. It also connects virtualization domain and other external entities of the healthcare system like insurance agencies, pharmacies, suppliers and billing services. The attack surface is much larger facing the external entities using the public cloud.

Attackers: These are all internal or external malicious agents who attempt to mutilate or alter the flow of data in any way. Threat could be from organized crime, nation states, hacktivists, business associates, skiddies and malicious insiders. In this threat model we assume that adversaries have unlimited resources and time while the targetted organization has limited resources and thus need efficient methods to counter attacks [200].

Attacks: Attacks targeting confidentiality, integrity and availability (CIA) are major security issues for cloud based IoT. There are several vulnerabilities that could be maliciously exploited by adversaries in many forms. Attackers can perpetrate a variety of attacks like denial of service to prevent patients' access to devices, changing device settings or stealing patient records for frauds like undergoing surgery or obtain prescription medicine. More serious attacks include capturing patient devices, ransomware and advanced persistent threat. The attackers also referred to as cyber criminals, evade detection by using multiple tactics, tools and targets in their attacks. Their techniques can change temporally and spatially. The sensor and cloud based healthcare technology is promising and its benefits far outweigh the risk of data breaches and potential malfunctions.

It is important to recognize the kind of attacks that we are protecting the system against. While in our system we are concerned with any intrusion that alters the dataflow in terms of the meta-information is of concern. This will happen if the dataflow is disrupted in any way. It would still be useful to look at these attacks, preferably using a standard attack model for the IoT-cloud bases system like the STRIDE model [199]. This model describes the following categories of attacks:

Tampering: tampering is a serious infringement to the healthcare system in which the attacker alters the dataflow and thereby changes the values of the patients' bio-markers or the meta-information the data flow (e.g., packet flow rate).

Spoofing: This is a man in the middle attack where the intruder fools the source to believe that it is a legitimate destination. Spoofing attacks can help attackers to cross the trust boundaries and cause data theft and deletion. If the masquerader intercepts, partially or fully modifies the dataflow then this is of concern to our system.

Denial of Service: Constrained devices are generally under DoS threat when they actively listen for inbound connections or unsolicited datagrams on a network. An attacker can open many connections in parallel and not service them or service them slowly, or the device can be flooded with unsolicited traffic. In both cases, the device can effectively be rendered inoperable on the network.

Information Disclosure: constrained devices will have simple security like a single PIN or password. Sometimes they just trust the network and allow access to devices on the same network. Active reconnaissance may allow attacker to obtain information about the target and then cause remote attacks. SQL injection can cause an information disclosure attack as it can obtain information about the data in the system.

Elevation of Privilege: A user with limited privilege assumes the identity of the privileged user and gets administrative privileges. With higher privileges attacker can cause an exploit attack. It can access gateways and change information flows.

Repudiation: Attacker can log data to wrong files or change data in the name of others.

Mitigation: our concern is to protect the data in motion from attacks that can cause any in changes in the meta-information of the dataflows. We need to take care of both seen and unseen attacks.

7.5 Merged Hierarchical Security for Data in Motion

Having discussed the threat model and evolved the security architecture, we now discuss the methods that we have evolved for security of data in motion. We consider the scenario where all heterogeneous data collected from a variety of medical sensors and other measuring devices are aggregated at the IoT gateway and transmitted to the edge cloud gateway. This does not cause any loss of generality as the model allows data to be directly sent to the public cloud gateway for storage or analytics. As our experimental cloud system is a hierarchy consisting of edge and public clouds, we have the opportunity of considering a part of that data moving from edge cloud to the public cloud. The data in motion could be encrypted and still be affected by various attacks. Our effort in this work has been to develop a proactive, distributed and hierarchical deep learning solution to protect the data flows from any adversarial attack that mutilates or alters the data stream, including its meta-information, in any way [201] [202].

7.5.1 Design Aspects of the Security System

Based on the recommendations of IETF Intrusion Detection Working Group, an IDS design should be based on four types of functionalities – information acquisition about the target system, storing information about the events, analyzing event information to detect hostile behavior and create response [203]. As far as their functionality is concerned, they are expected to continuously monitor and report intrusions. Performance wise they should give low false positives [204].

IDSs built on these principles detect hostile activities or exploits, in a network, that can lead to the data flows being compromised. They detect and react to isolated, distributed as well as coordinated attacks. IDSs are classified based on the information about the attacked data flows used for detecting attacks. Thus we have signature or misuse based IDS, anomaly based IDS and those that use a combination of these two techniques. In the signature based methods information about the known attacks is captured and stored for future use. If any flow of data matches with any of the stored attack patterns then presumption of an intrusive attack is made. In anomaly methods, behavior of the normal flow of data is captured and any deviation from this is considered as an attack. Another factor that needed to be decided was the location of deployment of the IDS. A centralized deployment e.g., in the IoT gateway can only detect mutilation or traffic tampering passing through the IoT gateway, the traffic heading from the edge to the public cloud domain cannot be checked for malicious activity.

In Chapter 3 we discussed the problems that the traditional IDSs face in the cloud set-up. Considering that our system would continually face unknown attacks, we have worked on a system that detects anomalies in the data flow that does not conform to normal flow. As regards the actual placement of the neural network based solution, we have a distributed solution that works at all the gateways – IoT, edge cloud, public cloud and visualization domain. In this work we focus on the data flowing from the IoT gateway to the edge cloud and from the edge cloud to the public cloud. The principle remains the same for any other flow of data through the healthcare network.

7.5.2 The Choice of Deep Learning System

In our target IoT-Cloud-VNS architecture, any IDS detecting anomalies must consider hundreds of indicators and interaction patterns across thousands of sensors and servers on a continuous basis. The data produced by such a system is multidimensional, voluminous and complex with patterns that are not evident with traditional analysis. Because of the ever changing threat environment and inability of the traditional systems in tackling them, these systems are giving way to machine learning based anomaly detection. Machine learning has been applied to security in healthcare in many forms. Some of the classifiers that have been used are: support vector machines, decision trees, naïve Bayes, K-nearest neighbors and random forest. In machine learning based intrusion detection system, for instance, the idea is to capture underlying statistical features of data and use them to detect any malicious attack [204].

Classical machine learning does not work well large number of features, the lack of automatic feature selection, lack of unlabeled data, large attack surface and the incapability of handling frequently changing tactics of the attackers. Machine learning works well when it is trained with examples of both the classes 'normal' and 'malicious' so that it can classify a new stream appropriately. It is difficult to get training data on new and unseen attacks. Training only on normal traffic would not suffice because of variability of data. The data in our system is of various kinds and may all be normal but difficult for the machine learning based intrusion detection system to handle. These methods also usually have relatively high false positive rates for detection [205], which causes risk of over medication or unnecessary procedures.

Deep learning has had mixed run in intrusion detection scenarios. Its successes in other areas have time and again prompted researchers to refine deep learning methods to be more accurate and produce low false positives. After all, deep learning performs better than machine learning in learning high-resolution images that can be categorized into hundreds of classes [98]. This has been demonstrated with a dataset of 1.2 million high-resolution images belonging to 1000 different classes. Speech recognition systems have been built using recurrent neural network (RNN) and the author's claim their system outperforms previously published methods achieving 16.0% error [14, 29]. The authors in [155] have used multimodal Deep Belief Network [DBN], a deep learning method, to effectively identify meaningful cancer subtypes from multi-platform cancer data. The authors in [60] have detected tens of thousands of disease-causing mutations, including those involved in cancers and spinal muscular atrophy using a deep learning based system. Deep learning thus has created renewed interest and there have been some recent reported works [215, [217]. The IoT environment and deep learning are very well matched as shown in Table 7.1.

Table 7.1 Suitability of deep learning for IoT healthcare

	Sensor based healthcare	**Deep Learning**
1	Generates large volume of data	Handles large volume of data
2	The data is mostly unlabeled and has temporal variations	Handles unlabeled and labeled data, including data in a time series form.
3	A large variety of sensors produce heteronomous data with many features	Extracts features from high-dimensional data
4	Data is complex	Works in layers and learns non-linear features and functions from complex data
5	Large number of features may cause overfitting	Effectively makes use of a large number of features
6	Hierarchy of gateways	Amenable to hierarchical deep learning
7	Of large number of sensors some may not be activated, drained or faulty	Effectively handles missing values as it works on characteristics of the flow
8	Data is sparse	Has capability of handling sparse data

Our work consists of a network of deep learners for protection of data in motion. In these networks as the data is passed through the layers of the deep structure of the networks, the anomalies caused by any kind of attacks would be detected. Our choice of deep learning was dictated by a number of factors. Table VII shows the complementarity of deep learning and sensor based healthcare. Deep learning has been shown to exhibits potentially revolutionary results in detecting first-seen malware. In real environment tests on publicly known databases of endpoints with advanced persistent threat (APT) malware, the detection rates of a deep learning solution were over 99.9% [206]. As these improvements seem to be consistent across a large variety of domains discussed above, we can view this as a motivation to use it in the medical computing field. Besides the advantage of learning huge amount of structure and extracting useful features from high dimensional data, deep learning can handle large models. It has the advantage of automatically discovering features that would give the best results. It makes use of many layers of non-linear information processing to select useful features and even fuse them to end up with a rich set of automatically selected features. This gives this class of methods great power to analyze and classify data in which features are related to the outcome in a complex way. In machine learning large number of features cause the problem of overfitting, which may lead to incorrect detection of attack patterns. It is resistant to small changes and can generalize from partial data, making it easy for the system to identify patterns from partial data [206]. A big advantage of deep learning is its ability to detect the effect of previously unseen attacks, unseen application protocols, unexpected structural anomalies in packets and flows and new variants of known threats.

7.5.3 Deep Neural Networks – Stacked Autoencoders

Among the deep learning it is proposed to use the stacked autoencoder for a number of reasons. A neural network mimics the way the human brain works to learn relationships in a given dataset. Thus Artificial Neural Network (ANN), learns by examples i.e., adjusts its weights as it sees more training examples, so that it can generate a correct output for the given input. An autoencoder (AE) is a 3-layer unsupervised neural network where outputs of each layer are directly connected to the input units of the next. With training an autoencoder learns to recreate the input as output. The data is reduced in dimension as it is encoded and expanded again to the same size by the decoding part of the autoencoder. This way an autoencoder is able to learn a compressed representation (called code) of the relationships in the dataset. The autoencoder is called sparse (SAE) when it has sparsity enforcer that restricts the code size required for reconstruction is restricted.

A stacked autoencoder (SSAE) is a neural network consisting of multiple layers of sparse autoencoders in which the outputs of each layer are fully connected to the inputs of the following layer. Please refer to Section 6.8.6 for an illustrative representation of the autoencoder.
We have experimented with stacked autoencoders and have found desirable outcomes with the kind of data that we encounter. A stacked autoencoder is a deep network of greater expressive power. It selects useful features automatically from high-dimensional data and filters information through the layers to achieve better accuracy. Autoencoder tends to learn features at several levels that together form a good representation of its input. In our impending fault localization problem, several situations leading to faults have overlapping features. The first layer of a stacked autoencoder tends to learn first-order features in the raw input, such as categories of faults. The second layer of a stacked autoencoder may learn second-order features corresponding to patterns in the first-order features e.g. frequency of occurrence of faults.

A Stacked autoencoder uses under-complete configuration of the neural network i.e., in which the encoding layers successively decrease in dimension. Being deep neural networks they can learn and model non-linear and complex relationships. They have the ability to learn hidden relationships in the data without imposing any fixed relationships in the data. The ANNs have the potential for high fault tolerance. When these networks are scaled across multiple machines and multiple servers, they are able to route around missing data or servers and nodes that can't communicate. The learnable parameters of the SSAE are the weights and biases that are calculated by the network while hyperparameters must be suitably set to obtain good results. These hyperparameters are: the code size, the number of layers, number of nodes per layer and loss function.

Healthcare systems produce a large amount of unlabeled data. Since a stacked autoencoder can work with voluminous unlabeled data and extract useful features, it can be used in the unsupervised form. The data with a large feature-set, passes through consecutive hidden layers of reducing dimensionality. The reduced and enriched feature set or code is in the deepest layer. This code is decoded by running it through the decoding layers of the autoencoder in reverse order and brought to the original dimensionality. The reconstruction error gives an indication of whether the data is normal or anomalous. Only data with normal instances are used to train the autoencoder. After training, the autoencoder will reconstruct normal data with low root mean square error, while failing to do so with anomalous data, which the autoencoder has not encountered.

