Cognitive Smart Agents for Optimising OpenFlow Rules in Software Defined Networks

By

ANN FAIK SABIH

A Thesis Submitted

in Partial Fulfilment of the Requirements for the Degree of

DOCTOR OF PHILOSOPHY

Department of Electronic and Computer Engineering

College of Engineering, Design and Physical Sciences

BRUNEL UNIVERSITY LONDON

September 2017

Abstract

This research provides a robust solution based on artificial intelligence (AI) techniques to overcome the challenges in Software Defined Networks (SDNs) that can jeopardise the overall performance of the network. The proposed approach, presented in the form of an intelligent agent appended to the SDN network, comprises of a new hybrid intelligent mechanism that optimises the performance of SDN based on heuristic optimisation methods under an Artificial Neural Network (ANN) paradigm. Evolutionary optimisation techniques, including Particle Swarm Optimisation (PSO) and Genetic Algorithms (GAs) are deployed to find the best set of inputs that give the maximum performance of an SDN-based network.

The ANN model is trained and applied as a predictor of SDN behaviour according to effective traffic parameters. The parameters that were used in this study include round-trip time and throughput, which were obtained from the flow table rules of each switch. A POX controller and OpenFlow switches, which characterise the behaviour of an SDN, have been modelled with three different topologies. Generalisation of the prediction model has been tested with new raw data that were unseen in the training stage. The simulation results show a reasonably good performance of the network in terms of obtaining a Mean Square Error (MSE) that is less than 10^{-6}. Following the attainment of the predicted ANN model, utilisation with PSO and GA optimisers was conducted to achieve the best performance of the SDN-based network. The PSO approach combined with the predicted SDN model was identified as being comparatively better than the GA approach in terms of their performance indices and computational efficiency.

Overall, this research demonstrates that building an intelligent agent will enhance the overall performance of the SDN network. Three different SDN topologies have been implemented to study the impact of the proposed approach with the findings demonstrating a reduction in the packets dropped ratio (PDR) by 28-31%. Moreover, the packets sent to the SDN controller were also reduced by 35-36%, depending on the generated traffic. The developed approach minimised the round-trip time (RTT) by 23% and enhanced the throughput by 10%. Finally, in the event where SDN controller fails, the optimised intelligent agent can immediately take over and control of the entire network.

Dedication

In loving memory of my **Dad**: *You are there even when I don't see you.*

Mom: *I owe you all that I have achieved and aim to achieve.*

Sura: *I will forever cherish the love and laughter we share.*

Omar: *You are the greatest thing to ever happen to me.*

My Teachers: *Thank you for making me the person I am today.*

Declaration

I declare that this thesis is my own work and is submitted for the first time to the Post-Graduate Research Office. The study was originated, composed and reviewed by myself and my supervisors in the Department of Electronic and Computer Engineering, College of Engineering, Design and Physical Sciences, Brunel University London UK. All the information derived from other works has been properly referenced and acknowledged.

Ann Faik Sabih
September 2017

Acknowledgements

I would like to express my sincere gratitude to my supervisors Prof. Hamed Al-Raweshidy and Dr. Maysam Abbod for the continuous support during my PhD and related research. It was due to your patient approach that I remained motivated to complete my research and gain immense knowledge. Your guidance paved the path for me to write this thesis with such competence and coherence. I certainly could not have hoped for a better research team for my PhD study.

I would like to also extend my appreciation for my dear friends Yousif Al-Dunainawi and Bilal Al-Kaseem who were always there to offer help and advice. Your company is what kept me smiling throughout the gloomiest of days and made me feel like having my family around me the entire time. Additionally, a warm thank you to my son Hussein Al-Ali for the amazing time we spent together. Your kindness knew no bound and your unconditional support, encouragement and help were evident throughout the few years we've known each other.

There were many friends that formed an inseparable part of my PhD experience; Shireen, Laith, Raad, Rand, and Rasha, you've made my time at Brunel truly wonderful. There are many other friends who I may not have the space to mention, but I will always remain indebted for their love and kindness, especially during the difficult times.

I would also like to thank my sponsor, the Iraqi Ministry of Higher Education and Scientific Research, as represented by the Iraqi Cultural Attache in London, for giving me this opportunity to complete my study. I am very grateful for their unstinting support and care throughout the past four years.

Lastly but not least I wish to thank my loving family and all my uncles and aunties, especially aunt Sana Al-Kayyat for always making me feel special. Your warm heart and kind personality were my escape even when all other doors were shut. My lovely brothers Omar and Younis: thank you for your love and support, your existence in my life enriched it with gracious moments and irrepressible laughter, thank you for being part of it.

Table of content

Table of figures

Table of tables

List of Abbreviation

Abbreviation	Meaning
ACO	Ant Colony Optimization
ACPI	Application Control Plane Interface
AI	Artificial Intelligence
ANN	Artificial Neural Network
ASIC	Specific Integrated Circuits
BYOD	Bring Your Own
DPi	Deep Packet Inspection
DS	Differentiated Services
E2E	End To End Delay
EC	Evolutionary Computing
ECN	Explicit Congestion Notification
FFANN	Feedforward Artificial Neural Network
FIB	Forwarding Information Base
GAs	Genetic Algorithms
IDS	Intrusion Detection Systems
IGP	Interior Gateway Protocol
IoT	Internet Of Things
IP	Internet Protocol.
IPS	Intrusion Prevention Systems
LM	Levenberg-Marquardt
LTE	Long Term Evolution
MSE	Mean Square Error
MST	Minimum Spanning Tree

NOS	Network Operating System
ONF	Open Networking Foundation
OSPF	Open Shortest Path First
PSO	Particle Swarm Optimisation
QoE	Quality of Experience
QoS	Quality of Service
RAM	Random Access Memory
RMs	Rack Managers
RNN	Recurrent Neural Network
RRT	Round Trip Time
SDN	Software Defined Networks
SFSA	Smart Flow Steering Agent
SSL	Secure Sockets Layer
STP	Spanning Tree Protocol
TCAM	Ternary Content Addressable Memory
TCP	Transmission Control Protocol
TLV	Type-Length-Value
U-ACAL	Universal Access Control Lists
UDP	User Datagram Protocol
UMTS	Universal Mobile Telecommunications Systems
VLAN	Virtual Local Area Network
VLSI	Very Large Scale Integration
VMs	Virtual Machines

1. Chapter One: Introduction

1.1. Preliminaries to Programmable Network

The standard computer network is constructed using multiple network devices, which are associated with very complex system rules. They include switches, routers, and several other types of middle boxes. The term ‘middle box’ refers to a component that modifies or influences traffic for an objective unrelated to packet forwarding, such as a firewall. It is the job of the network administrator to create policies and rules that are compatible with a broad variety of network conditions and scenarios. This is achieved by manually creating low level configuration requests out of high end policies. The problem is that administrators usually have to fulfil all of these goals with only a small number of basic resources. At the same time, they must ensure that the devices are flexible enough to be able to handle inconsistent conditions. Normally, network devices are vertically assimilated *black boxes*. As a result, the incidence of errors is high and the obstacles that network managers have to overcome are plentiful. These are just some of the reasons why network maintenance and construction is such a difficult job.

One important obstacle that network managers and creators struggle with is called ‘internet ossification.’ Ultimately, the internet is almost impossible to develop manually and to enhance, due to the enormous size of its deployment base and the increasing dependency we are putting on its resources (as an essential component of our social infrastructure). This applies to its rules, policies, guiding principles, general performance, and physical requirements. As existing and future internet applications and tools grow ever more sophisticated, the need for their development is becoming increasingly urgent. According to many experts, this kind of ossification is an almost insuperable issue, one that still has no obvious solution.

However, a potential solution has been proposed in the form of ‘programmable networks’, with some technology experts believing that they can support and enrich network development. The primary objective is to give software developers an opportunity to utilise network tools in the same uncomplicated, productive way they can with other computing and data storage resources. SDN works by logically positioning network data in software based controllers (the control plane) and consequently, the network components function as basic packet forwarding tools (the data plane), which can be configured with the use of an open interface [1].

Much attention is being paid to Software Defined Networking (or SDN), which is a very young technology that can provide a networking paradigm such that the forwarding hardware is not inextricably linked to control actions. The result is faster and more efficient network management, requiring less time and fewer resources. The implication is that it could also allow for consistent, regulated development and innovation.

Over the last decade, artificial intelligence and soft computing strategies have played an increasingly important part in the construction of contemporary networks (smart transport systems, for example). The introduction of these capabilities provides opportunities for enhancing the quality and integrity of existing computer networks. It is opening the door to novel ways of solving both cutting edge network problems (associated with SDN). In sum, the assimilation of AI strategies and the abstraction construct of the SDN paradigm has the potential to produce more versatile activity from network components.

1.1.1. Concepts of Programmable Network

The growing popularity of big data science and cloud servers is placing even more strain on traditional network structures. These require precision grained regulation of server to server networking, as well as enhanced stability, safety, and easy functionality. Similarly, it is becoming more common for applications to range across multiple databases and even servers that are geographically distinct and isolated from one another. As a result, basic user to server processes are now dwarfed by the amount of traffic produced by server to server interactions. Increasingly, these strains are being overcome through the use of SDN, which allows for network owners swapping the manual interface for a programmable version. This supports the automation of processes and actions (for example, policy regulation and configuration), being also a good way of helping networks to react faster to a variety of application requests.

The purpose of network programming scenarios and conditions is to offer the provisions required to program novel network structures in an efficient, versatile manner. The way to identify a programmable network from a conventional one is to determine whether it has the capability of being configured via a marginal series of APIs, from which a practically unlimited variety of high end services can be constructed. Crucially, network programming scenarios cannot provide fundamental network algorithms with the ability to describe and distinguish network structures (for signalling or routing, for example). So, for such scenario a

series of network programming interface hardware, which supports the design and creation of network structures is required. From an abstract perspective, it is the equivalent of using software development ‘toolboxes’ to create novel applications, although, the application is actually a network structure in this context [2].

1.1.2. Traditional Network Architecture

As the amount of components and requests relating to the Internet of Things (IoT) and cyber physical frameworks increases, information and communication technology resources must continue to develop and evolve [3], [4]. In fact, a number of researchers have proved that conventional (older) networks are no longer sophisticated enough to fulfil rapidly expanding demands. One of the reasons for this is that the network devices are vertically assimilated and this creates a multifarious system that is very difficult to maintain [5], [6]. Conventional networks are only robust enough to facilitate vendor specific protocols [5], [1]. When it comes to fast moving network conditions, they simply do not provide a reasonable degree of adaptability.