For training of the stacked autoencoders, a data set is divided into a training set and one or more test sets. A training example is selected from the training set and then the values of output are checked for their quality of reconstruction. If the chosen indicator of error e.g., root mean square error, is below a threshold then the training concludes. The trained model is then tested with the test dataset, which was not used during training. It is common practice among the developers of neural network models to “cross-validate” the network on the test set periodically during training and to save the network weight configuration meeting one of two criteria: (1) the network with the minimum error in the training set or (2) the network with the minimum error in the test set. The latter technique is often used to prevent the network from overtraining because networks are prone to overfitting.

In the semi-supervised model the autoencoder layers are trained with unsupervised data. To avoid over-fitting, such data require sparse methods in feature selection and learning [207]. To achieve the sparse representation, we minimize the reconstruction error with a sparsity constraint. Backpropagation is used to improve weights such that reconstruction becomes better. This signal is a mathematical factor that adjusts the value of each weight in the neural network to reduce the difference between the predicted and known output. Cases are selected from the training set and are continually presented to the neural network until the overall error has been minimized. One pass through all of the training cases is considered an epoch of training. The overall duration of training is often expressed in terms of the number of epochs required to reach an error minimum [208].

7.5.4 Hierarchical Merged Model with Layer Reuse

The aim of the new model is to provide adequate security for data in motion from IoT domain to the edge clouds and from the edge clouds to the public cloud. In the security architecture, discussed in Section 7.3, this would mean specifically focusing on the perimeters of the edge clouds and the public cloud. Based on our threat model, we would focus on adversarial intrusions that result in any change in the data flows: exploits, active reconnaissance, sequel injection, denial of service, tampering, device resetting and takeover. The main concern here is to use deep learning to find anomalies in inter-domain streams of data for any indication of malicious intent [209]. In the unsupervised mode, deviation based anomaly detection is mainly based on spectral anomaly detection, which uses reconstruction errors as anomaly scores. Reconstruction error of a data point, which is the error between the original data point and its low dimensional reconstruction, is used as an anomaly score to detect anomalies. In such models the loss is usually represented as the residual sum of squares and error of reconstruction is represented as the mean square or root mean square error. The main objective then becomes reduction of the loss function or the mean square error by training the model adequately in the forward direction or using techniques like back-propagation.

We have designed our system with a distributed anomaly based intrusion detection system that has presence in IoT gateways, every edge cloud and the public cloud. In this work we have restricted to the edge clouds and public cloud. Following the architecture of the network, the autoencoders are arranged in a hierarchical manner with Level 1or small 1-2 layer autoencoders at the IoT gateway, Level 2 or medium with 3-5 layer autoencoders at edge clouds and Level 3 or large autoencoders with more than 10 layers in the public clouds. These numbers have evolved from simulations using actual IoT network data and health system data generated on the test bed created in our lab at Washington University in St. Louis.

A diagrammatic representation of the distributed hierarchical structure of the autoencoder based intrusion system is given in Figure 7.5.

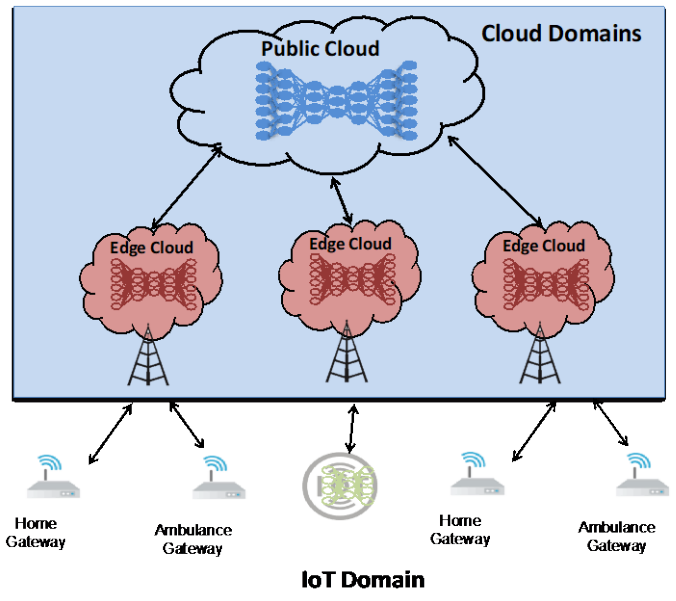


Figure 7.5 Hierarchical autoencoder based system

The dataflow coming from the IoT domain is passed through the autoencoder of the edge clouds in the service area of which the IoT domain falls. We shall see in the next section the training and test datasets used to train these models. The training of Level 2 neural network models at the edge cloud takes relatively less amount of time. The edge cloud areas are relatively small. For instance, In case of mobile edge-clouds, the edge clouds area would coincide with the cell area in which the base station and servers are located. We will also see in the evaluation section that training of edge clouds is fast because of the simplicity of the models. The public cloud area is large, a city, a state, a region or even a full country. The Level 3 neural network models that are useful at the public cloud may be large depending on the variability of the data. Training the autoencoder at the public cloud could take substantial time. To reduce this training time we have worked on merged models at the public cloud, which are aggregation of the edge models. This way the training of layers at the edge can be used at the public cloud. Figure 7.6 illustrates how the edge models are merged to create public cloud model.

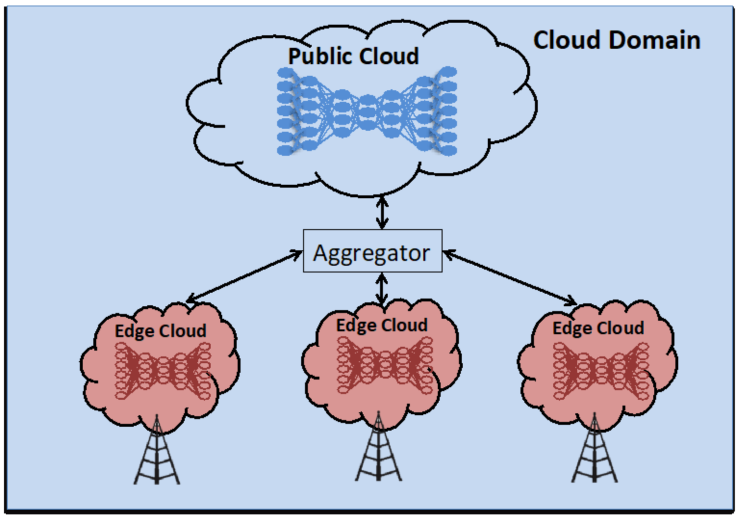


Figure 7.6 The merged model

While in operation, the incoming data passes through the meta-information extractor, which extracts the relevant information from the dataflow. As the dataflow passes through the gateway, the datasets of meta-information are passed though the stacked autoencoders. In case of normal traffic flow the meta-information is reconstructed with root mean square error below the present threshold. In cases where the traffic consisting of patient data has been intruded upon then the value of parameters or the meta-information about the traffic flow is affected. In this case the meta-information can no longer be reconstructed with low root mean square error and the system indicates intrusion. In the present evaluation the meta-information is extracted using Argus Network Management Systems [210] and ranked using the Weka machine-learning tool [147], in future work automatic extraction will be integrated with the system.

7.6 Evaluation of the Proposed System

In this section we first discuss the two test datasets, one generated and the other public, used for evaluation of the network 7.7.2. Then we use these datasets to evaluate the functionalities and draw conclusions.

7.6.1 BOT-IoT Dataset

Table 7.2 The feature set of the BoT-IoT dataset

Feature	**Description**	**Feature**	**Description**
pkSeqID	Row Identifier	Spkts	Source-to-destination packet count
Stime	Record start time	Dpkts	Destination-to-source packet count
Flgs	Flow state flags seen in transactions	Sbytes	Source-to-destination byte count
flgs_number	Numerical representation of feature flags	Dbytes	Destination-to-source byte count
Proto	Textual representation of transaction protocols present in network flow	Rate	Total packets per second in transaction
proto_number	Numerical representation of feature proto	Srate	Source-to-destination packets per second
Saddr	Source IP address	Drate	Destination-to-source packets per second
Sport	Source port number	TnBPSrcIP	Total Number of bytes per source IP
Daddr	Destination IP address	TnBPDstIP	Total Number of bytes per Destination IP.
Dport	Destination port number	TnP_PSrcIP	Total Number of packets per source IP.
Pkts	Total count of packets in transaction	TnP_PDstIP	Total Number of packets per Destination IP.
Bytes	Totan number of bytes in transaction	TnP_PerProto	Total Number of packets per protocol.
State	Transaction state	TnP_Per_Dport	Total Number of packets per dport
state_number	Numerical representation of feature state	AR_P_Proto_P_SrcIP	Average rate per protocol per Source IP. (calculated by pkts/dur)
Ltime	Record last time	AR_P_Proto_P_DstIP	Average rate per protocol per Destination IP.
Seq	Argus sequence number	N_IN_Conn_P_SrcIP	Number of inbound connections per source IP.
Dur	Record total duration	N_IN_Conn_P_DstIP	Number of inbound connections per destination IP.
Mean	Average duration of aggregated records	AR_P_Proto_P_Sport	Average rate per protocol per sport
Stddev	Standard deviation of aggregated records	AR_P_Proto_P_Dport	Average rate per protocol per dport
Sum	Total duration of aggregated records	Pkts_P_State_P_Protocol_P_DestIP	Number of packets grouped by state of flows and protocols per destination IP.
Min	Minimum duration of aggregated records	Pkts_P_State_P_Protocol_P_SrcIP	Number of packets grouped by state of flows and protocols per source IP.
Max	Maximum duration of aggregated records	Attack	Class label: 0 for Normal traffic, 1 for Attack Traffic
Flgs	Flow state flags seen in transactions	Category	Traffic category

The BOT-IoT dataset has been made available by the UNSW Canberra Cyber Center updated in November 2018 [211]. It was created using a realistic network environment with simulated existence of IoT devices in the Virtual Network. The environment incorporates a combination of normal and botnet traffic. The dataset includes DDoS, DoS, Service Scan, Keylogging and Data exfiltration attacks. Reliability of the BoT-IoT dataset has been established using different statistical and machine learning methods for forensics purposes compared with the existing datasets. Table 7.2 gives the set of features that have been used.

7.6.2 Data Generated on Testbed

The BoT-IoT dataset consists of meta-information extracted from the flows from many IoT devices. While this would be indistinguishable from that generated in the healthcare systems, we generated a healthcare specific dataset in our own testbed at Washington University in St. Louis. The testbed set-up shown in Figure 7.7 is adapted from [212].

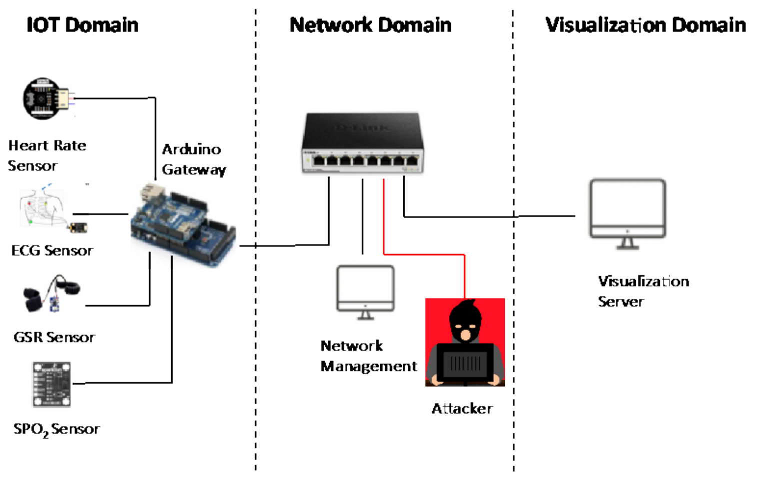


Figure 7.7 Healthcare testbed for dataset generation

The IoT domain: Consists of an Arduino Mega based microcontroller and gateway for health IoT sensors. Because of the absence of any Wi-Fi or Ethernet post on this microcontroller, an external Ethernet shield was attached. The following sensors have been used: heart rate monitor, pulse oxygen sensor, electrocardiography sensor and galvanic skin response and body temperature sensor.

The Network Domain: A 24 port Ethernet Switch to which the microcontroller and three servers were connected, all configured as one private network with each device assigned a private IP. The switch in turn was connected to the Internet. One of the servers was used to mirror and record traffic coming to the network from the IoT domain.