The reality is that all computer networks can be tricky to maintain. They include a large number of components; everything from switches to routers, attack detection systems, firewalls, server load balancers, address translators, and more. The switches and routers depend on intricate, carefully configured control policies, which are normally private and exclusive to the network owner. These policies apply network protocols that will pass through an enormous amount of interoperability and standardisation evaluations. It is common for network managers to construct separate network components via the use of configuration interfaces. However, the exact nature of these interfaces depends on the company that originally created them. In fact, distinct variations often exist among interfaces designed by the same company as well. While network management resources sometimes provide a primary stance from which the network can be constructed, systems of this kind do still function according to separate configuration interfaces, rules, policies, and components. It is this limited system that is preventing internet infrastructures from developing further. It has made network management substantially more challenging and raised operating costs.

The underlying weakness of conventional network structures is that they continue to be inflexible. Consequently, opportunities for their development are few, particularly when it comes to network management, regulation, safety, scalability, error tolerance, virtualisation,

programmability, and automation. To provide a potential solution, some experts propose a different method of broadening network regulation and influence. The strategy is mostly based on the idea that reconfiguring standalone computer systems is a simple task. At the moment, managers, operators, and users do not have a lot of influence and there is not a lot of focus beyond just attempting to fulfil current working requirements. So, a viable solution is becoming increasingly important [7]. Over the last decade, Software Defined Networking (SDN) has emerged as the most suitable replacement for older networking structures, offering fast, dynamic, and adaptable network management. Furthermore, it converts overly intricate network structures into much more controllable versions [8].

This substantial degree of adaptability and versatility, as compared with traditional hardware, is one of the reasons why SDN is such a promising prospect. It should be pointed out that traditional networks cannot be 'programmed' in any valuable way, which is why active networking represents a much needed and very innovative form of network regulation. It involves a programmable interface (or network API) that uncovers resources (packet queues, processing, storage, etc.) on the separate network nodes [7]. The aim is to facilitate the creation of bespoke functions which can be combined with the subsets of packets that travel through the nodes.

To summarise, SDN is a rapidly developing paradigm with the ability to overcome the limitations of vertical assimilation in older networks. The objective is to increase adaptability as a way to facilitate the configuration of networks via centrally positioned (logical) network policies and protocols. Crucially, SDN is able to modify its network parameters very quickly by utilising information provided by the networking conditions or scenario. Furthermore, the standalone framework of SDN offers an improved approach to network management, leading to the creation of more efficient and easy to handle systems. The centrally positioned cost efficient structure offers greater network transparency and this supports the productive use of resources as well as the strengthening of performance. As a result of the progressively persistent presence of 'intelligent' programmable components within networks, SDN is needed improving safety and energy efficacy, whilst also enhancing network virtualisation.

1.1.3. Cognitive System

Recently attention has been drawn to Cognitive network as a promising technology to improve the performance of the network by distributes the system resources efficiently.

Over the last ten years, the scope and quality of communication networks has increased at a rapid rate. This is particularly true when it comes to mobile technologies and commercial IP networks (such as Long Term Evolution [LTE] or Universal Mobile Telecommunications Systems [UMTS]). For one thing, consumer demand for network resources has expanded and the range of services offered has been diversified to meet it. Consequently, the amount of network nodes has expanded too. In many cases, the movement of users presents a big challenge. Modern mobile traffic requires super-fast responses and the ability to maintain performance across broad geographical expanses. To achieve this, the paradigm of network management and application needs to be improved. Furthermore, most services present distinct traffic requests and some (such as cloud solutions and video streaming) depend on very fast connections as well as a high degree of stability and consistency [9] .

In previous years, the application of new networks was only possible after a period of meticulous designing, testing, and preparation. The purpose of this was to ensure long term viability and simultaneously, to accommodate for the existing and future (forecasted) demands of end users. The problem is that all of the challenges discussed in previous sections (growing number of users, more pressure on traffic, etc.) serve to make this period of preparation obsolete. The only real alternative is the reactive and adaptive regulation of networks as a way to facilitate the flexible distribution of resources and the universal streamlining of network functions. Normally, network managers are most concerned with configuring them correctly, streamlining performance, and overcoming errors and challenges to security. It should be pointed out that there is another obstacle associated with the rapid expansion of network nodes, end users, and services, which is the old, unsophisticated manner in which networks are currently maintained and regulated [10].

Only a handful of management tasks and activities are fully automated. As a response to this limitation, the notion of Cognitive Systems was developed. It pertains to focusing on consistent evaluations of the network and the application of logical, valuable choices based on network conditions. The problem with traditional systems of management is that they are based on what is known as ‘quasi static’ approaches, i.e. they prioritise centralised solutions, with the human facilitator making an essential contribution. SDN, and other systems like it, depend on reliable algorithms. They are designed to fulfil previously established objectives relating to network maintenance and regulation. Consequently, a suitable control theory or some other type of reactive optimisation strategy is needed for their successful application.

The thing to remember is that networks must function as efficiently as possible. This means that any concerns about the reliability and stability of cognitive algorithms are a major issue. It is certainly possible for heuristically oriented and control theory based algorithms to be incorporated into network regulation and autonomic management systems. Nevertheless, because the network processes are so intricate, the identification of suitable algorithms and the refining of their parameters are either extremely tough or completely impossible. One possible answer is to employ cognitive strategies, which are algorithms with the capacity to learn and adapt. They are a very promising prospect, inspired from how the living human being dealt with problems and solutions. Cognitive techniques consist of numerous algorithms, which based on huge learning system. The algorithms are including neural networks, machine learning, evolutionary algorithms, knowledge based on system and several other methods. The main characteristics of them are studying the problem environment and taking decision according to analyzing results of earlier decisions [8].

1.2. Future Internet Architectures and the Internet of Things

Ultimately, the expansion of Internet of Things (IoT) has presented ever more challenging requirements for both internetworking and networking systems. For example, there is the matter of allowing interactions between components linked to heterogeneous networks. The overall level of adaptability and configurability provided by SDN technologies is hindering the progress of network development. However, it has the potential to benefit network regulation enormously by creating new forms of interaction through the provision of robust (but easy to use) switching components (forwarders). These components are able to utilise any individual field of a message or packet to identify the outgoing port that will receive it. Capabilities like these are relevant to IoT, because they are likely to eliminate many of the problems which it has created. It should be noted that every IoT item (component or related ‘thing’) is shaped, designed, and constructed to fulfil a very particular purpose. Additionally, the conditions (in their entirety) in which some items exist are normally constructed with a specific goal or role in mind. Consequently, in order to realise the benefits in full, networks have to be willing to embrace heterogeneity as part of networking processes, fundamental policies, and networking components [11].

From a wider perspective, IoT involves comprehensive interactions between multiple heterogeneous networks, the items that exist within them, the conditions of their existence, the top and bottom layer policies that define them, and even the distinct goals that they are designed to fulfil. The best way to achieve all of this is with the application of a shared protocol for each of the conditions, scenarios, and items. It is what the IoT is based on and what continues to define its evolution, particularly now that IPv6 is emerging. The problem is that there are still some major challenges to overcome and a long way to go before many of these concepts and proposals become workable systems. Items, conditions, rules, and protocols are constructed to fulfil precise goals, which means that when attempting to apply a shared protocol, this may not even be a viable process and any reduction in central heterogeneity at the network level is going to cause major faults [12].

Then, on the other side of the argument, there is SDN. It embraces the comprehensive configurability of network components, in relation to both middle and end point constructs. To realise in full such an enormous endeavour, a number of general strategies for isolating data and control planes within the routing and switching components have been developed. They depend on the streamlining of middle point network components and converting them into basic, easy to use, packet forwarders. To fulfil the network goals, a refined general control policy is used to apply them, in accordance with the required forwarding principles.

When considering the attributes of SDN, particularly from an IoT viewpoint, it is important to ask what part it has to play in the maintenance of heterogeneous networks and items. This is not the only requirement it must fulfil, for it also needs to contribute to a broader, stronger collaborative system by introducing a higher layer control solution to the network [13]. This solution should be able to communicate efficiently with the SDN controllers. It should be noted that the SDN approach suggests the construction of a conceptually centrally positioned ‘brain’, with a working knowledge of the conditions and topology of the network. The brain is responsible for making choices related to packet forwarding. It converts these selections into new forwarding protocols and delivers them to all of the relevant forwarding components.

1.3. **Motivations**

In recent years, researchers have shown an increased interest in enhancing network performance. The continued evolution of network applications and the ability of many

devices operate from the same location shows that networking requirements have become substantially more sophisticated. Today, internet traffic is a lot more expansive and demanding, with increasingly complex needs, particularly in an age where big data reigns supreme. Now, thousands of end point devices can communicate and access different types of network traffic, with these intricate patterns being too fast paced for traditional networking systems to handle. This is why contemporary data facilities need to be highly adaptable and capable of growing to meet changing demands. Furthermore, new approaches to networking are essential.

When traffic rapidly increases, there is a risk of network congestion. The danger here is that intolerable levels can lead to an unreliable service and, potentially, lost information. It may be possible to remedy the issue and deal with some of the other vulnerabilities by upping the bandwidth and creating additional channels for the data to travel down. Nevertheless, the amount of information that businesses are required to handle has also increased exponentially. The result is a greater degree of pressure and responsibility for network operators, particularly when it comes to providing a high quality user interface. Unfortunately, boosting bandwidth is a costly process, particularly when you have to do it for every single network component; it simply is not affordable in most cases. It certainly is not value for money at times when the amount of traffic is only moderate.

The Software Defined Network, however, has the potential to provide intelligent data monitoring and optimised traffic management. It also makes it possible to gather centralised data from the entire network. Moreover it makes automation, regulation, and reconfiguration easier than ever before. If used correctly, it promises sustainable solutions for all of the issues described. Essentially, the SDN model unchains network controls from the forwarding/data plane within the network devices and instead, offers a centralised form of control. Consequently, the core networking structure can be shifted and adapted to fit different types of service and application.

When huge volumes of data are combined with intelligent computing systems, information is interpreted much faster and this helps network operators to make smarter, more dynamic decisions. Consequently, the assimilation of machine learning/artificial intelligence with SDN architectures is a highly valuable resource and it can substantially increase the integrity and reliability of networks. Currently, artificial intelligence is a great way to observe the activities and responses of SDN applications and virtual machines. If

problems or vulnerabilities are identified, it is easier to make rapid choices regarding the optimal solution.