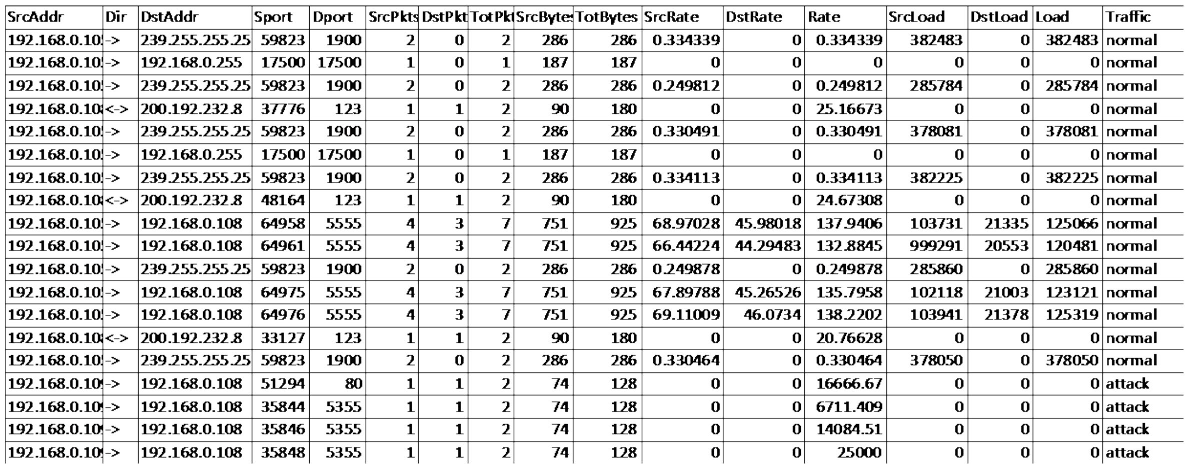
The Visualization Domain: A Linux server was used to visualize patient's data generated through the sensors described above.

Table 7.3 Features in the healthcare dataset

SrcAddr	Source Address	DIntPkt	Destination inter packet arrival time (ms)	Load	Bits per second
DstAddr	Destination Address	SIntDist	Source inter packet arrival time distribution	Loss	Packets transmitted or dropped
Sport	Source Port #	DIntDist	Destination inter packet arrival time distribution	sMinPktSz	Min packet size for source traffic
Dport	Destination Port #	SIntPktAct	Source active inter packet arrival time	dMinPktSz	Max packet size for source traffic
SrcBytes	Source-to-destination byte count	DIntPktAct	Destination active inter packet arrival time	pLoss	Percent packet transmitted or dropped
DstBytes	Destination-to-source byte count	SrcJitter	Source jitter (ms)	pSrcLoss	Percent source packet transmitted or dropped
SAppBytes	Source to destination application bytes	DstJitter	Destination jitter (ms)	pDstLoss	Percent destination packet transmission or dropped
DAppBytes	Destination to source application bytes	sMaxPktSz	Max packet size for source traffic	Dur	Duration of a flow
SrcLoad	Source bits/sec	dMaxPktSz	Max packet size for dest. traffic	Trans	Aggregation record cound
DstLoad	Destination bits/sec	DstGap	Destination bytes missing	TotPkts	Total transaction packet count
SrcGap	Source bytes missing	SIntPkt	Source interpacket arrival time (ms)	TotBytes	Total transaction byte count

Attacker: A server with BlackArch Kali Linux operating system was used to cause malicious activity in different domains.

The normal and attack datasets have the features given in Table 7.3. An extract of the dataset is given in Figure 7.8.



SrcAddr	Dir	DstAddr	Sport	Dport	SrcPkts	DstPkt	TotPk	SrcByte	TotBytes	SrcRate	DstRate	Rate	SrcLoad	DstLoad	Load	Traffic
192.168.0.10	->	239.255.255.25	59823	1900	2	0	2	286	286	0.334339	0	0.334339	382483	0	382483	normal
192.168.0.10	->	192.168.0.255	17500	17500	1	0	1	187	187	0	0	0	0	0	0	normal
192.168.0.10	->	239.255.255.25	59823	1900	2	0	2	286	286	0.249812	0	0.249812	285784	0	285784	normal
192.168.0.10	<->	200.192.232.8	37776	123	1	1	2	90	180	0	0	25.16673	0	0	0	normal
192.168.0.10	->	239.255.255.25	59823	1900	2	0	2	286	286	0.330491	0	0.330491	378081	0	378081	normal
192.168.0.10	->	192.168.0.255	17500	17500	1	0	1	187	187	0	0	0	0	0	0	normal
192.168.0.10	->	239.255.255.25	59823	1900	2	0	2	286	286	0.334113	0	0.334113	382225	0	382225	normal
192.168.0.10	<->	200.192.232.8	48164	123	1	1	2	90	180	0	0	24.67308	0	0	0	normal
192.168.0.10	->	192.168.0.108	64958	5555	4	3	7	751	925	68.97028	45.98018	137.9406	103731	21335	125066	normal
192.168.0.10	->	192.168.0.108	64961	5555	4	3	7	751	925	66.44224	44.29483	132.8845	999291	20553	120481	normal
192.168.0.10	->	239.255.255.25	59823	1900	2	0	2	286	286	0.249878	0	0.249878	285860	0	285860	normal
192.168.0.10	->	192.168.0.108	64975	5555	4	3	7	751	925	67.89788	45.26526	135.7958	102118	21003	123121	normal
192.168.0.10	->	192.168.0.108	64976	5555	4	3	7	751	925	69.11009	46.0734	138.2202	103941	21378	125319	normal
192.168.0.10	<->	200.192.232.8	33127	123	1	1	2	90	180	0	0	20.76628	0	0	0	normal
192.168.0.10	->	239.255.255.25	59823	1900	2	0	2	286	286	0.330464	0	0.330464	378050	0	378050	normal
192.168.0.10	->	192.168.0.108	51294	80	1	1	2	74	128	0	0	16666.67	0	0	0	attack
192.168.0.10	->	192.168.0.108	35844	5355	1	1	2	74	128	0	0	6711.409	0	0	0	attack
192.168.0.10	->	192.168.0.108	35846	5355	1	1	2	74	128	0	0	14084.51	0	0	0	attack
192.168.0.10	->	192.168.0.108	35848	5355	1	1	2	74	128	0	0	25000	0	0	0	attack

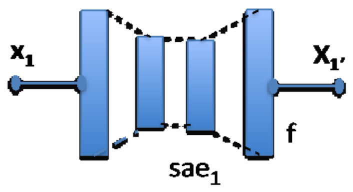
Figure 7.8 Extract of the Training Dataset

7.6.3 Evaluation Results

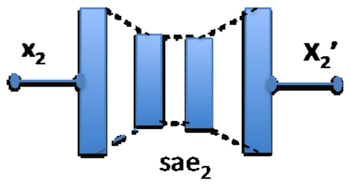
The results discussed in this section are the training of neural networks at the edge cloud, the training of public cloud neural network with and without layer reuse.

1. Training and Testing of Stacked Autoencoders at the Edge Clouds

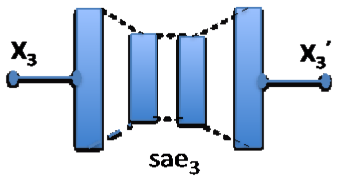
The configuration discussed in Section 7.6 and represented in Figures 7.5 and 7.6 was used as the basis for training and testing. The generated dataset was randomized and mutually exclusive parts were selected to train the autoencoders in the three edge clouds. All the examples in the training datasets constituted normal traffic. Some parts of each of the segregated datasets were kept apart to be used as training datasets. The models did not see these datasets in the training phase. Figure 7.7 shows three-edge cloud and Algorithm 7.1 gives a look into how they could be trained in Keras (with TensorFlow backend).



(a) Autoencoder edge cloud 1 (sae1)



(b) Autoencoder edge cloud 2 (sae2)



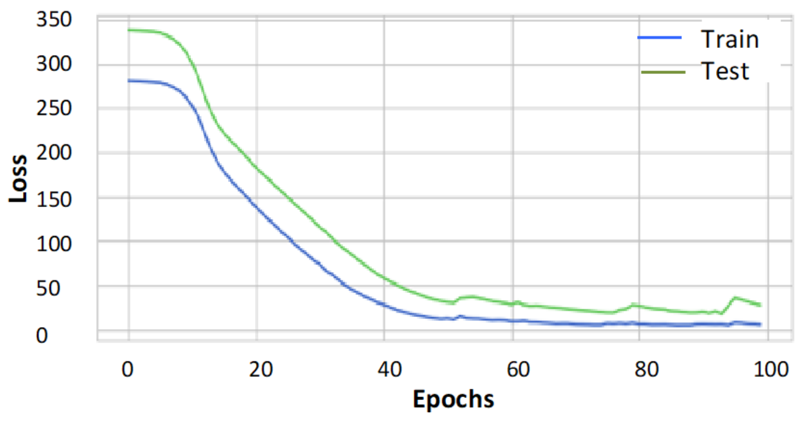
(c)Autoencoder edge cloud 3 (sae3)
Figure 7.7 Three edge cloud autoencoders

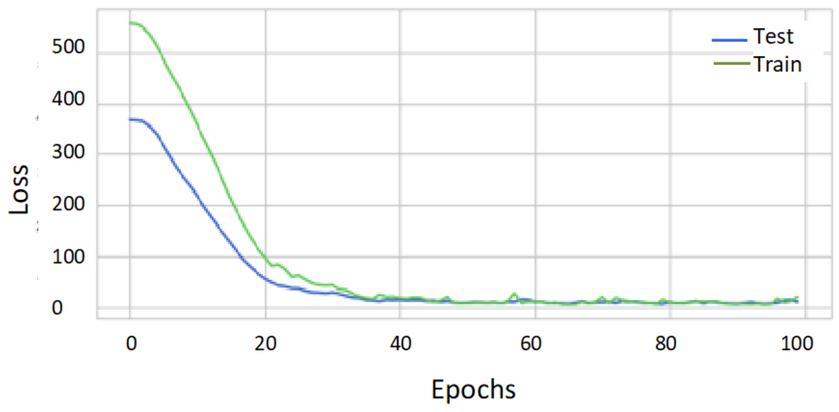
Code Snippet 7.1

```
...
...
X_train1, X_test1 = train_test_split(X1, test_size=0.2,
                        random_state=42)
input_dim1=X_train1.shape[1]
input_layer1 = Input(shape=(input_dim1, ))
enc1_edge1 = Dense(encoding_dim, activation="tanh",
                        activity_regularizer=regularizers.l1(10e-
                        8))(input_layer1)
enc2_edge1 = Dense(int(encoding_dim/2),
                        activation="relu")(enc1_edge1)
dec1_edge1 = Dense(int(encoding_dim/2),
                        activation='relu')(enc2_edge1)
dec2_edge1 = Dense(input_dim, activation='relu')(decoder1_edge1)
sae1 = Model(inputs=input_layer1, outputs=decoder2_edge1)
sae1.compile(optimizer='adam',
                       loss='mean_squared_error',   metrics=['accuracy'])
```

The algorithm shows part of the code for edge-cloud 1. The model is created in line 63 and set up for training in line 64. Lines 56 to 62 define the training and test data, the number of features being fed to the input layer and sets up the autoencoder layers. The other edge cloud models are created similarly

The training and testing results are shown in Figures 7.8 (a) to (c). It can be seen from the figures that the training accuracies are good and the model generalizes well. The blue lines represent training losses while the green lines represent test losses. Each epoch represents one forward plus one back-propagation iteration of the complete dataset through the autoencoder. As the number of epochs increase, the model gets trained and the losses come down. The losses of the models on the test data settle down to their low value close to the training losses in 40-60 epochs of training.


Train-loss = 4.697, Test-loss = 17.595, Train-accuracy = 0.993, Test-accuracy = 0.995
(a) Edge cloud 1
Train-loss = 8.526, Test-loss = 8.661, Train-accuracy = 0.986, Test-accuracy = 0.985
(b) Edge cloud 2



Train-loss = 7.418, Test-loss = 6.715, Train-accuracy, = 0.993, test-accuracy = 0.992

c) Edge cloud 3

Figure 7.8 Training and test performance of the edge cloud models

2. Training and testing of stacked autoencoders at the public cloud

The aggregator in the simple mode merges all the layers of the edge models and produces a composite model (Figure 7.9). Aggregator can also be made to merge layers selectively to improve the outcome.