1.4. Research Aim and Objectives

The aim of this study is the notion (and application) of bringing SDN together with AI strategies and approaches to networking. A new intelligent approach of selecting the optimum flow rules parameter set that is proposed depends on Artificial Neural Networks (ANN), Genetic Algorithms (GAs) or Particles Swarm Optimisation (PSO) to achieve the best performance in an SDN network. It is hoped that this will lead to the discovery of adaptive behaviours for modern network components. To fulfil this goal, several objectives have been pursued to find the potential solutions to meet this aim:

- Simulating the SDN network: By Mininet emulation, three different topologies were generated with different numbers of switches and hosts.
- Data Collection: For three hours, three recommended SDN topologies were run and when finished, comprehensive sets of data were gathered for all three.
- Data analysis: Prior to the training phase, the data that had been gathered were analysed and pre-processed. The datasets were divided into inputs and outputs, being subsequently randomly split into three subcategories: training set (70%), testing set (15%) and validation set (15%).
- ANN Training: The training set data which had been studied was supplied as input for the ANN, which meant the latter could output a prediction in advance of the optimisation period, The table parameters from OpenFlow acted as the input data, while the output of the ANN was shown by latency and throughput.
- Data Post-Processing and Testing: The forecast ANN output was proven with concealed raw data (validation set) in order to authenticate the ANN training. Its accuracy was established using the testing dataset.
- Data Optimisation: In order to ascertain which is the optimum OpenFlow rule table set of the SDN network, the established ANN output was used to run two distinct optimisation techniques. These EAs were evaluated alongside one another to verify which parameter was more effectual when implemented in the above approach.

1.5. Contributions to Knowledge

As understand here, this work makes many vital contributions across numerous crucial areas. The outcomes of this study enhance current knowledge of OpenFlow rule table optimisation in the following ways.

1. **Predicting SDN network behaviour**

   As a means to select the optimum ANN topology and Levenberg-Marquardt (LM) learning algorithm, ANN with a comprehensive search can be deployed. The proficient neural network improves the understanding of SDN performance behaviour. Principally, in terms of its capability of permitting elevated levels of traffic, the ANN provides an uncomplicated way of predicting the effectiveness of SDN systems. This novel, hybridised, framework is created with a variety of distinct topologies. The proposed solution has the capability of both creating and maintaining excellent input/output interactions among complex, non-linear tasks.

2. **Presenting an intelligent optimiser for an SDN network**

   In order to increase the network throughput and decrease the end-to-end delay with relative an innovative and advanced form of hybridised networking, with the aim of improving productivity and efficacy in relation to SDN architectures, is implemented. This is achieved by the adoption of empirical alterations and the incorporation of new neural paradigms. Sophisticated streamlining methods, including Genetic Algorithms (GAs) and Particle Swarm Optimisation (PSO), are deployed as ways of identifying the optimal arrangement of inputs for ensuring high level performance. As described, the proposed framework had to be scrutinised and put to the test by introducing a small amount of previously unseen content. Following this, it was combined with GA and PSO to discover the most beneficial integration of systems and components between the two. The results show that SDN with PSO provides this, because when assimilated they perform at an accelerated level, with fewer inaccuracies and congestion problems than with the other arrangement. The performance comparison of PSO and GA is implemented using MATLAB 2017a.

3. **Identify a suitable method for directing surplus traffic**

   The risk of congestion can be lowered by smart steering principles, which are used to enhance network throughout and optimise resource allocation.

4. **Avoiding traffic congestion and the bottleneck problem**

   When packets have to be updated two or even three times, this holds up movement in the rest of the network and leads to delays. Consequently, it is also necessary to

minimise the number of packets lost and to decrease the drop ratios. This is achieved by avoiding traffic congestion. Ultimately, the goal was to make sure that as few packets as possible are being directed back to the controller to avoid the bottleneck problem that can occurs in the SDN controller.

5. **Improve controller reliability**

   The SDN controller lessens the general network accessibility in SDN, and does so by exhibiting a single point of stoppage. Nonetheless, the entire network is susceptible to collapse when a crucial controller crashes in an SDN network. The suggested method could work as a support smart controller, that is to say, if the main controller fails and the support controller has the same information about the networks, it can make cognitive decisions on its behalf.

1.6. Organization of the Thesis

- Chapter two: Presents the background information for legacy networks and architecture of SDN networks, followed by SDN challenges and the related works regarding the thesis and how of using soft computing is implemented in an SDN network.
- Chapter three: Describes the experimental tools and the topologies adopted for this research, in detail, as well as the controller and traffic generation tools. Also, it introduces a performance prediction scheme for an SDN-based network using different ANN topologies. Finally, it provides the experimental results with their corresponding analysis.
- Chapter four: Introduces the way to optimise a hybrid intelligent system for SDN. In addition, two evolutionary based optimisation methods, namely, genetic algorithms and particle swarm optimisation are introduced. All these intelligent-based approaches are employed in this thesis to enhance network performance by using intelligent controllers.
- Chapter five: Describes the different types of routing protocols used in SDN and gives an explanation of the proposed solution of a smart slow steering agent in such networks. In addition, a comparison of the SDN network performance with the proposed intelligent SDN controller is illustrated.
- Chapter six: Provides a summary of the thesis, conclusions and suggestions for future research avenues.

1.7. List of Publications

- A. Sabbeh, Y. Al-Dunainawi, H. S. Al-Raweshidy and M. F. Abbod "Performance prediction of software defined network using an artificial neural network." *SAI Computing Conference (SAI), 2016*. IEEE, 2016.
- A. Sabeeh, Y. Al-Dunainawi, M. F. Abbod and H. S. Al-Raweshidy "A hybrid intelligent approach for optimising software-defined networks performance." *Information Communication and Management (ICICM), International Conference on*. IEEE, 2016.
- A. Sabih, Y. Al-Dunainawi, H. S. Al-Raweshidy and M. F. Abbod "Optimisation of Software-Defined Networks Performance Using a Hybrid Intelligent System." Advances in Science, Technology and Engineering Systems Journal 2.3 (2017): 617-622.

2. Chapter Two: Literature Survey and Related Works

2.1. Introduction

Traditional (older) networks depend on specialised algorithms, which are applied to particular components (hardware devices) as a way to regulate and observe the movement of information. They can also organise and supervise routing channels as well as making decisions about how many components can be linked to the network [14]. Normally, routing algorithms of this kind are used in conjunction with hardware devices like Application Specific Integrated Circuits (ASIC), which are constructed to fulfil a targeted goal or purpose.

Take packet forwarding, for instance. When it comes to older networks, the receipt of a packet from a routing component triggers the application of a series of protocols positioned deep inside the firmware. These protocols identify the destination component and the routing channel which the packet has taken. Crucially, high end routers are able to choose whether or not packets are handled in certain ways, as a result of their makeup and information type. Normally though and certainly when it comes to basic, low end routers, all data packets bound for the same location are treated in the same way.

A Cisco routing device, for instance, enables a user to determine the objectives of different flows via modified local router programming techniques. Consequently, it is possible to intervene directly in the formation of router queues. To be precise, a tailored, bespoke local router structure facilitates superior management of traffic and helps users to position their goals and demands in order of urgency. However, a weakness associated with this approach is the restriction of existing network components when in the presence of elevated network traffic. This has the potential to cause serious problems and substantially degrade overall performance. The faster network traffic increases, the bigger the challenges relating to safety, dependability, stability, scalability and efficiency become.

It remains very difficult to reconfigure and reorient conventional network processes. Older networking components are not dynamic or flexible enough to handle differing packet types or varying packet contents. This is because the routine protocols are so rigidly applied and embedded with no real room for adaptability [15]. It is a major limitation, as the networks (which form the very heart of the internet) must have the capacity to respond to

shifting networking conditions [16]. They must be able to do this without consuming huge amounts of time and resources.

One potential answer is to stop applying data handling protocols as embedded elements of the hardware and to start treating them as software modules instead. Interestingly, this idea returns to the concept of Software Defined Networking [4]. It was initially constructed by Nicira Networks and it has its roots in tests and experiments performed by researchers at Princeton, Stanford, and CMU [14]. This approach would give network managers a lot more power over network traffic. Consequently, they would have the opportunity to enhance substantially the efficiency of the network and to ensure that it is making full use of its allocated resources.

The primary purpose of SDN is to enable transparent user controlled regulation of the forwarding hardware within a network [17]. It takes advantage of opportunities to isolate the control plane from the data plane inside switches and routers. It should be pointed out that the control plane may be made out of a single or several controllers [18], the exact number being determined according to the size of the network. The job of the data plane is to receive requests and information from the control plane and to apply them as required to the hardware [19]. When several controllers are present, they are able to construct highly efficient and dependable peer to peer network protocols. Whether single or multiple, all of the switches associated with a data plane must provide the same information. The system offers a comprehensive (but simple) perspective on the complete network and it allows users to implement universal modifications without the need for device specific programming. The switches of the data plane are expected to monitor the flow tables, which are also regulated by the controllers. Yet, their only truly essential responsibility is to pass information between themselves. So, the switches are strained more than is necessary, because they are involved with control processes even though this is not strictly a requirement.

Around thirty years ago, some network operators were using highly specialised systems (like cloud storage) to separate the network operating architecture (equivalent to the control processes of the SDN control plane) from the more resource and time heavy requests (equivalent to the movement of information to the data plane). Hence, the truth is that the ideas and constructs relating to SDN are not entirely novel and they have actually been around for quite some time. Currently, cloud technologies allow network storage and computation without the need for local resources. This separation of data and control planes is an essential part of the most sophisticated, wide ranging networking systems. Ultimately,

SDN leads to greater network efficiency and productivity, with a particular focus on streamlined network maintenance, information handling, and network regulation [6]. For all of these reasons, it is a viable way to mitigate the issues facing older networks. It is good news, then, that SDN is growing in popularity and appeal all the time [20] (see Figure 2-1).

SDN is becoming an increasingly common element of cloud software and solutions, as well as commercial data centres. When it is applied, network managers have the opportunity to exert a greater influence over the movement of information. It also allows them to modify the attributes of routing and switching components within a network from a principal position [10]. As a result, it becomes possible to make random modification to routing channels. Furthermore, they get to enjoy an additional level of influence over the network information and can order goals and objectives in accordance with their urgency [13]. They may also choose to permit or deny specific data packets, depending on what they are trying to achieve. When control applications are handled as if they were software modules, a manager no longer needs to make decisions for each component in turn.