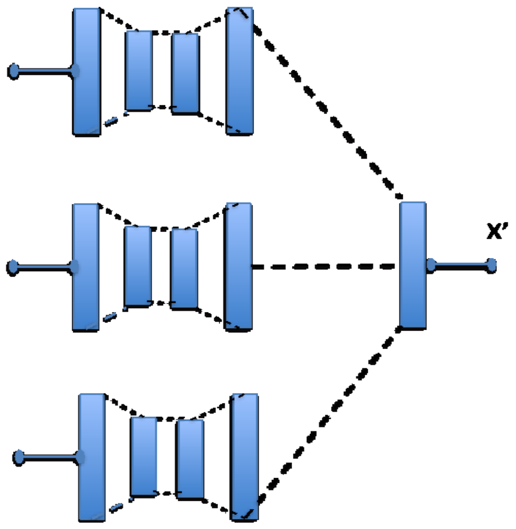


Figure 7.9 Merged model with all edge layers reused

Figure 7.10 shows selective merging of layers. We shall compare the training times of public cloud neural network models created ab-initio with the models created by reusing trained layers from edge cloud neural networks.

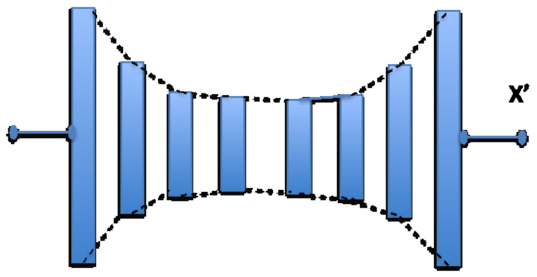


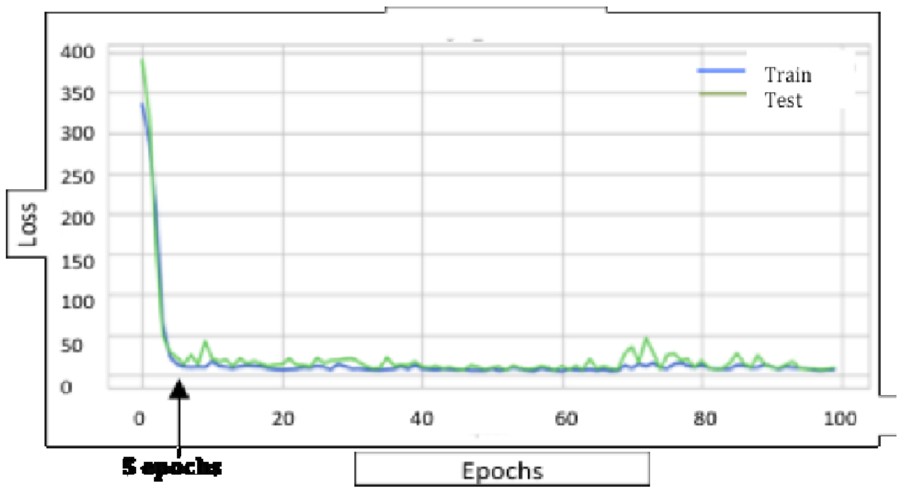
Figure 7.10 Merged model with selective layer reuse

To test our assumption that the Level 3 public cloud model will take much longer to train than the Level 2 edge clouds and that the reuse of the layers from the edge clouds will make the public cloud train faster we followed our aggregation model Figure 7.6. Algorithm 7.2 shows a part of the Keras program that can be used to merge the edge models to form public cloud model.

Code Snippet 7.2

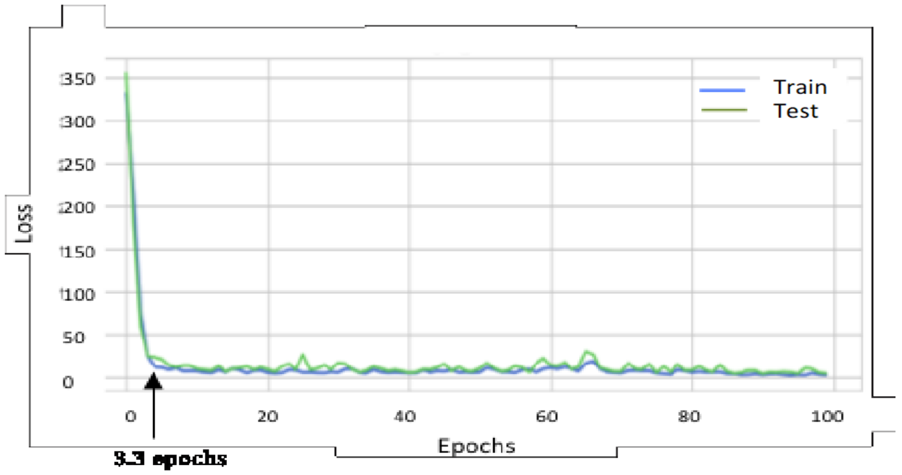
```
…
…
merged_input=keras.layers.concatenate([encoder1_edge1,encoder1_edge2,encoder1_edge3], axis=-1)
merged_output=keras.layers.concatenate([decoder2_edge1,decoder2_edge2,decoder2_edge3], axis=-1)
output_main = Dense(input_dim, activation='relu')(merged_output)

merged_model=Model(inputs= [input_layer1, input_layer2, input_layer3], outputs=output_main)
merged_model.compile(optimizer='adam',  loss='mean_squared_error', metrics=['accuracy'])
```

The training and testing results are shown in Figure 7.11 a) for the merged model in the public cloud while Figure 11 b) shows results for the merged model.



Train-loss = 4.517, Test-loss = 6.151 Train-accuracy = 0.989, Test-accuracy = 0.990

a) Public cloud merged layers



Train-loss = 2.1575, Test-loss = 4.1718, Train-accuracy = 0.992, Test-accuracy = 0.993

b) Public cloud merged layers (cross-trained)

Figure 7.11 Public cloud merged model training and testing

Figure 7.12 shows the training and validation losses for increasing number of epochs. It is seen that the performance stabilizes between 35 and 40 epochs. Figure 7.13 shows the training speed of the merged model in the public cloud. It is seen that because of use of already trained layers from the edge clouds, the model in the public cloud stabilizes between 6 and 8 epochs. This is a great improvement in speed of training because of reuse, in the public cloud, of layers trained in the edge clouds.

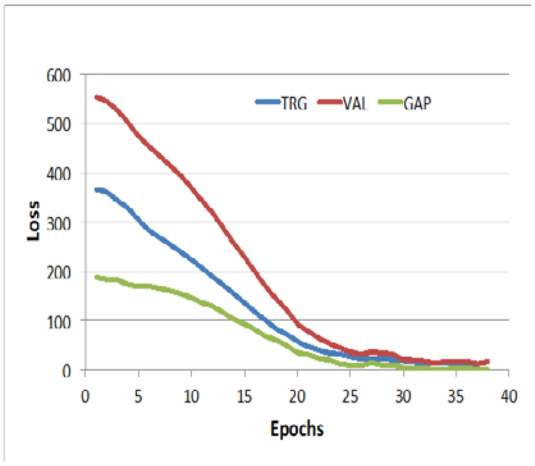


Figure 7.12 Edge cloud training

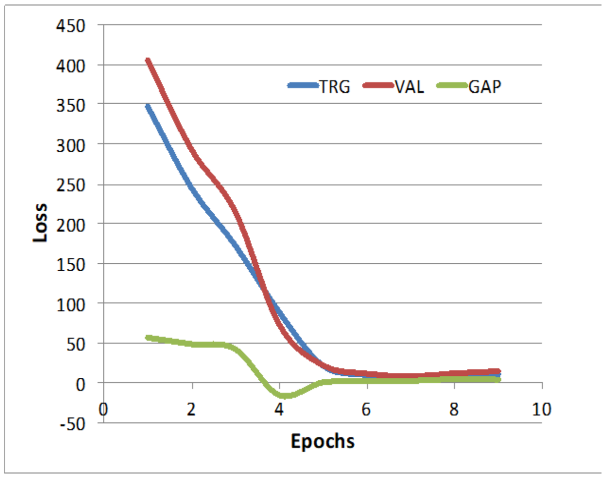


Figure 7.13 Public cloud training

The experimental results discussed above show that merged the model trains faster than even much smaller edge clouds. Now we return to the point that the training speed of merged public cloud will be faster than if the public cloud was made with fresh layers. To compare the improvement in training times, we constructed public cloud autoencoder model from new layers. From several runs, we see that even much smaller 'fresh' public cloud models take much less than a merged 12-layer model. A comparison of training in both the cases is given in Table 7.3. We can see from the table that even a 12-layer model with layer reuse takes 15.7% to 27.63% less than a 4-layer untrained model.

Table 7.3 Comparison of reuse and non-reuse models

	No layer reuse		**With layer reuse**	
Parameter	**4 layers**	**8 layers**	**12 layers**	
Trainable parameters	14941	31521	24737	24737
Training Time	158.99 s	176.183 s	115.055s	134.067 s

To see the performance of the model in filtering out the attack cases, we used the attack data both from the data generated on our testbed as well as data from the BOT-IoT dataset. It is seen that the anomalous data produce very high root mean square error and it is easy to fix a threshold value that decides whether the data has been affected by malicious activity. The confusion matrix for several runs is given in in Table 7.4.

Table 7.4 Confusion matrices for attack detection

Sl. No.	Attack as Attack (TP)	Attack as Normal (FN)	Normal as Attack (FP)	Normal as Normal (TN)	Total vectors	Accuracy (%)
1.	200	0	19	355	574	96.69
2.	188	0	0	511	699	100.00
3.	92	42	0	580	714	94.12
4.	96	0	3	232	331	99.09
5.	92	16	0	184	292	94.52
6.	100	0	0	280	380	100.00
7.	80	17	0	243	340	95.00
8.	110	1	1	287	399	99.50
9.	27	1	1	100	129	98.45

The random runs generally give low false positives and the accuracy ranges from about 95% to 100%.

7.7 Summary and Future Directions

Deep neural network in the form of stacked in the form of stacked autoencoders are being investigated by researchers. Some successes have been reported, using ensemble of autoencoders [215], using autoencoders with k-means [216] and stacked autoencoder with random forest for prediction [217]. To our best of knowledge no investigation is available for the IoT-multi-cloud infrastructure, especially with hierarchical and merged autoencoders. Not all methods would be suitable by way of performance or speed for such and environment. Based on the architecture discussed in section 7.3, a hierarchical model seemed a natural configuration to explore. While the usefulness was apparent after careful testing with multiple datasets, responsiveness required improvement. The training times at the gateways and edge clouds were well within the performance limits expected. However, the training times in the public clouds were reason for deeper investigation. Previously studies ensembles of autoencoders, trained separately not only increase the training time but the final decision taken by another trained model, that takes inputs from the ensemble autoencoders, introduce a layer that could enhance false positives and false negatives. We thus explored reducing the training time through an innovative technique of merged model with layer reuse. The trained layers in the edge clouds are reused at the public cloud to speed up the process with only top-up training required at the public cloud. Exceeding our expectations, not only the timings improved drastically, the accuracies were much better than those that were achieved by training a new stacked-autoencoder at the public cloud.

As for future direction, we need to think about the reasons why the autoencoder classifies a particular stream as attack or normal traffic. Neural networks are opaque, non-intuitive and difficult for people to understand [213]. They are not intuitive and provide limited ability to explicitly identify possible causal relationships. In critical applications like healthcare, the clinical experts may like to know why a certain diagnostic decision was made by the system. Explainable AI will be essential if users are to understand, appropriately trust, and effectively manage this incoming generation of artificially intelligent partners. The challenge will be to produce a new suite of machine learning techniques that produce explainable models but still having high prediction accuracy as shown in Figure 7.14

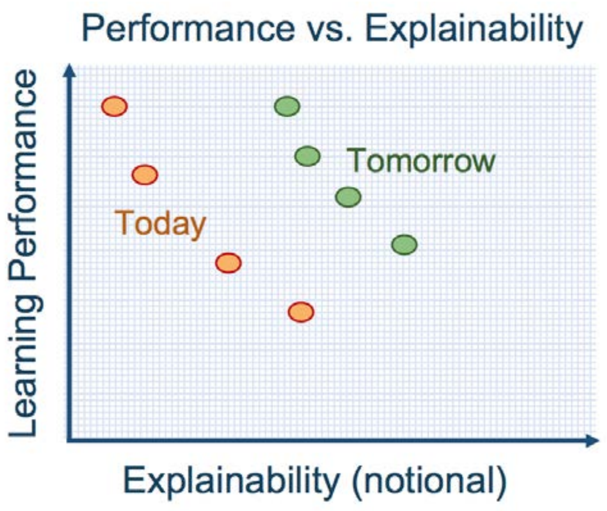


Figure 7.14 Learning Performance and Explainability of current and future systems [DARPA]

The acceptability of AI by the medical profession, the courts of law and the common man for whom the decisions are being taken will depend on how much trust it can generate by providing explanations for its decisions. xAI researchers need to address the challenge of providing AI the capability of providing contrasting explanations of the decisions taken by them. The new AI models should be able to produce contrasting explanations, which can be discussed and debated by the affected parties to arrive at the most feasible explanation [214].