When SDN is considered from the viewpoint of cloud technology, it is clear that it can offer enormous advantages. For one thing, it allows cloud vendors to utilise a broad range of components. Usually, mainstream vendors like Amazon and Google have no choice but to source high end routers and switchers from one single supplier. This is so they can quickly reprogram the routing parameters and ensure that all of the components are compatible. Ultimately though, every system seems to have its strengths and weaknesses [8]. Certainly, no supplier is perfect, but purchasing from one source negates the need to apply custom settings to every new component. However, it is now possible to use SDN to reconfigure resource or routing allocations rapidly via a cloud vendor. In addition, it allows cloud users to get even more value out of cloud based tools and resources as well as to carry out tests and trials involving virtual flow slices. These are legitimate solutions, if all of the routers used by the company adhere to SDN requirements.

A communications network forms the backbone of any successful organisation. These networks transport a multitude of applications, including real-time voice, high-definition video and delay-sensitive data. Networks must provide predictable, measurable and, sometimes, guaranteed services by managing throughput, delay, jitter and loss parameters on a network.

QoS technologies refer to the set of parameters such as throughput, delay, PDR, and many others. The objective of QoS technologies is to make voice, video and data convergence appear transparent to end users. QoS technologies allow different types of traffic to contend inequitably for network resources. Voice, video, and critical data applications may be granted priority or preferential services from network devices so that the quality of these strategic applications are not degraded to the point of being unusable.

In network, the loss of data is one of the more detrimental effects of a network congestion problem. As an example, packet loss, where data packets are lost in transmission, causes poor video quality, bad VoIP service, and online gameplay to react slowly.

Accordingly, network congestion causes packet loss when an Internet route becomes saturated. The full Internet route allows no further data to be transmitted. The result is data packets will not make it to the receiving computer. In this study we try to enhance the QoS of the network to reach a higher quality service in network applications.

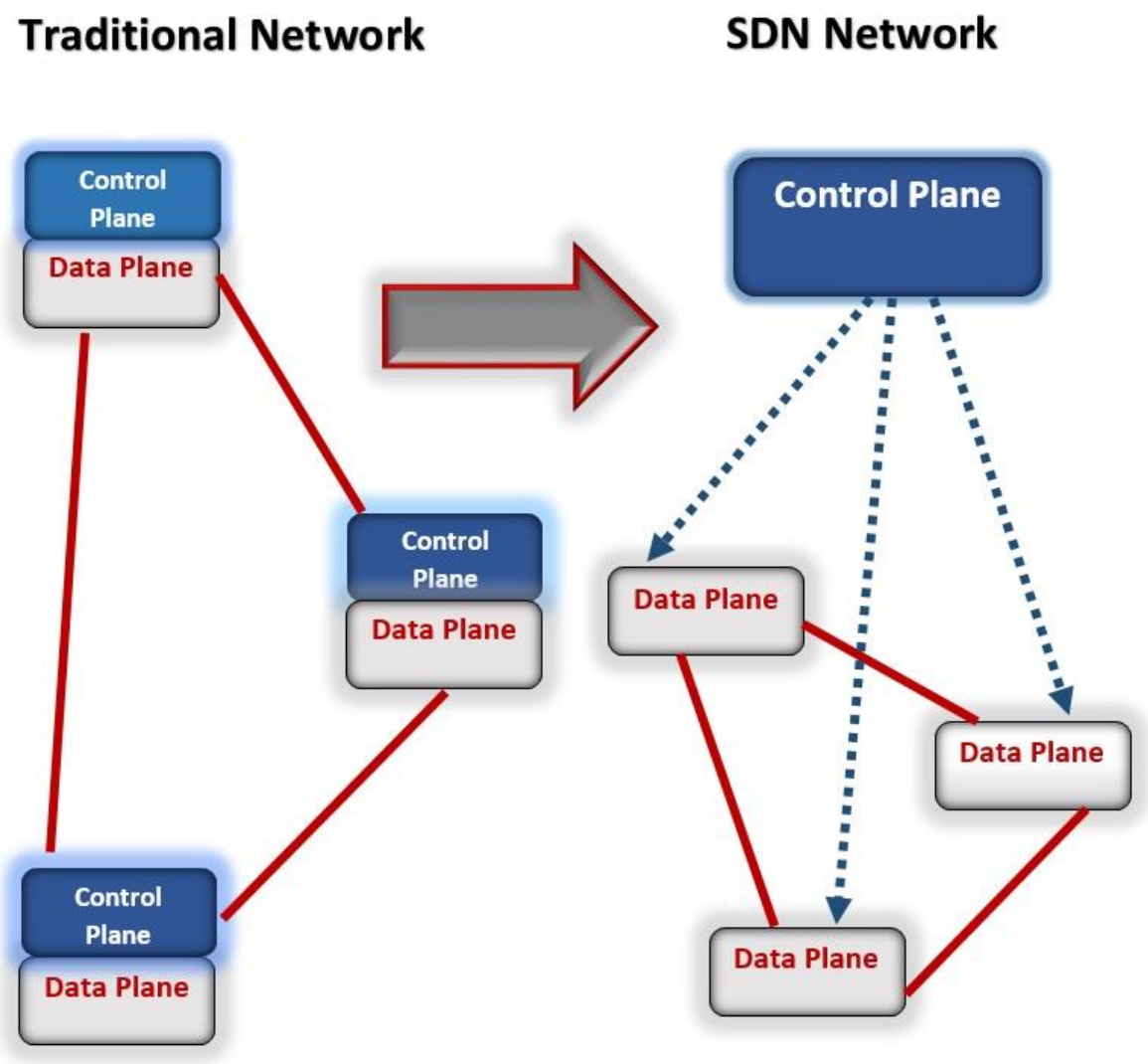

Figure 2-1 Comparison of a traditional network and SDN network

2.2. Software Defined Networking Architecture

The aim of SDN is to allow the network to transition to an explicitly programmable state and make the core infrastructure available for abstraction, as a way to streamline network services and applications further. The Open Networking Foundation ONF describes it as ‘an evolving network structure which separates the forwarding processes from the network controls [8].

SDN controllers always feature centrally positioned (logically) network cognisance, because it provides a broad, clear picture of the network in its entirety, which ensures efficiency, stability, and ease of use. When it comes to a network structure like this, the

cognitive features extract information from the networking components and regulate the network as a cohesive, unified system. This is achieved with the addition of the centralised controller. Normally, the controller operates via a remote commodity server and interacts across a secure connection, with the forwarding engines adhering to a series of standardised rules. The objective is to convert the infrastructure components into basic forwarding engines with the ability to receive and interpret inbound packets [21]. They do this according to a series of parameters created in a rapid, reactive manner by either one or multiple controllers. This occurs at the control level and relies upon a degree of previously established program logic.

Refer to [22], where ONF is provided as an example of a high end SDN networking structure divided into three vertical levels, as illustrated in Figure 2-2. The three illustrated layers are the Application layer and Northbound Application Programming Interface (API), Control layer and Infrastructure layer.

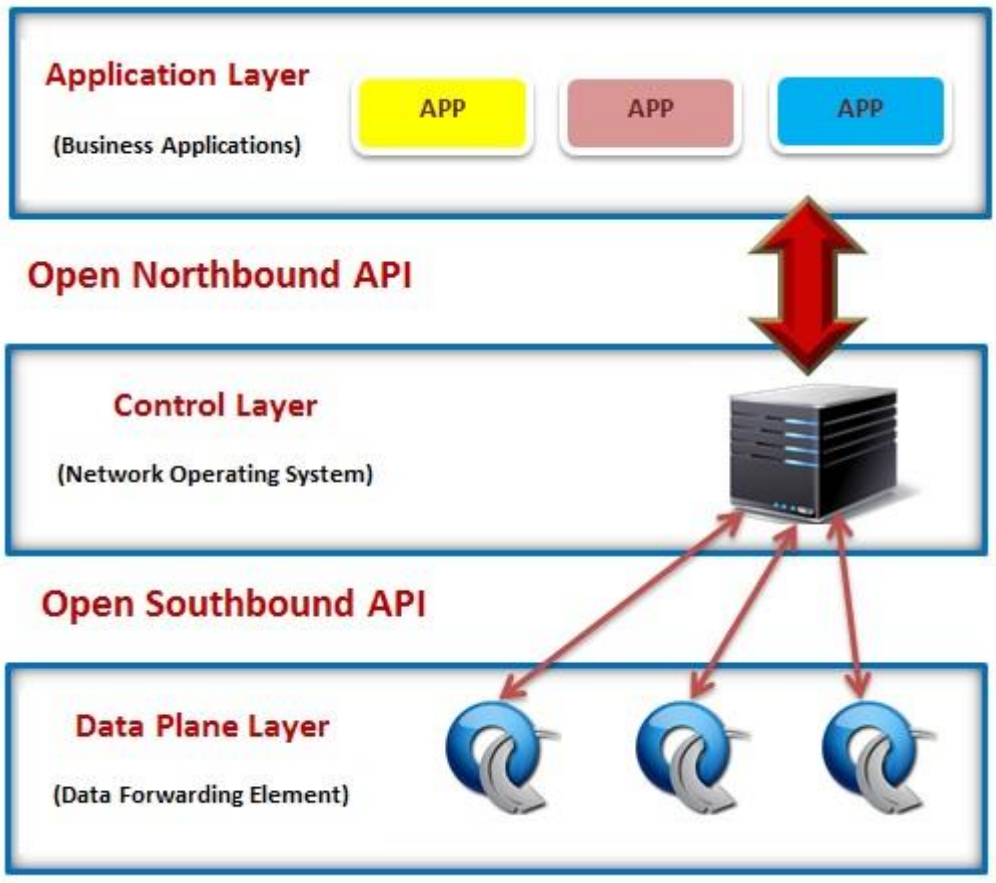

Figure 2-2 Block diagram architecture of a software defined network (SDN)

2.2.1 Application Layer and Northbound Application Programming Interface (API)

As shown in Figure 2-2 the application level is the most important for the SDN. It interacts with the control level, via the use of an Application Control Plane Interface (or ACPI), which in some literature is referred to as the 'northbound application interface' [23]. It plays an essential role, because it is tasked with managing and regulating software based security and business requests. Some common examples of applications like this include firewalls, network virtualisation, intrusion prevention systems (IPS), mobility management and Intrusion Detection Systems (IDS).

When it comes to exploring the application level, existing literature on the subject tends to take one of two approaches. The first involves focusing on the role of SDN applications as a way to regulate particular networking conditions (also known as 'use cases,' in this context). The second is concerned with the value of SDN applications for regulating particular network processes and objectives.

A) SDN Applications: Within this level, the SDN requests and processes communicate with the controllers as a way to fulfil precise network objectives. The processes can be grouped into different types of network ability or form. The most common are universal access control lists (U-ACAL), load balancing (LB), traffic engineering (TE), security, and quality of service (QoS). The overarching goal is to meet the demands and requirements of the network owner.