Chapter 8

Conclusions and future directions

Cloud computing has taken roots in business and the government for routine as well innovative applications. The variety of equipment and the software and the speed with which the infrastructure, software or platforms can be leased over the clouds, becomes a great disincentive in factoring in procurement and deployment of hardware and software at ones own expense and at the cost of project time. As the experience of organizations with cloud grew, its use for IT applications has already gone through its own mini-generations – physical datacenters, virtualization in datacenters, cloud datacenters, cloud microservices, cloud containers and now multi-cloud systems! Clouds have become the test-bed for easier and faster testing of innovative ideas.

Despite advancements in the inherent technologies, the very nature of cloud computing resists zero downtime. Cloud compute, storage and networking failures are quite common even for established cloud services, resulting in hours of service blackouts. This is forcing organizations involved in critical services like airline reservations and healthcare as well as carriers' virtual network service deployments consider multi-cloud deployment a necessity. It is, therefore, not surprising that most organizations have some kind of a multi-cloud strategy. Carrier network function virtualization over clouds is a major evolving paradigm, with far reaching effects, on both the carrier as well as cloud computing domains. As would be in any new arena, problems abound.

8.1 What Have We Achieved in this Dissertation?

All multi-cloud applications require a management and control platform that would be able to ease the pains of application managers in onboarding and managing the lifecycle of applications on virtual resources distributed over geographically disparate clouds. When we started this work, multi-cloud platforms were few and far between and the shortcomings were quite glaring. These platforms were a complex piece of software that needed to perform multiple activities simultaneously using all the techniques in the book – synchronous and asynchronous processes, multi-threading, concurrency and fate decoupling. With all these complexities it is not surprising that the platforms that cater to multifarious applications are actually not optimized for any of those. One of the first tasks that we undertook was to evolve a procedure that will help in fine-tuning these platforms to the applications that they are catering to. We developed a three-stage process consisting of dynamically profiling platform behavior, carrying out a two factorial analysis for rating the importance of these factors and then using the outcome to optimize the platform. Details of the work done in this area are explained in Chapter 4.

It was now time for greater adventure. All the forecasts and projections by industry pundits were speaking of carrier service deployments over clouds as the next big thing for both the cloud service providers as well as the carriers. It has the multi-billion dollar potential for earning money for the former and saving money for the later. The idea of virtualized services started in 2012-13 and today snowballed into nothing less than frenzy. All majors carriers around the world AT&T, Verizon, BT, Orange, Telecom Italia and the standards organizations ETSI, ITU

Linux, IETF trying to put the pieces in place. We worked on the placement of virtual network resources onto multi-cloud systems. Using some innovative ideas we achieved results, in many cases better than the state-of-the art. This work has been explained in Chapter 5.

Having optimized the platform, placed the elements of a complex application like carrier network service over multiple clouds we turned our attention to the performance issue of these applications. It is common knowledge by now that performance and availability are two areas that are inhibiting real deployments of virtualized carrier services over clouds. We studied the problem and zeroed in on the complexities and factors that we considered are most affective for virtual carrier service deployments and how they can be tackled. This led to the development of a framework called HYPER-VINES that would assist in meeting the performance and availability challenges. This is elaborated in Chapter 6. NSF chipping in with funding for this research was a great encouragement.

Service like next generation healthcare would only be possible through a confluence of clouds, virtualized services and artificial intelligence. The missing piece here was security of data in motion between the IoT domain and the multi-cloud hierarchy consisting of edge and public clouds. We drew up a plausible architecture for such services and decided to have distributed hierarchical security solution based on neural networks. The part of the problem that we tackled was to have good accuracy in detecting any kind of intrusions that would cause anomaly in data flows. This also involved optimizing the training time of comparatively large neural network models in the public cloud. Our work on this is explained in Chapter 7.

8.2 Future Directions

The work done in this dissertation is a step towards making it possible to deploy carrier network services over multi-cloud systems by improving their performance and availability and to ensure security of data in motion over these infrastructures. Any single research work cannot aim to completely solve a problem. In the words of Thorstein Veblen, a well-known American economist, "The outcome of any serious research can only be to make two questions grow where only one grew before."

For the new researchers, who wish to take the path that has started to emerge through this research, a number of obstacles await to be overcome. In H.L. Mencken's words, "If a solution seems to be too simple and obvious it is probably wrong (adapted from H.L. Mencken). We will try to point out a few possible directions one can take. The confluence of the fields of NFV, AI, Multi-cloud and Cybersecurity is powerful for, which like confluences of rivers, have the potential to slough off the obsolete [218] and give rise to new paradigms that will change the way services are given and make them many orders more advanced in all respects.

With 5G standards to be released in the year 2020 and way they are shaping up, all future networks would have edge computing and virtualization as supporting pillars. The virtualized network services would be deployed on a combination of mobile edge cloud and public clouds.

Carrier services by very nature require high uptime. We have discussed the need for multi-cloud deployment for these services. Issues involved in making virtualized carrier services a reality are still being worked out. Future has more complexity in store. With 5G due in 2020 all future networks would have some mandatory features like multi-cloud deployment, virtualized network services and use of artificial intelligence. These networks have three broad areas of services – ultra reliable low latency communication, enhanced mobile broadband and massive machine type communication. Since all of these are expected to work in virtual network slices over the same physical infrastructure, multi-cloud placement would become a complicated affair. Comparatively simple problems of today i.e. meeting QoS, latency and optimizing cost would pale in comparison with what would be required in future. Not only multiple sets of criteria would govern placements over multi-cloud, optimization would require resources to move from one layer to another and change in specifications of virtual resources as they move from one to another would be taken for granted. None of these could possibly materialize without support of virtualization and multi-cloud.

Carrier networks deployed in accordance with the performance agreements can only guarantee successful acceptance testing and commissioning. Continuous successful operations require a framework that would that would keep meeting the service level agreements and standards for performance and availability. With multiple layers of virtualization, multi-cloud deployments and highly demanding services the simple HYPER-VINES framework presented in this work would have to undergo drastic changes. They would have to deal with virtual slice managers, mobile edge and public cloud providers and a variety of high bandwidth, low latency and IoT based services. One direction to watch is the AI pipelines for self-organizing networks. Different from today's machine learning pipelines, the AI pipelines would conceptually consist of a variety of sensors that collect performance data from various parts of the network. It then uses intelligence to decide on the deep or machine learning models to use and will do the job of feature extraction if required. After the set-up it would send necessary information to the network manager to reorganize the network. The pipeline itself would be able to perform some simple performance management tasks.

Planners of critical services like next generation healthcare are looking at reducing healthcare cost and improving diagnostics and monitoring of patients. Use of virtualized network services and clouds would increase the cyber security risks. We have proposed an AI solution for data in motion among edge and main clouds. Much work remains in providing end-to-end security to patient data for precluding any threat to patients' health, life and money. Use of AI models introduces an element of skepticism about the results obtained. These systems may analyze a situation with high accuracy, but they do not provide any help in understanding how they have arrived at these results. It must be possible to explain results to the main stakeholders – patients and doctors – to make them trust the system. Explainable AI (xAI) has started to address this in a very rudimentary way. Much more research is required to make the AI systems produce opposing viewpoints about the proposed solution to help the affected parties discuss and debate the results.

Bibliography

[1] P. Mell, T. Grance, “The NIST Definition of Cloud Computing,” NIST, Special Publication 800-145, 2011

[2] P.M. Figliola, E.A. Fischer, “Overview and Issues for Implementation of the Federal Cloud Computing Initiative: Implications for Federal Information Technology Reform Management,” Congressional Research Service (CRS) Report (7-5700), 2015

[3] A. Ajabre, “Cloud Computing for Increased Business Value,” International Journal of Business and Social Science, 2012, pp. 234-239

[4] B. Butler, “And the cloud provider with the best uptime in 2015 is… “, Network World, IDG, 2016

[5] R. Mijumbi et al., “ Network Function Virtualization: state-of-the-art and research challenges,” IEEE Communications Surveys and Tutorials, 2016, pp.236-262.

[6] C. J. Bernardos et al., “ Network Virtualization research challenges,” Internet Engineering Task Force (IETF) Draft. https://tools.ietf.org/html/draft-irtf-nfvrg-gaps-network-virtualization-10 Accessed Sept 2, 2019

[7] L. Gupta, R. Jain, M. Samaka, A. Erbad, D. Bhamare, "Performance Evaluation of Multi-Cloud Management and Control Systems," Recent Advances in Communications and Network Technology, 2016

[8] L. Gupta, R. Jain, M. Samaka, "Analysis of Application Delivery Platform for Software Defined Infrastructures," International Journal of Communication Networks and Distributed Systems, 2016

[9] M.A. Khan, C. Hankendi, A.K. Coskun, M.C. Herbordt, “Software Optimization for Performance, Energy, and Thermal Distribution: Initial Case Studies,” International Green Computing Conference and Workshops (IGCC), 2011, pp. 1-6.

[10] K. Sen, D. Marinov, G. Agha, “CUTE: A Concolic Unit Testing Engine for C,” ESEC-FSE’05, September 5–9, 2005,

[11] C. Cader, P. Godefried, S. Khurshid, C.S. Pasareanu, “Symbolic Execution for Software Testing in Practice –Preliminary Assessment,” ACM ICSE 2011

[12] P. González de Aledo, P, Sanchez, R. Huuck, “An Approach to Static-Dynamic Software Analysis,” Artho C., Ölveczky P. (eds) Formal Techniques for Safety-Critical Systems. Communications in Computer and Information Science, vol 596, 2016

[13] B. Palanisamy, A. Singh, and L. Liu, “Cost-effective resource provisioning for MapReduce in a cloud,” Parallel and Distributed Systems, IEEE Transactions on, vol. PP, no. 99, pp. 1–1, 2014.

[14] J. Lee, S Kang, Survey on software testing practices, IET Software, 2012]

[15] Eklov, D., Nikoleris, N. and Hagersten, E. (2012) A Profiling Method for Analyzing Scalability Bottlenecks on Multicores, Technical Report 2012-030, Department of Information Technology, Uppsala University.

[16] M. Yamamoto, M. Ono, K. Nakashima, Unified Performance Profiling of an Entire Virtualized Environment International Journal of Networking and Computing –Volume 6, Number 1, pages 124–147, January 2016

[17] Du, N. Sehrawat, W. Zwaenepoel, “Performance Profiling of Virtual Machines,” Proceedings of the 7th ACM SIGPLAN/SIGOPS international conference on Virtual execution environments Pages 3-14 March 9–11, 2011

[18] H.K. Cho, T. Moseley, R. Hank, D. Bruening, S. Mahlke, “Instant Profiling: Instrumentation Sampling for Profiling Datacenter Applications,” IEEE/ACM International Symposium on Code Generation and Optimization (CGO), pp. 1-10, 2013.

[19] M. Martins, C. Ahmed, V. Raiciu, M. Olteanu, M. Honda, R. Bifulco, and F. Huici, “Clickos and the art of network function virtualization,” in Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation, 2014, pp. 459-473.

[20] F. Lopez-Pires and B. Baran, “Virtual machine placement literature review,” Polytechnic School, National University of Asuncion, Tech. Rep., 2015. [Online]. Available: http://arxiv.org/abs/1506.01509.

[21] C.J. Bernardas, A Rahman, JC Zunjia, L.M. Contreras, P. Aranda, P. Lynch “Network Virtualization Research Challenges,” IETF internet draft, 2018.

[22] Series G: Transmission Systems And Media, Digital Systems And Networks, The E-model: a computational model for use in transmission planning, Recommendation ITU-T G.107, 2015.

[23] F. Callegati, W. Cerroni, C. Contoli, G. Santandrea, “Performance of Network Virtualization in cloud computing infrastructures: The OpenStack case,” IEEE 3rd International Conference on Cloud Networking (CloudNet), 2014, pp. 132-137.

[24] K. Li, H. Zheng, and J. Wu, “Migration-based Virtual Machine Placement in Cloud Systems,” IEEE Cloudnet, 2013, pp. 83-90

[25] ETSI GS NFV-INF 010 V1.1.1, “Network Functions Virtualization (NFV): Service Quality Metrics,” 2014

[26] Network Functions Virtualisation (NFV); Accountability; Report on Quality Accountability Framework, ETSI GS NFV-REL 005 V1.1.1, 2016.

[27] R. Yu, G. Xue, V.T. Kilari, X. Zhang, “Network Function Virtualization in the Multi-Tenant Cloud, IEEE Network,” 2015, pp 42-47.

[28] ITU-T Recommendation M.3400 Series M: TMN AND Network Maintenance: International Transmission Systems, Telephone Circuits, Telegraphy, Facsimile and Leased Circuits TMN Management Functions,” 2002

[29] L. Gupta, M. Samaka, R. Jain, A. Erbad, D. Bhamare, C. Metz, "COLAP: A Predictive Framework for Service Function Chain Placement in a Multi-cloud Environment," The 7th IEEE Annual Computing and Communication Workshop and Conference (CCWC), 2017.

[30] D. Young, M. Toussaint “Hype Cycle for Enterprise Networking and Communications,” Gartner Report #G00338722, 13 July 2018, 69 pp.

[31] ETSI Whitepaper, “Network Functions Virtualisation: An Introduction, Benefits, Enablers, Challenges & Call for Action,” SDN and OpenFlow World Congress, 2012, 16 pp.

[32] ETSI Report, “ETSI Plugtests demonstrate high interoperability levels and increased feature support,” http://www.etsi.org/index.php/news-events/news/1276-2018-02-news-2nd-etsi-nfv-plugtests-demonstrate-high-interoperability-levels-and-increased-feature-support, February 2018.

[33] ETSI GR NFV-IFA 015 V2.1.1, Group Report, “Network Functions Virtualisation (NFV) Release 2; Management and Orchestration; Report on NFV Information Model,” 2017.

[34] ITU-T Recommendation M.3400 Series M: TMN AND Network Maintenance: International Transmission Systems, Telephone Circuits, Telegraphy, Facsimile and Leased Circuits TMN Management Functions,” 2002

[35] ITU Recommendation X.733, “Information Technology-Open System Interconnection-Systems Management-Alarm Reporting Function,” 1992

[36] ISO 9595: "Information Processing Systems - Open Systems Interconnection, Management Information Service Definition – Part 2: Common Management Information Service", 22 December 1988.

[37] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee, “Network function virtualization: Challenges and opportunities for innovations,” Communications Magazine, IEEE, vol. 53, no. 2, 2015, pp. 90–97

[38] R.J. Priyadarsini, L. Arockiam, “Failure Management In Cloud:An Overview,” International Journal of Advanced Research in Computer and Communication Engineering, 2013

[39] A. Yilmaz, “Comparative study for identification of multiple alarms in telecommunication networks,” Turkish Journal of Electrical Engineering & Computer Sciences, 2016

[40] ISO 9596, “Information Processing Systems - Open Systems Interconnection, Management Information Protocol Specification - Part 2: Common Management Information Protocol,” 22 December 1988.

[41] D. Gruer, I. Khan, R. Ogier, R. Keffer, “An Artificial Intelligence Approach to Network Fault Management,” SRI International, 2015

[42] P. Reynolds, C. Killian, J. L. Wiener, “Pip: Detecting the Unexpected in Distributed Systems,” 3rd Symposium on Networked Systems Design & Implementation, 2006, pp. 115-128.

[43] R.R. Kompella, J. Yates, A. G. Greenberg, A. C. Snoeren, “Fault Localization via Risk Modeling,” IEEE Trans. Dependable Sec. Computing, 2010, pp. 396-409.

[44] B.Y. Lee, B. C. Lee, “Fault Localization in NFV Framework,” ICACT, 2016, pp. 352-355.

[45] H. Aziz, A. Guled, “Cloud Computing and Healthcare Services,” Journal of Biosensors and Bioelectronics, 2016

[46] M. Cousin, T. Castillo-Hi, G. H. Snyder, “Devices and Diseases: How the IOT is Transforming Medtech,” Deloitte University Press, 2015

[47] “Concurrency Profiling,” 2013. Available from: http://msdn.microsoft.com/en-us/library/dd264994.aspx

[48] G. Paré et al, “Clinical effects of home telemonitoring in the context of diabetes, asthma, heart failure and hypertension: A systematic review,” Journal of Medical Internet Research, 2010

[49] PwC Global State Information Security Survey, 2015.

[50] IOT Guardian for the Healthcare Industry, Whitepaper, ZingBox, 2016

[51] C. Tsaia, Y. Hsub, C. Linc, W. Lin, “Intrusion detection by machine learning: A review,” Expert Systems with Applications, 2009.

[52] G.D. Waddington, N. Roy, D.C. Schmidt, “Dynamic Analysis and Profiling of Multi-threaded Systems,” 2009, IGI Global.

[53] G. Ren, E. Tune, T. Moseley, Y. Shi, S. Rus, R. Hundt, “Google-Wide Profiling: A Continuous Profiling Infrastructure For Data Centers,” IEEE Micro, vol. 30, pp. 65-90, 2010

[54] D. Jackson, M. Rinard, “Software Analysis: A Roadmap,” Proceedings of the IEEE International Conference on Software Engineering, 2000, pp. 133-145.

[55] García-Ferreira, Iván et al. “A Survey on Static Analysis and Model Checking.” SOCO-CISIS-ICEUTE (2014).

[56] Clarke, E.M., Grumberg, O. and Peled D.A. (2000) Model Checking. The MIT Press, Massachusetts Institute of Technology, Cambridge, Massachusetts.

[57] M. Rinard, “Analysis of Multithreaded Programs,” Proceedings of the 8th International Symposium on Static Analysis, 2001, pp. 1-19.

[58] J. Mars, R. Hundt, ‘‘Scenario Based Optimization: A Framework for Statically Enabling Online Optimizations,’’ Proceedings of the Int’l Symposium on Code Generation and Optimization (CGO 09), IEEE CS Press, 2009, pp. 169-179.

[59] W.A. Rankothge, J. Me, F. Le, A. Russo, J. Lobo, “Towards making network function virtualization a cloud computing service,” IEEE International Symposium on Integrated Network Management (IM), May 2015.

[60] H. Y. Xiong, B. Alipanahi, L. J. Lee, H. Bretschneider, D. Merico, R. K. Yuen, Y. Hua, S. Gueroussov, H. S. Najafabadi, T. R. Hughes, et al., “The human splicing code reveals new insights into the genetic determinants of disease,” Science, 2015

[61] Z.A. Mann and M. Szab, “Which is the best algorithm for virtual machine placement optimization?” This paper has been accepted for publication in Concurrency and Computation: Practice and Experience, 2017

[62] D.B. Oljira, K.-J Grinnemo, J. Taheri, A. Brunstrom, “A Model for QoS-Aware VNF Placement and Provisioning,” IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), 2017, pp. 1-7.

[63] R. Gouareb, V. Friderikos, A.H. Aghvami, "Delay Sensitive Virtual Net-work Function Placement and Routing," 25th International Conference on Telecommunications, 2018, pp. 394 – 398

[64] D. Li, P. Hong, K. Xue, J. Pei, “Virtual Network Function Placement Con-sidering Resource Optimization and SFC Requests in Cloud Datacenter,' IEEE Transactions On Parallel And Distributed Systems, Vol. 29, No. 7, Ju-ly 2018 pp. 1664-1677.

[65] R. CZiva, C. Anagnostopoulos, D.P. Pesaros, "Dynamic, Latency-Optimal vNF Placement at the Network Edge," IEEE Conference on Computer Communications(INFOCOM), 2018, pp. 693-701.

[66] A. Leivadeas, M. Falkner, I. Lambadaris, M. Ibnkahla, G Kesidi, "Balancing Delay and Cost in Virtual Network Function Placement and Chaining," IEEE NetSoft 2018 - International Workshop on Smart network Technolo-gies and Edge computing for the Tactile Internet (STET), 2018, pp. 433-440.

[67] S. Ahvar, H. P. Phyu, S.M. Buddhacharya, E. Ahvar, N. Crespi, and R. Glitho, “CCVP: Cost-Efficient Centrality-Based VNF Placement And Chaining Algorithm For Network Service Provisioning,” in Proc. IEEE Conference on Network Softwarization (NetSoft), 2017, pp. 1-9.

[68] Z. Xu, X. Zhang, S. Yu, Ji Zhang, "Energy-efficient Virtual Network Func-tion Placement in Telecom Networks," IEEE International Conference on Communications (ICC), 2018, 1-7.

[69] T.W. Kuo, B.H. Liou, K.C.J. Lin, M.J. Tsai, Member, “Deploying Chains of Virtual Network Functions: On the Relation Between Link and Server Usage,” IEEE/ACM Transactions On Networking, 2016, pp. 1562-1576.

[70] M. Dieye, S. Ahvar, J. Sahoo, E. Ahvar, R. Glitho, H. Elbiaze, N. Crespi, “CPVNF: Cost-Efficient Proactive VNF Placement and Chaining for Value-Added Services in Content Delivery Networks,” IEEE Transactions On Network And Service Management, Vol. 15, No. 2, June 2018, pp. 774-786.

[71] L. Zhao, J. Liu, "Optimal Placement of Virtual Machines for Supporting Multiple Applications in Mobile Edge Networks," IEEE Transactions on Vehicular Technology, 2018, pp. 6533 – 6545.

[72] L. Askari, A. Hmaity, F. Musumeci, M. Tornatore, "Virtual-Network-Function Placement For Dynamic Service Chaining In Metro-Area Networks," International Conference on Optical Network Design and Modeling (ONDM), 2018, pp. 136-141

[73] M Boucadair, C. Jaquenet, “Handbook of research on redesigning the future of Internet architectures,” IGI Global, 2015, 621 pp.

[74] Gap Analysis on Network Virtualization Activities draft-bernardos-nfvrg-gaps-network-virtualization-00, 2016

[75] G. Kunzmann, C. Goncalves, R. Mibu, “OPNFV – Doctor Fault Management,” OPNFV Summit, Linux Foundation, 2015

[76] M.A. Al-Zain, B. Soh, E. Pardede, "A survey on data security issues in cloud," Journal of Software, Vol. 8, No. 5, 2013, pp.1068–1078.

[77] M. Lichman, UCI Machine Learning Repository http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science https://archive.ics.uci.edu/ml/datasets.html, 2013.

[78] B. Addis, D. Belabed, M. Bouet, S. Secci, “Virtual Network Functions Placement and Routing Optimization,” IEEE 4th International Conference on Cloud Networking, pp. 171-177, Oct 2015

[79] ETSI GS NFV-REL001, “European Telecommunications Standards Institute: Network Functions Virtualization (NFV); Resiliency Requirements,” 2015

[80] ETSI GS NFV INF 010, “European Telecommunications Standards Institute: Network Functions Virtualization (NFV); Service Quality Metrics,” 2017

[81] ETSI GS IFA 013, “European Telecommunications Standards Institute: Network Functions Virtualization (NFV); Management and Orchestration; Os-Ma-Nfvo reference point - Interface and Information Model Specification,” 2016.

[82] ONAP Whitepaper, “Open Network Automation Platform: Architecture Overview,” 2017, https://www.onap.org/wpcontent/uploads/sites/20/2017/12/ONAP_CaseSolution_Architecture_120817_FNL.pdf, Accessed March 20, 2018

[83] Y. Endo and M. Seltzer, “Improving interactive performance using TIPME,” Proc. ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, Volume 28, Issue 1, 2000, pp. 240-251.

[84] M. Chen, E. Kiciman, E. Fratkin, E. Brewer, and A. Fox, “Pinpoint: problem determination in large, dynamic, internet services,” Proc. International Conference on Dependable Systems and Networks (IPDS Track), 2002, pp. 595-604.