B) SDN Use cases: In contrast, some SDN applications are created to fulfil a particular purpose, while being subject to a specific scenario or set of conditions. Areas in which SDN has been used in this way include Network Function Virtualisation (NFV), Information Content Networking (ICN), cloud solutions, mobile networks, and network virtualisation.

The involvement of the northbound interface offers a stable, consistent way for network operators and application developers to utilise SDN services and carry out important network maintenance roles [24]. In some cases, it is compatible with a broad range of controllers. This is because a carefully constructed interface allows developers to build software that is fully separate from not just the data plane, but also the limitations of the

specific SDN controller server. The significance of the northbound interface will be explored in more detail later.

The beauty of SDN is that it can be used in conjunction with any type of conventional networking scenario (either personal or commercial), which makes it useful for individual users, businesses, data centres, and internet exchange facilitators. Due to its versatility, SDN is a common feature of many different types of network application. Currently, low to mid-level applications can carry out standard processes like load balancing, security implementation, and routing. However, they are also capable of supporting newer processes and perspectives. These include mobility regulation for wireless networks, network virtualisation, the minimising of energy consumption, failover and reliability processes relating to the data plane, QoS requirements, amongst other things.

One popular analogy is to look at network applications as if they are miniature brains, which have a great deal of influence over how forwarding devices operate. In fact, they are responsible for applying the regulatory logic that is eventually converted into requests designed for use in the data plane. To understand this better, it might be easier to stick with routing as an example, because it is quite a basic process. In this context, the logic of the application is concerned with uncovering the channel that packets will use to move from the point of transmission to the point of receipt (this channel can be called 1-2, say). To fulfil the objective, the routing application must, according to the topology parameters, determine which channel provides the optimal journey. Once identified, it can tell the controller to implement the relevant forwarding protocols to all forwarding components encountered along this channel (from point 1 to 2).

One of the most convincing reasons for widespread implementation of SDN is this enormous degree of versatility and variability. When brought together with reactive ‘use case’ allocations, the benefits for network management become clear. Even though the amount of possible networking conditions is practically limitless, the majority of SDN applications can be divided into one of five classifications [25]: (1) measurement and monitoring; (2) mobility and wireless; (3) data centre networking; (4) security and reliability; and (5) traffic engineering. Generally, the northbound interface is treated as a software API and not a protocol. When it is incorporated into a system, it becomes easier to utilise control plane features and services. There does not even need to be that much information about the core network switches for this to work.

2.2.2 Control Layer

The most essential part of the SDN structure is the controller, which is tasked with regulating and facilitating every single movement within the network. This is achieved by introducing flow entries to switch devices. It is also the most intricate component. The controller embodies the heart of the network, because it is the primary contributor to the Network Operating System (NOS). The applications are positioned at one end of the process and the network components sit at the other, with the controller linking the two. The control level is applied in the form of a server or a collaborative series of servers (referred to as 'SDN controllers,' collectively). The job of the SDN control level is to convert service commands from the application level into clear, practical instructions and requests. These are transmitted to the data plane switches and they give applications valuable information about their behaviour and conditions. As previously explained, the controller is in charge of identifying the optimal route and ensuring that it runs successfully [26]. Refer to Figure 2-3 for a demonstration of how this happens. The figure provides details on the key processes that all controllers are expected to offer and they can be listed as follows:

Shortest path forwarding: the controller uses routing data gathered from switches to identify the quickest channel;

Notification manager: the controller interprets, reconfigures, and passes on details about certain conditions to the application (these include security alerts, condition changes, and other types of scheduled/pre-set alarms);

Security mechanisms: the controller facilitates safe forms of separation and security among services and applications;

Topology manager: the controller constructs and regulates data relating to switch interconnection topologies;

Statistics manager: the controller uses the switches to gather information about traffic;

Device manager: the controller modifies switch protocols and features as a way to better regulate flow tables.

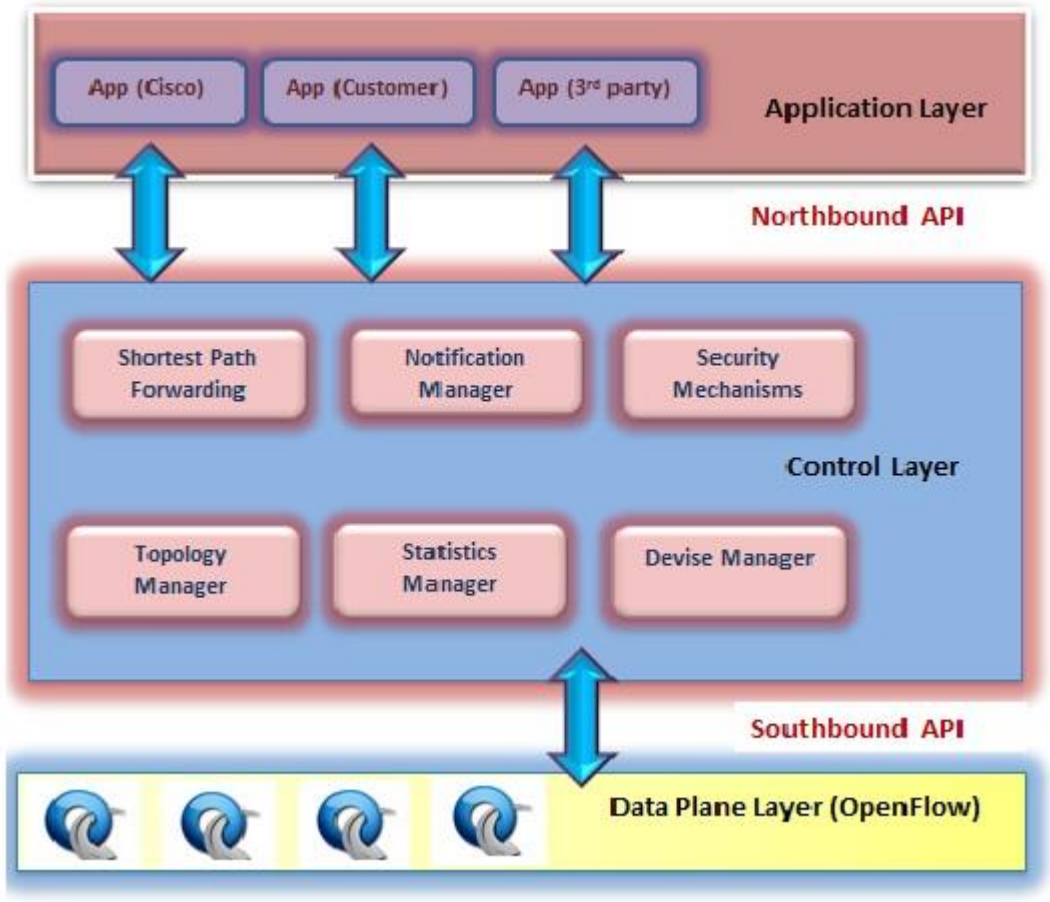

Figure 2-3 SDN control plane functions and interfaces

Often, interactions shared by the switches and the controller are kept safe with the use of Secure Sockets Layer SSL or a similar security standard. It should be pointed out that the forward level must apply the security feature, if it wants to bind clients to a Service Level Agreement. Another strength of the controller is its robust ability to determine optimal QoS conditions. It is able to manipulate explicitly the flow table as a way to distinguish between dissimilar traffic objectives and priorities. The network structure presented above must include the following three elements:

(A) **Supervision of Resources** – the forwarders must be able to watch and supervise the relevant network resources consistently. The controller expects and requires regular status updates. In some cases, the controller will specifically ask for these updates;

(B) **Signalling of Resources** – the forwarders are expected to interact with the controller via the use of signalling alerts. The controller needs data to carry out certain processes like upgrading the flow table or modifying the QoS policies. The most important information relates to the rate of resource consumption;

(C) **Allocating Resources** – in some circumstances, the controller asks the forwarder to hold specific resources back, usually for use at a later time. Common requests relate to the allocation of CPU calculation time, buffer size, and memory space, among other things [25], [27].

The primary role provided by the SDN controller is perceived as a type of Network Operating System (NOS). The purpose of an SDN NOS is to make it easier for the developers to formalise network protocols and to carry out efficient network maintenance without having to worry about the complex attributes of individual network devices. This is highly valuable, because these devices are often fast moving, broad ranging, and heterogeneous. In a similar way to traditional OS structures, an NOS offers developers abstraction of low end components, a range of key features and processes, and some shared application programming interfaces [28].

There are two main types of flow setup: reactive and proactive. When it comes to reactive setups, flow commands are established by the controller, if there is no entry present within the flow tables. The instructions are delivered immediately after the initial packet of flow arrives at the OpenFlow switch. Consequently, it is just this initial packet that stimulates interaction between the controller and the switch. Crucially, all flow entries become obsolete once a pre-established period of stagnation is complete, being then erased. While reactive setups are limited by a relatively slow journey, their inherent adaptability makes it easier to implement real time responses as well as to accommodate traffic load and QoS demands.

On the other hand, proactive setups are based on flow commands that are already present within the flow tables. This means that the setup is carried out well before the initial packet is received by the OpenFlow switch. The biggest benefit of this system is that there is no need to wait around for the controller to respond. Hence, it is faster and the controller only needs to be 'activated', if there is a problem or a flow command cannot be detected. The downside is that it can overload flow tables linked to the switches.

It should be noted that, while fine grained traffic movements (referred to as miniature or micro flows) are inherently versatile, they are not usually compatible with bigger networks. However, macro flows can be constructed by assimilating multiple miniature flows. They are a suitable option, because implementing a rougher grained movement allows the controller to make smaller, less impactful compromises between scalability and adaptability. To react to a flow setup command, the controller must first ensure that the flow

is compatible with the protocols attached to the application level. Once it has done this, it can determine which steps to take next. It is expected to identify an optimal channel for the flow and, where necessary, to implement new flow entries at every switch encountered on the journey. This applies to the switch responsible for generating the command as well. In accordance with the flow entries implemented, a design decision is made on the regulated granularity flow, which triggers the need for a compromise between scalability and adaptability. The controller prioritises by interpreting the conditions required for efficient network maintenance.

Refer to Table 2-1 for a list of existing controller applications and a brief description of each [29]. Currently, each controller listed (apart from RYU) is compatible with Version 1.0 of the OpenFlow protocol[1]. Also featured are two special function controller applications (RouteFlow [1] and Flowvisor [30]).