[85] P. Barham, R. Isaacs, R. Mortier, and D. Narayanan, “ Magpie: online modeling and performance-aware systems,” Proc of the 9th conference on Hot Topics in Operating Systems, 2003, pp. 15-15.

[86] B. Trammell et al., “mPlaneBuilding an intelligent measurement plane for the internet,” IEEE Communications Magazine, Volume: 52, Issue: 5, 2014, pp. 148-156.

[87] OPNVF, “Building fault management into NFV deployments background and purpose of OPNFV’s Doctor Project,” https://www.opnfv.org/wpcontent/uploads/sites/12/2016/11/opnfv_faultmgt_final.pdf. Accessed January 20, 2018

[88] A. Lakhina, M. Crovella, C. Diot, “Diagnosing network-wide traffic anomalies,” SIGCOMM, 2004, 12 pp.

[89] D. Kushnir, M. Goldstein, “Causality Inference for Failures in NFV,” IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS): SWFAN 16: International Workshop on Software-Driven Flexible and Agile Networking, 2016

[90] Cisco Content Hub, "Causality correlation," https://content.cisco.com/chapter.sjs?uri=%2Fsearchable%2Fchapter%2Fwww.cisco.com%2Fcontent%2Fen%2Fus%2Ftd%2Fdocs%2Fnet_mgmt%2Factive_network_abstraction%2F3-7-2%2Ftheory%2Foperations%2FTheoryofOperations%2Fcaus-theory.html.xml&platform=Cisco%20Active%20Network%20Abstraction, 2018. Accessed 20th September 2018

[91] L. Lewis, “A case-based reasoning approach to the management of faults in communication networks,” Infocom, 1993, 1422-1429.

[92] S. Jiang, D. Siboni, A. A. Rhissa, G. Beuchot, “An intelligent and integrated system of network fault management: artificial intelligence technologies and hybrid architectures,” IEEE Networks, 1995, pp. 265-268.

[93] D.W. Gürer, I. Khan, R. Ogier, R. Keffer, “An Artificial Intelligence Approach to Network Fault Management,” SRI International, Citeseer 1996, pp. 1-10.

[94] R.D. Gardner, D. A. Harle, “Alarm correlation and network fault resolution using the Kohonen self-organizing map,” Globecom 1997, pp. 1398-1402.

[95] J. McClelland, A. Rumelhart, “Distributed model of human learning and memory,” Parallel distributed processing: Explorations in the microstructure of cognition (Vol. II). Cambridge, MA: MIT Press, 1986, 550 pp.

[96] G.E. Hinton, “Connectionist learning procedures. Artificial Intelligence,” 1989, pp. 185-234.

[97] P.E. Utgoff , D.J. Stracuzzi, “Many-layered learning,” Neural Computation, 2002, pp. 2497-2529.

[98] A. Krizhevsky, I. Sutskever, and G. Hinton, “Imagenet classification with deep convolutional neural networks,” NIPS, 2012

[99] V. Dhar, “Data Science and Prediction,” Communications of the ACM, 2013, pp. 64-73.

[100] A. Yilmaz, “Comparative study for identification of multiple alarms in telecommunication networks,” Turkish Journal of Electrical Engineering & Computer Sciences, 2016, pp. 677-688

[101] E. Rozaki, “Network Fault Diagnosis Using Data Mining Classifiers,” Eleni Rozaki International Journal of Data Mining & Knowledge Management Process, 2015, pp. 29-40.

[102] M. Jaudet, N. Iqbal, A. Hussain, and K. Sharif, (2005) “Temporal classification for fault-prediction in a real-world telecommunications network,” International Conference on Emerging Technologies, 2005, pp. 209-214.

[103] K. Qadar, M. Adda, “Network Faults Classification Using FCM” International Journal of Advanced Research in Computer and Communication Engineering, 2013.

[104] C. Sauvanaud, K. Lazri, M. Kaniche, K. Kanoun, “Anomaly Detection and Root Cause Localization in Virtual Network Functions,” IEEE 27th International Symposium on Software Reliability Engineering, 2016.

[105] M. Miyazawa, M. Hayashi, R. Stadler, “vNMF: Distributed Fault Detection using Clustering Approach for Network Function Virtualization,” IFIP/IEEE International Symposium on Integrated Network Management (IM2015), 2015.

[106] M. Hayashi, “Machine Learning-assisted Management of a Virtualized Network,” Optical Fiber Communication Conference, Optical Society of America, 2018.

[107] IOT Guardian for the Healthcare Industry, Whitepaper, ZingBox, 2016.

[108] E. Larson, Nilsson, K. Dennis, E. Jonsson, “An approach to specification-based attack detection for in vehicle networks,” IEEE Intelligent Vehicles Symposium, 2008.

[109] P. Tyagi, D.A. Dembla, “Investigating the Security Threats in Vehicular ad hoc Networks (VANETs): Towards Security Engineering for Safer on-road Transportation,” Advances in Computing, Communications and Informatics, 2014.

[110] J. Jaccard, S. Nepal, "A survey of emerging threats in cybersecurity," Journal of Computer and System Sciences, 2014, pp. 973-993.

[111] M. Zamani, M. Movahedi, “Machine Learning Techniques for Intrusion Detection,” arXiv:1312.2177 [cs.CR] v2, 2015

[112] S. Paul, R. Jain, M. Samaka, J. Pan, “Application Delivery in Multi-Cloud Environments using Software Defined Networking,” Computer Networks Special Issue on cloud networking and communications, 2014

[113] R. Jain, “The Art of Computer Systems Performance Analysis,” New York; Wiley, 1991.

[114] "OpenDaylight Platform Overview," The Linux Foundation Projects, https://www.opendaylight.org/what-we-do/odl-platform-overview, accessed 2015, 2019 S

[115] T. Saemundsson, H. Bjornsson, G. Chockler, “Dynamic Performance Profiling of Cloud Caches,” ACM Symposium on Cloud Computing, 2014, pp. 1-14

[116] S. Paul, R. Jain, M. Samaka, J. Pan, "Application Delivery in Multi-Cloud Environments using Software Defined Networking, ” Computer Networks Special Issue on cloud networking and communications, pp. 166-186, Feb 2014.

[117] S. Paul, R. Jain, J. Pan, J. Iyer, D. Oran, “OpenADN: A Case for Open Application Delivery Networking,” 22nd International Conference on Computer Communications and Networks (ICCCN), pp. 1-7, 2013

[118] D. Knuth, “Structured Programming with go to Statements,” Computing Surveys, vol 6, No 4, December 1974.

[119] B. Lantz, N. Handigol, B. Heller, V. Jeyakumar, V. (2015) “Introduction to Mininet,” https://github.com/mininet/mininet/wiki/Introduction-to-Mininet, last accessed 18th March, 2019.
[120] Line_profiler, Python Package, https://pypi.org/project/line_profiler/, last accessed 18th March, 2019.
[121] L. Gupta, M. Samaka, R. Jain, A. Erbad, D. Bhamare “A P-ART framework for for Placement of Virtual Network Services in a Multi-cloud Environments,” Elsevier Computer Communication, March 2019
[122] Network Functions Virtualization (NFV); Virtualised Network Function; Specification of the Classification of Cloud Native VNF Implementations (work-in-progress), ETSI GS NVF-EVE 011 V0.0.9 2018.
[123] R. Ricci, E. Eide, the CloudLab Team, “Introducing CloudLab: Scientific Infrastructure for Advancing Cloud Architectures and Applications,” Usenix login: 39(6), 2014. pp. 36-37.
[124] NFV Use cases, ETSI GS NFV 0001, 2013.
[125] P. Quinn, T. Nadeau, "Problem Statement for Service Function Chaining," IETF RFC 7498, 2015.
[126] D. Bhamare, R. Jain, M. Samaka, A. Erbad, "A Survey on Service Function Chaining." Journal of Network and Computer Applications, 2016, pp. 138-155.
[127] G. Nychis, D. R. Licata, "The Impact Of Background Network Traffic On Foreground Network Traffic," The Proceeding of the IEEE Global Telecommunications Conference GLOBECOM, 2001, pp. 1-16.
[128] Amazon Opworks, https://aws.amazon.com/opsworks/, 2019, last accessed 2 January 2019.
[129] S. Clayman, E. Maini, A. Galis, A. Manzalini and N. Mazzocca, “The Dynamic Placement of Virtual Network Functions,” IEEE Network Operations and Management Symposium (NOMS), 2014, pp. 1-9.
[130] The CAIDA Anonymized Internet Traces 2016 Dataset, Center for Applied Internet Data Analysis, http://www.caida.org/data/passive/passive_2016, last modified June 2018, last accessed 17 December 2018.
[131] A. Bifet, G. F. Morales, J. Read, G. Holmes, B. Pfahringer, “Efficient Online Evaluation Of Big Data Stream Classifiers,” Proceedings of the 21st ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD ’15), 2015, pp. 59-68.
[132] Y. Sun, Z. Wang, H. Liu, C. Du, J. Yuan, “Online Ensemble Using Adaptive Windowing for Data Streams with Concept Drift,” International Journal of Distributed Sensor Networks, 2016, pp. 1-9.
[133] L.M. Seversky, S. Davis, “On Time-series Topological Data Analysis: New Data and Opportunities,’ IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops, 2016, 1014 – 1022.
[134] D.P. Helmbold, P.M. Long, “Tracking drifting concepts by minimizing disagreements,” Machine Learning, 14(1), 1994, pp. 27-45.
[135] R. Klinkenberg and Th. Joachims, “Detecting Concept Drift with Support Vector Machines,” Proceedings of the 17th International Conference on Machine Learning (ICML), 2000, pp. 487-494.

[136] A.J. Smola, B. Scholkopf, “A Tutorial on Support Vector Regression,” Statistics and Computing, 2004, pp. 199-222.
[137] R.W. Becker, G.V. Lago, “A global optimization Algorithm,” Proceedings of the 8th Allerton Conf. Circuits Systems Theory, 1970.
[138] F.J. Solis, J-b. Wets, “Minimization By Random Search Techniques,” Mathematics Of Operations Research, 1998.
[139] T. Weis, "Global Optimization Algorithms – Theory and Application – e-book, http://www.it-weise.de/projects/book.pdf, 2009, last accessed 10 December 2018.
[140] Zelda B Zabinsky, “Random Search Algorithms,” Wiley Encyclopedia of Operations Research and Management Science, 2009.
[141] A. Zhigljavsky, "Stochastic global optimization," School of Mathematics, Cardiff University, Cardiff, U.K, Springer book 2008 edition
[142] C.J.P. B´elisle, “Convergence Theorems for a Class of Simulated Annealing Algorithms on Rd,” J. Applied Probability, 1992, pp. 885-895
[143] V. Gupta, S. Dharmaraja, andV. Arunachalam, "Stochastic modeling for delay analysis of a VoIP network," Ann Oper Res, Springer Science, 2015, pp. 171-180
[144] A. Akella, “Experimenting with Next-Generation Cloud Architectures Using CloudLab,” IEEE Internet Computing, 2015, pp. 77-81.
[145] M.J. Kearns, “The computational complexity of machine learning,” MIT Press, 1990, 176 pages
[146] S. Raschka, "Model Evaluation, Model Selection, and Algorithm Selection in Machine Learning," arXiv:1811.12808v2 [cs.LG], 2018
[147] E. Frank, M.A. Hall, I. H. Witten, “The WEKA Workbench. Online Appendix for Data Mining: Practical Machine Learning Tools and Techniques,” Morgan Kaufmann, Fourth Edition, 2016.
[148] M.C. Luizelli, L.R. Bays, L.S. Buriol, M.P. Barcellos, L.P. Gaspary, “Piecing Together the NFV Provisioning Puzzle: Efficient Placement and Chaining of Virtual Network Functions,” IFIP, 2015.
[149] L. Wang et al., "Joint Optimization of Service Function Chaining and Resource Allocation," IEEE Access, 2018
[150] M.T. Beck and J.F. Botero, ‘‘Coordinated allocation of service function chains,’’ in Proc. IEEE Global Commun. Conf. (GLOBECOM), Dec. 2015, pp. 1–6
[151] J. Liu, Y. Li, Y. Zhang, L. Su, D. Jin, ‘‘Improve service chaining performance with optimized middlebox placement,’’ IEEE Trans. Serv., 2017.
[152] S.A. Ajila, A.A. Bankole, “Cloud Client Prediction Models Using Machine Learning Techniques,” IEEE 37th Annual Computer Software and Applications Conference, 2013.
[153] A. Alleg, R. Kouah, Moussaoui, T. Ahmed, "Virtual Network Functions Placement and Chaining for Real-Time Applications," 2017, pp. 1-6.
[154] T.W. Kuo, B.H. Liou, K.C.J. Lin, M.-J. Tsai, Member, “Deploying Chains of Virtual Network Functions: On the Relation Between Link and Server Usage,” IEEE/ACM Transactions On Networking, 2016, pp. 1562-1576.
[155] M. Liang, Z. Li, T. Chen, J. Zeng, “Integrative data analysis of multi-platform cancer data with a multimodal deep learning approach,” Transactions on Computing Biology and Bioinformatics, 2015

[156] L. Gupta, T. Salman, R. Das, A. Erbad, R. Jain, M. Samaka, “HYPER-VINES: A Hybrid Learning Fault and Performance Issues Eradicator for Virtual Network Services over Multi-Cloud Systems,” IEEE ICNC 2019

[157] Lav Gupta, Tara Salman, Maede Zolanvari, Raj Jain, Aiman Erbad, "Fault And Performance Management In Multi-Cloud Virtual Network Services Using AI: A Tutorial And A Case Study," Elsevier Computer Networks, February 2019 (submitted revised version).