Table 2-1

Current Controller implementations compliant with the OpenFlow standard

Controller	**Language**	**Open Source**	**Created by**	**Description**
POX [59]	Python	Yes	Nicira	General, open-source SDN controller written in Python.
NOX [17]	Python/C++	Yes	Nicira	The first OpenFlow controller written in Python and C++.
MUL [60]	C	Yes	Kulcloud	OpenFlow controller that has a C-based multi-threaded infrastructure at its core. It supports a multi-level north-bound interface for application development.
Maestro [21]	Java	Yes	Rice University	A network operating system based on Java; it provides interfaces for implementing modular network control applications and for them to access and modify network state.
Trema [61]	Ruby/C	Yes	NEC	A framework for developing OpenFlow controllers written in Ruby and C.
Beacon [22]	Java	Yes	Stanford	A cross-platform, modular, Java-based OpenFlow controller that supports event-

				based and threaded operations.
Jaxon [62]	Java	Yes	Independent Developers	a Java-based OpenFlow controller based on NOX.
Helios [24]	C	No	NEC	An extensible C-based OpenFlow controller that provides a programmatic shell for performing integrated experiments.
Floodlight [38]	Java	Yes	BigSwitch	A Java-based OpenFlow controller (supports v1.3), based on the Beacon implementation, that works with physical- and virtual-OpenFlow switches.
SNAC [23]	C++	No	Nicira	An OpenFlow controller based on NOX-0.4, which uses a web-based, user-friendly policy manager to manage the network, configure devices and to monitor events.
Ryu [63]	Python	Yes	NTT, OSRG group	An SDN operating system that aims to provide logically centralised control and APIs to create new network management and control applications. Ryu fully supports OpenFlow v1.0, v1.2, v1.3, and the Nicira Extensions.
NodeFlow [64]	JavaScript	Yes	Independent Developers	An OpenFlow controller written in JavaScript for Node.JS [65].
Flowvisor [48]	C	Yes	Stanford/Nicira	Special purpose controller implementation.
RouteFlow [66]	C++	Yes	CPqD	Special purpose controller implementation.

2.2.3 Eastbound and Westbound

Typically, SDNs feature either a distributed or a centrally positioned control plane. As Figure 2-4 shows:- west/east bound APIs represent a unique type of interface, which is

sometimes needed by distributed control planes [5]. Presently, every controller is responsible for applying its own west/east bound APIs. The roles carried out by the interfaces cover things like algorithms for data stability frameworks, observation/alert functions (for example, testing whether a controller is operational or warning of a takeover from a series of forwarding components), and export/import movements among controllers. It should be noted that controller to controller interactions are not usually determined by OpenFlow protocols.

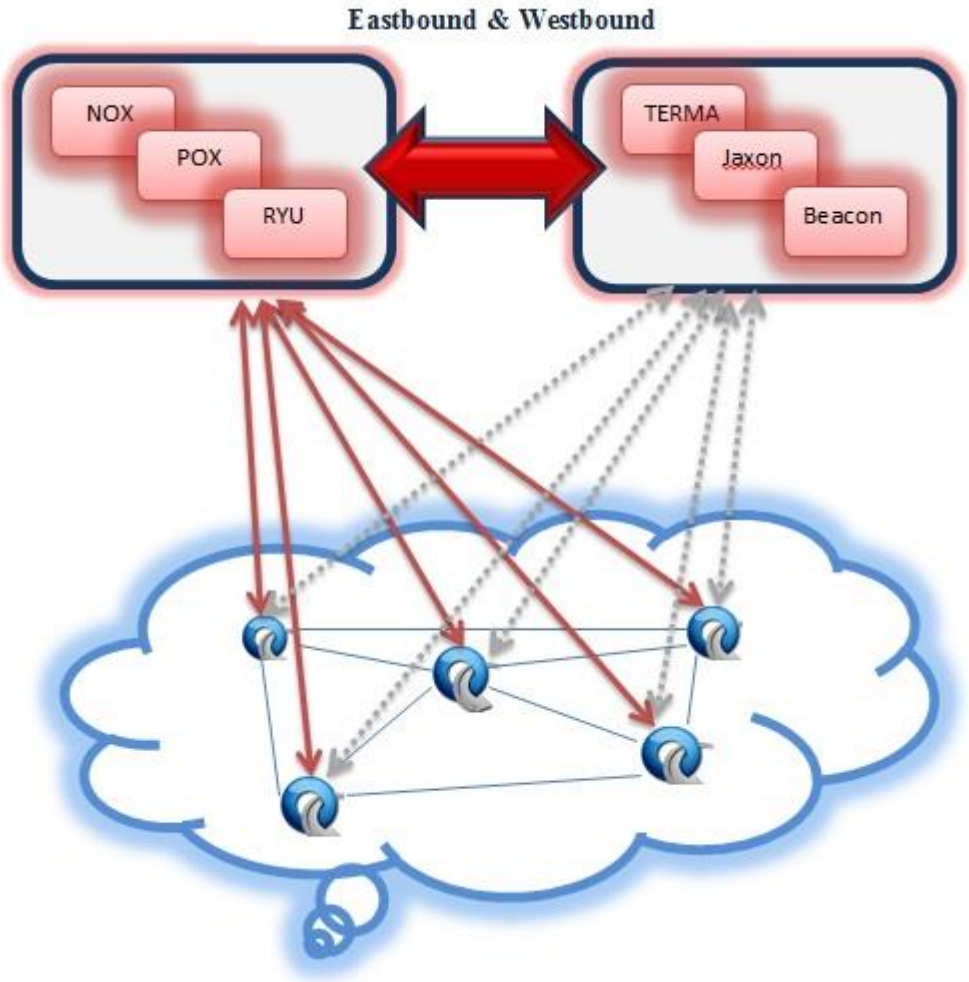

Figure 2-4 Distributed controllers: east/westbound APIs

2.2.4 SDN infrastructure Plane

Most SDN structures share common features with conventional network arrangements. They consist of networking components (middle box tools, switches, routers, etc.) and the key thing that sets them apart is the low level nature of the older systems. The conventional physical components are utilised as basic forwarding resources providing no inherent influence and there is no software capable of making instant, informed choices. As a result, the network intelligence gets withdrawn from the data plane components and placed in a centrally positioned (logical) control structure (refer to Figure 2-2). This particular strategy guarantees interaction and regulation efficiency, as well as smooth connections between the control and data plane components. Crucially, the new networks that are constructed get positioned above standard and open interfaces [21].

When it comes to OpenFlow/SDN structures, two constructs are very important. As has been discussed at length, the network relies heavily on its forwarding components and controllers and there is a representation of this in Figure 2-5. OpenFlow compatible forwarding components are structured according to an assortment of flow tables, with every entry in a table being made up of three aspects: (a) a common principle or instruction; (b) commands to be carried out by reciprocal packets; and (c) counters that record information from the reciprocal packets. The term ‘data plane component’ refers to a specific type of software or hardware resource, which is designed to handle packet forwarding. On the other hand, controllers are a type of software arrangement (or a ‘network brain,’ to return to a previous analogy) operated via a commodity hardware foundation. To put it another way, open interfaces allow controller components to configure heterogeneous components quickly and accurately. This is a strength that is highly valuable, because older networks tend to struggle as the range of closed and proprietary interfaces is just too vast and the distribution of the control plane is too sophisticated. The figure outlined below is one of the most widely recognised representations of an SDN data plane component [25].

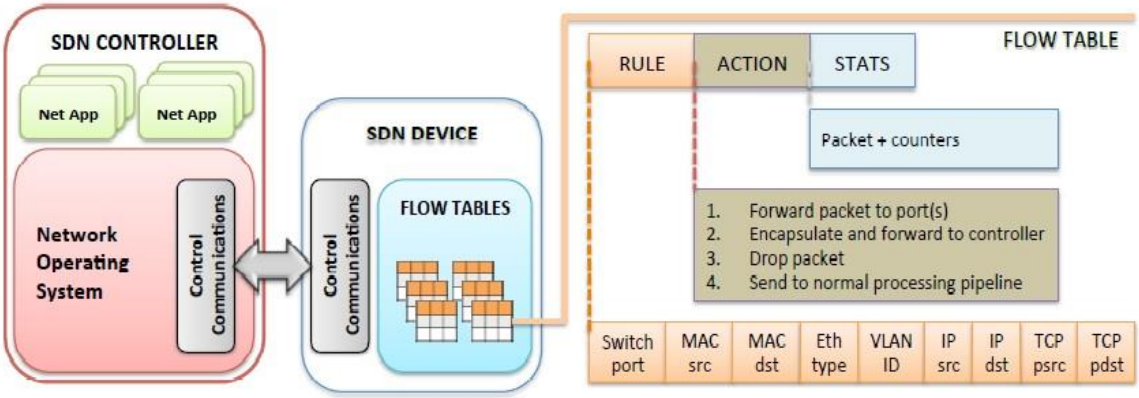

Figure 2-5 OpenFlow-enabled SDN devices

In accordance with the associated table, actions are taken and decisions are made as soon as a packet emerges. Within the OpenFlow component, a channel moving through the arrangement of flow tables tells the packets how they will be processed and transported. These actions are ceased when a match is found within one of the flow tables or it is determined that there are no compatible matches and therefore, no viable commands. The urgency of the commands is determined by the established arrangement of numbers within the tables and the row positions.

Three potential outcomes are as follows: (a) deliver the packet to an outbound port; (b) modify it and deliver it to the controller; or (c) eliminate it completely. If no standardised command or action can be found, the packet is eliminated. It is important to point out though that it is also possible to apply a standardised command, which instructs the switch to deliver the packet to the controller [31]. Sometimes, it is possible to construct a flow command by bringing together several distinct but compatible fields (refer to Figure 2-5).

2.2.5 Southbound Protocol

Southbound APIs (also known as southbound interfaces) provide a strong link between forwarding and control components. Accordingly, they play a vital role when it comes to keeping the data and control planes communicating without compromising their standalone nature. The problem is that these interfaces are rigidly bound to the forwarding components within the core virtual or physical framework. It is not unusual for new types of switch to undergo many years of testing and preparation before being introduced to the

market [32]. The build time is even lengthier if they are being constructed from the ground up. In fact, the update rotations and cycles alone can last as long as eight to ten months. For this reason, investing in such a project can be hazardous and challenging. The reality, however, is that southbound interfaces are an integral part of emerging network capabilities [33], so they do need to be made more accessible and stable. OpenFlow is just one example of an alternative, improved interface and like many others; it has been championed by networking specialists and experts.