[158] ETSI GR NFV-IFA 015 V2.1.1, Group Report, “Network Functions Virtualization (NFV) Release 2; Management and Orchestration; Report on NFV Information Model,” 2017.

[159] R. Glitho, “Cloudifying the 3GPP IP Multimedia Subsystem: Why and How?” 6th International Conference on New Technologies, Mobility and Security (NTMS), 2014, pp. 1-5

[160] D. Lopez, “Network Functions Virtualization: Beyond Carrier-Grade Clouds,” Optical Fiber Communications Conference and Exhibition (OFC),” 2014, pp. 1-18

[161] ITU-T SG13 Q19 “Potential New Work For Q19/13” https://www.itu.int/md/T17-SG13-171106-TD-WP2-0139/en, 2017, accessed September 2018.

[162] R. Jain, “Fault and Performance Management in Carrier-grade Virtual Networks Over Multiple Clouds,” NSF Proposal, NETS, 2017.

[163] L. Gupta, M. Samaka, R. Jain, A. Erbad, D. Bhamare, H. A. Chan, “Fault and Performance Management in Multi-cloud based NFV using Shallow and Deep Predictive Structures,” Journal of Reliable and Intelligent Environment, 2017, pp. 1-8.

[164] C.J. Bernardos(Ed.), "Multi-domain Network Virtualization," draft-bernardos-nvvrg-multidomain-04, Informational Internet-Draft, https://tools.ietf.org/html/draft-bernardos-nfvrg-multidomain-05. Last Accessed 4 December 2018.

[165] ETSI GS NFV 002, General Specification, “Network Functional Virtualization; Architectural Framework,” 2013

[166] J. Halpern, C. Pignataro, “Service Function Chaining (SFC) Architecture,” IETF RFC 7665, 2015

[167] J Guichard et al., "NSH and Segment Routing Integration for Service Function Chaining," IETF draft-guichard-SFC-nsh-sr-00, June 2018.

[168] F. Khan, “A Beginner’s Guide to NFV Management & Orchestration (MANO), http://www.telcocloudbridge.com/a-beginners-guide-to-nfv-management-orchestration-mano, 2015.

[169] R Mijumbi, J Serrat, J-L Gorricho, S. Latre, M. Charalambides and D Lopez, “Management and Orchestration Challenges in Network Function Virtualization,” IEEE Communications Magazine, 2016, pp. 98-105

[170] M. Ersue, “Management and Orchestration - An Overview,” ETSI NFV MANO WG Co-chair IETF #88, 2013

[171] ONF-XOS, https://www.opennetworking.org/xos/. Last Accessed November 2018

[172] D. Andrushko, G. Elkinbard, “What is the best NFV Orchestration platform? A review of OSM, Open-O, CORD, and Cloudify,” https://cloudify.co/2017/03/15/what-best-nfv-orchestration-platform-review-osm-openo-cord-cloudify.html, 2017, accessed October 2018

[173] ITU-T E.800 [i.10]: Recommendations, “Definitions of terms related to quality of service,” 2008

[174] TRAI Report, “Performance Indicator Reports,” http://www.trai.gov.in/release-publication/reports/performance-indicators-reports, 2018

[175] H. N. Mhaskar and T. Poggio, “Deep vs. Shallow Networks: an Approximation Theory Perspective,” Center for Brains, Minds, and Machines (CBMM), CBMM Memo No. 054, 2016

[176] V. Sindhwani, “Shallow vs. deep: the great watershed in learning,” Princeton University Lectures, 2017

[177] G. Ososkova, and P. Goncharov, “Shallow and Deep Learning for Image Classification,” Optical Memory and Neural Networks 26(4):221-248, October 2017

[178] J. Schmidhuber, “Deep Learning in Neural Networks: An Overview,” Neural Networks, Elsevier, 2014, pp. 85-117.

[179] T. Hastie, R. Tibshirani, J. Friedman, “The Elements of Statistical Learning,” Springer Science & Business Media, 2009, 745 pp.

[180] M.A. Jabber, B.L. Deekshatulu, P. Chandra, “Alternating decision trees for early diagnosis of heart disease,” Proceedings of International Conference on Circuits, Communication, Control and Computing, 2014, pp. 322-328

[181] G. Louppe, “Understanding Random Forests – From Theory to Practice,” Ph.D. Dissertation, University of Liege, France, 2014.

[182] “Stacked autoencoder (Unsupervised Feature Learning and Deep Learning),” http://ufldl.stanford.edu/wiki/index.php/Stacked_Autoencoders. Last Accessed December 29, 2018.

[183] D. Bhamare, T. Salman, M. Samaka, A. Erbad, R. Jain, "Feasibility of Supervised Machine Learning for Cloud Security," 3rd International Conference on Information Science and Security (ICISS2016), 2016, pp. 1-5.

[184] Kaggle datasets, available at https://www.kaggle.com/c/telstra-recruiting-network/data. Last accessed October 2018.

[185] M.A. Makary, M. Daniel, “Medical error—the third leading cause of death in the US,” British Medical Journal 2016.

[186] "American Health Care: Health Spending and the Federal Budget," Committee for a Responsible Federal Budget, https://www.crfb.org/papers/american-health-care-health-spending-and-federal-budget. Last accessed 20th March 2019

[187] M. Cousin, T. Castillo-Hi, G. H. Snyder, “Devices and Diseases: How the IOT is Transforming Medtech,” Deloitte University Press, 2015.

[188] H. Aziz, A. Guled, “Cloud Computing and Healthcare Services,” Journal of Biosensors and Bioelectronics, 2016.

[189] G. Paré et al, “Clinical effects of home telemonitoring in the context of diabetes, asthma, heart failure and hypertension: A systematic review,” Journal of Medical Internet Research, 2010

[190] PwC Global State Information Security Survey, 2015.

[191] A global study by NetDiligence of cyber insurance claims in 2017.

[192] Emerging Threats to Healthcare Providers, Axel Wirth, Symantec Technical Architect, https://www.symantec.com/blogs/expert-perspectives/4-emerging-threats-healthcare-providers, last accessed 22nd December 2018.

[193] Report "Pacemaker Recall Exposes National Need for Research and Education in Embedded Security," by Kevin Fu, Chair Cybersecurity Task Force, University of Michigan Computer Research Association News, Vol 29 No 9, 2017.

[194] W. Widanagamaachchi, Y. Livnat, P. Bremer, S. Duvall, Vv. Pasucci, " Interactive Visualization and Exploration of Patient Progression in a Hospital Setting," AMIA Annual Symposium, 2017, pp. 1773-1782.

[195] A. Josey, "The TOGAF open standard," The Open Group whitepaper, www.theopengroup.org, 1995-2019.

[196] M. Meyer, "The rise of healthcare data visualization," Data Revolution, A Journal of AHIMA Blog, 2017

[197] Healthcare reference architecture, Whitepaper, Extreme Networks, 2018

[198] J. Cusimano, “Assessing the Security of ICS Using Threat Modeling.” https://scadahacker.com/howto/howtothreatmodeling.html, Last accessed on 22 December 2018

[199] R. Shahan, B. Lamos, "Internet of Things (IoT) security architecture," 10/08/2018 https://docs.microsoft.com/en-us/azure/iot-fundamentals/iot-security-architecture, 2018, last accessed 17 February 2019.

[200] K.A. McDonald, A Wirth, "The Intersection of Patient Safety and Medical Device Cybersecurity," HIMSS, 2018

[201] A.K. Roy, “Impact of Big Data Analytics on Healthcare and Society,” Journal of Biometrics & Biostatistics, 2016

[202] M. Hassanalieragh, A. Page, T. Soyata, G. Sharma, M. Aktas, G. Mateos, B. Kantarci, Silvana Andreescu, “Health Monitoring and Management Using Internet-of-Things (IoT) Sensing with Cloud-based Processing: Opportunities and Challenges,” IEEE International Conference on Services Computing, 2015

[203] T. Bajtoš, A. Gajdoš, L. Kleinová, K. Lučivjanská, P. SokolNetwork, "Intrusion Detection with Threat Agent Profiling," Security and Communication Networks, 2018.

[204] A.L. Buczak and E. Guven, “A survey of data mining and machine learning methods for cyber security intrusion detection,” IEEE Communications Surveys & Tutorials, 2016, pp. 1153–1176, 2016.

[205] M. Zamani, M. Movahedi, “Machine Learning Techniques for Intrusion Detection,” arXiv:1312.2177 [cs.CR] v2, 2015

[206] G. Caspi, “Introducing Deep Learning: Boosting Cybersecurity With An Artificial Brain,” Information Week IT Network, 2016

[207] J. Ye, J. Liu, "Sparse methods for biomedical data," ACM SIGKDD Explorations Newsletter 14(1), 2012

[208] J.V. Tu, "Advantages and Disadvantages of Using Artificial Neural Networks versus Logistic Regression for Predicting Medical Outcomes," Clin Epidemid Vol. 49, No. 11, pp. 1225-1231, 1996 Copyright 0 1996 Elsevier Science.

[209] M. Hassanalieragh, A. Page, T. Soyata, G. Sharma, M. Aktas, G. Mateos, B. Kantarci, Silvana Andreescu, “Health Monitoring and Management Using Internet-of-Things (IoT) Sensing with Cloud-based Processing: Opportunities and Challenges,” IEEE International Conference on Services Computing, 2015

[210] Argus Documentation, http://nsmwiki.org/Argus, last accessed 25th January 2019.

[211] N. Koroniotis, N. Moustafa, E. Sitnikova, B. Turnbull, “Towards the Development of Realistic Botnet Dataset in the Internet of Things for Network Forensic Analytics: Bot-IoT Dataset”, https://arxiv.org/abs/1811.00701, 2018.

[212] A. Ghubaish, "Healthcare Testbed for Dataset Generation," Rotation Technical Report, Washington University in St. Louis, 2018.

[213] D. Gunning, "Explainable Artificial Intelligence (XAI)," DARPA/I2O Program Update, November 2017.

[214] B. Mittelstadt, C.R.S. Wachter, B. Mittelstadt, C. Russell, S. Wachter, "Explaining Explanations in AI," Conference on Fairness, Accountability, and Transparency, 2019.

[215] Y. Mirsky, T. Doitshman, Y. Elovici, A. Shabtai, "Kitsune: An Ensemble of Autoencoders for Online Network Intrusion Detection," Network and Distributed Systems Security Symposium (NDSS), 2018, Pages 15.

[216] A. Dawoud, S. Shahristani, C. Raun, "Deep Learning for Network Anomalies Detection," International Conference on Machine Learning and Data Engineering (iCMLDE), 2018, pp. 149-153.

[217] N. Shone, T. Nguyen Ngoc, V. D. Phai , Q. Shi, "A Deep Learning Approach to Network Intrusion Detection," IEEE Transactions On Emerging Topics In Computational Intelligence, 2018, Pp. 41-50.

[218] K.B. Sheehan, D.K. Morrison, "Beyond convergence: confluence," https://firstmonday.org/article/view/2239/2121, 2009. Last accessed 11 March, 2019.