As stronger, faster interfaces are developed, levels of interoperability will increase and it will become easier for network managers to apply vendor agnostic components. In fact, this is already happening as a result of innovative technologies like OpenFlow. Currently, the system is highly regarded and hence, is one of the most broadly utilised southbound interfaces for developers working with SDN. Yet, OpenFlow is not alone, for is certainly not the only southbound interface compatible with SDN. Some other API structures include POF [32, 33], Hardware Abstraction Layer (HAL) [34], [35], ForCES [36], OpenState [37], Programmable Abstraction of Data Path (PAD) [38], OVSDB, Revised Open Flow Library (ROFL) [39], and OpFlex [40]. Nevertheless, one of the primary advantages of OpenFlow is that it offers a shared protocol for the incorporation of OpenFlow oriented forwarding components and the smooth maintenance of interactions among control and data plane constructs (controllers and switches).

More specifically, the protocol presents three key data sources, these data channels being the primary method for transmitting valuable information throughout the network. The first (1) involves the movement of statistical information from the forwarding components to the controller. The second (2) sees packet data being delivered to the controller (by forwarding components) in the event that it is not clear how to handle an imminent flow or because there is a direct command to ‘deliver to controller’ attached to the flow table entry. Finally, the third (3) involves event related data being delivered to the controller only if a shift in conditions is detected.

2.3. OpenFlow Specification

The structure of OpenFlow interfaces is made up of three key elements (refer to Figure 2-6): a robust flow channel; a controller; and a compatible switch. The switches depend on flow tables to move their packets. The term ‘flow table’ simply refers to an arrangement of flow entries or priorities. As part of the OpenFlow system, there is a standard

that enables software applications to configure the flow tables relating to various switches. Counters are used as a way to record statistical information about moving packets. The entries all contain data relating to compatible counters, commands, and fields. Packets that are inbound are checked against these credentials and if a match is identified, the packet is handled exactly as the entry command instructs. In some cases, even if there is no match, the packet is still delivered to the controller.

The controller is a type of software system tasked with modifying the flow tables of switches. This is achieved via the OpenFlow principles and standards. The switches and controllers are linked by a robust interface and the packets move within its parameters. The interface helps to regulate the switches, process packets, and to make sure that outbound packets are transported to the right location. If there are interactions between the controller and the switch, it is because the flow table has sanctioned them [41]. From an internal perspective, switches employ Random Access Memory (RAM) and Ternary Content Addressable Memory (TCAM) to interpret packets. All switches that are compatible with OpenFlow need to have the ability to forward packets based on the commands provided by the relevant flow table. Refer to Figure 2-7 for a detailed representation of how packets are handled by network components.

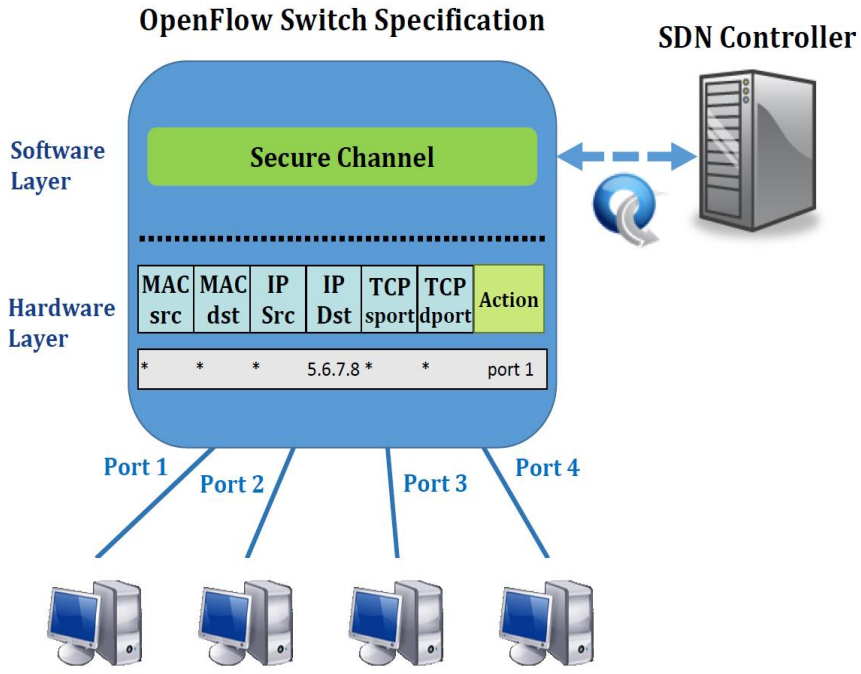

Figure 2-6 Components of an OpenFlow 1.1 switch

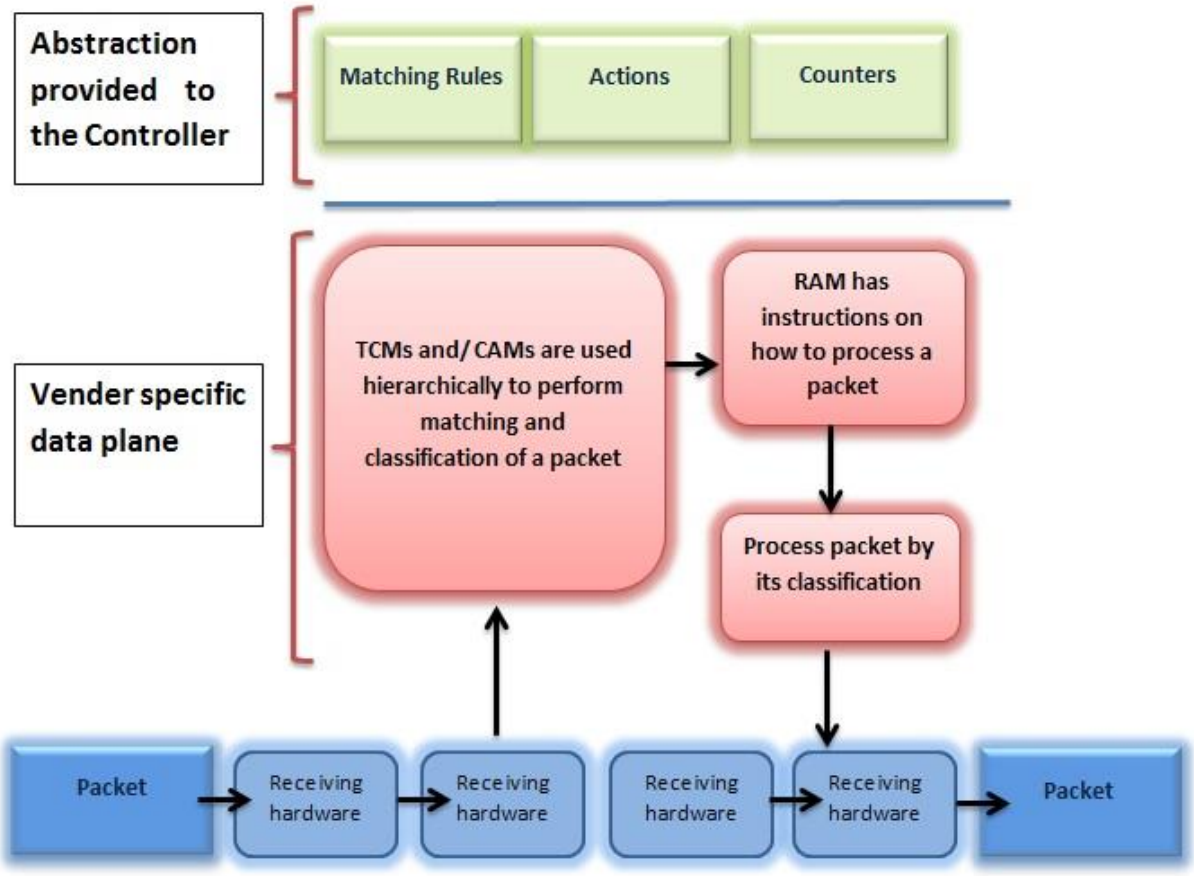

Figure 2-7 Elements of an OpenFlow-compliant switch

A. OpenFlow 1.0

The benefit of the OpenFlow framework is that it precisely defines the matching algorithm and packet parsing requirements. The packet compatible algorithm begins by checking the VLAN and Ethernet fields against one another. They may then be checked against IP header fields, if more information is needed. If the IP type signals TCP or UDP, the corresponding transport layer header fields are considered. Several actions can be set per flow. It is possible to combine IP packets and Ethernet by searching for compatible data within the destination and source address. Similarly, VLAN and Ethernet based fields are able to be brought together for the purposes of Ethernet, Explicit Congestion Notification (ECN), and differentiated services (DS). In addition, combining UDP and TCP destination and source port numbers is a viable option too. In 2009, Version 1.0 of OpenFlow was

introduced to the market [42], being now the most broadly employed interface and common to networks right across the world.

The most significant activity takes place during forwarding behaviours, which is because they transport the packets to either one targeted port or they deliver it to all of them. In some scenarios, the header fields of the packet are edited and this might include tweaks to the IP source, VLAN label, or destination address. As already discussed, there are other conditions that lead to the elimination of the packet as well. When a packet is disregarded, the controller allows for the application of network access control via the OpenFlow interface [43]. It is able to ask the switch to secure all packets within the flow and to deliver them to the controller, even if there is no specific command to do so.

Crucially, statistical records and monitoring are the responsibility of the counters in the switch. However, the controller has the authority to challenge this information. It has the right to question statistics that provide specific entries relating to active and delivered packets. Statistical records pertaining to the packet flows are kept safe within the flow tables. There is also accessible information about the number of packets per queue and per port. Switches working in conjunction with Version 1.0 of OpenFlow utilise a total of twelve header fields: Ether type; Ingress Port; IP ToS bits; Ether src; TCP/UDP dst port; IP dst; Ether dst; VLAN id; VLAN priority CoS; IP src; IP Proto; and TCP/UDP src port. They are displayed as part of the payload and header sections of the Ethernet packets moving towards the switches.

One or several header fields can be used to find a matching flow entry and packet. Crucially, a field within the flow table may present the value of ANY and be compatible with every packet. Consequently, ANY can be applied to the switch hardware, via the third masking protocol of the Ternary Content Addressable Memory, if indeed, the forwarding table has been handled according to TCM principles. During the first phase of the process, the Ethernet packet moving into the switch gets sent to a packet parsing point. Then, during phase two, the header fields are taken and inserted into a packet lookup header, where they will be employed as a matching tool. The third phase involves the lookup header being delivered back to the packet matching tool. Following this, the header is checked against the commands associated with each flow entry contained in the flow table. It is important to recall that the entries are arranged according to their degree of urgency [41], which is why the lookup header is checked first, beginning with the initial entry in the table. When a match is identified, phase five sees the packet being handled according to the command provided by

the table. However, if no match exists, the initial two hundred bytes of the packet are delivered to the controller for interpretation and further handling. Figure 2-7 demonstrates the primary parts of an OpenFlow compatible switch, whilst Figure 2-8 presents the associated information from the data plane attached to this switch.

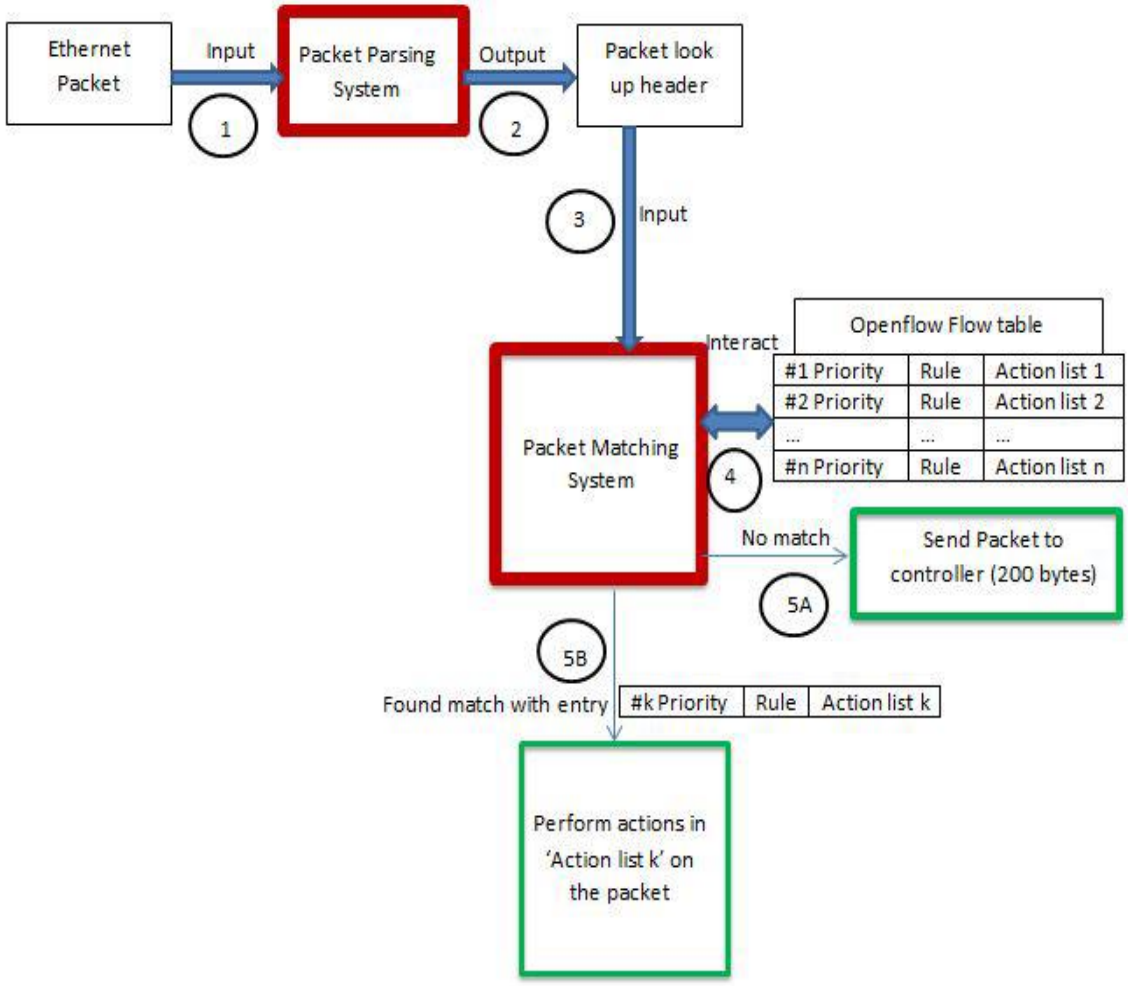

Figure 2-8 How a packet is processed and forwarded in an OpenFlow 1.0 switch

B. OpenFlow 1.1

In 2011, Version 1.1 of OF was introduced [44] and within this system, multiple flow tables are pipelined. It uses fields arranged into a total of fifteen tuples. Additional data is provided in the form of meta descriptions and tags and they can also be shared among different tables. MPLS tagging is facilitated with the use of two more data fields. Other added support includes: Support for VLAN and QinQ, adding; modifying and removing VLAN headers; Support for virtual ports and tunnels; Multipath routing; and ECMP. Instead of

depending on traditional commands, it utilises Instructive Content, which contains a greater amount of detail and more insight into how the packets should be treated or changed. When a packet reaches a switch, it begins to search for a matching entry in Table Zero. Presuming that such an entry can be identified, the packet is handled precisely as the commands in the entry describe. For example, the command may be in the form of a 'Deliver To' request and this might direct the packet to a different table. In some cases, the flow commands direct the packet to a series of tables, each with other own entries. However, this usually only occurs when there is more than one flow that needs handling.

C. OpenFlow 1.2

Version 1.1 of OF was introduced in early 2011. It was quickly followed, in late 2011, by an updated system called Version 1.2 [24]. The new version allows the switches to link up to more than one controller at any one time. The result is a more robust system, with a smaller risk of faults, and superior load balancing. It provides extra support for resources, like extensible matches, type-length-value TLV functions, and IPv6. The introduction of TLV increases the overall versatility of OF, because it means that the fundamental protocols are no longer limited according to size and arrangement.

D. OpenFlow 1.3

The updates didn't stop with Version 1.2, however. Version 1.3 was introduced in early 2012 [39]. This newer system is an improvement because it incorporates metre charts, which make it easier to apply QoS processes, such as rate restrictions. Furthermore, a cookie field has been attached to all messages containing packets that are bound for the controller. The aim is to facilitate more efficient searches. The more information that a controller receives, the faster it can decide what to do with the packet. Now, each flow table is required to keep a table miss record, which provides detailed instructions on how best to handle a packet with no identifiable match. For example, the command may be to eliminate the packet, it could be delivered to the controller or possibly directed on to a different table to continue searching for a match.

Earlier iterations of OF allowed switches to link up with more than one controller, if it helped with load balancing and error tolerance. Now, switches can construct auxiliary links and use them to strengthen the primary links which they have with the controller. Version 1.3 is a good example of how an interface can maximise the parallelism capabilities inherent within the majority of switches. In addition, Version 1.3 incorporates 'per connection' event

filtering as way to enhance and expand the degree of controller support. The idea is that every controller is able to remove event related data that it deems invaluable.

E. OpenFlow 1.4

The most recent iteration and upgrade of OF is Version 1.4, which was introduced in late 2013 [2]. There is more support for optical ports, because there are updated port attributes and standards. They incorporate data fields designed to aid the tracking and programming of delivered/received frequencies. One of the main differences, however, is that TLV functions play a greater role in some aspects of the system. Within older iterations, when the flow table became full and overloaded, the switch would transmit a fault alert to the controller, which would tell the latter that action was needed. The problem is that the process is not very efficient. It can take a relatively long time for the controller to be alerted and for an appropriate response to be taken. Fortunately, Version 1.4 of OF features an all new 'Eviction' function. It gives a switch the power to disregard and remove entries with a lower degree of urgency, if the table is so full that immediate action is required.

Moreover, switches can now warn the controller if it detects that a table will be overloaded soon. This can only be achieved, however, if certain parameters and boundaries are established early on [45]. For example, the controller first has to determine what counts as 'too full', so that an alert is sent when this number is surpassed or almost surpassed. In addition, a new bundle feature supports the quasi-atomic completion of multiple commands. Finally, the fault alert is a type of symmetric trigger, which can be launched by the controller or the switch if the conditions warrant it. For the purposes of the recommended system in this thesis, POX controllers will be used combined with Version 1.0 of OpenFlow, because the vast majority of SDN controllers are compatible with this interface (apart from RYU) [5].

2.4. Centralised Control of the SDN

As aforementioned, when it comes to SDN structures, the data plane gets separated from the control plane (which is closely associated with the controlling components of routing technologies). Within the data plane, there are flow tables designed to identify the optimal outbound port for every inbound flow. Consequently, every control command or preparation from the router is gathered and accumulated at the heart of the controller. The controller is tasked with the job of maintaining and directing network processes. For a

centrally positioned controller, an extensive understanding of the system is essential and multiple OpenFlow switches may be linked to just one controller. This is how cohesive, centralised actions are taken [27]. Rather than relying on multiple network components with an insufficient understanding of the system, a single knowledgeable controller can formulate responses according to its familiarity and awareness of the network. Thus, one of the biggest benefits of incorporating a centralised controller is the fact that maintenance and the location of key data is the responsibility of one easily handled central point. The result is a more organised, more efficient network.

The standard SDN is software oriented [5]. In other words, all network intelligence is centrally positioned at the heart of controllers. The network is skilfully maintained by a program operated according to a centrally positioned controller, which is achieved by modifying the packet delivering processes within the core network components. The principles used to interpret and handle the packets are incorporated into the network components as a way of supporting the duties required for sustaining a network. These duties might include server load balancing, general network upkeep, routing, access regulation, and traffic optimisation. Ultimately, the control logic is shaped around a much more universal perspective, instead of being a primarily localised, distributed resource.

The way that SDN configures and controls network traffic is via a console with centrally positioned controllers. It saves an enormous amount of time and resources, particularly when it comes to network updates and system maintenance for very big companies. For, all granular/regulatory modifications to the core network can be carried out from one central location. This is particularly desirable because it negates many of the challenges associated with programming separate switches. Some other advantages of using a centrally positioned controller are outlined below.

Automation and Virtualisation – the reality is that centralised functions and comprehensive automation being incorporated right at the heart of SDN is a great way to boost productivity, integrity, reliability, and scalability. One of the biggest advantages of SDN is the fact that network virtualisation constructs a cohesive framework right across every piece of associated software, which includes the routing and switching components too. The result is that features and processes can be modified at speed and without incurring high expenses. Plus, the network manager no longer needs to find a way to access the service layer just to make a quick alteration and this results in a much easier to use network.

Increased Speed and Flexibility – the construction of VMs (virtual machines) makes it substantially easier to create fully functioning SDN structures.

Network Scalability and Unified Maintenance – there is more freedom to try innovative system configurations and arrangements. The power of SDNs is that they enable network managers to bypass the restrictions created by Simple Network Management Protocols.

Broader, More Granular Security – SDNs are a simple way to implement robust security, even for end systems, apps and Bring Your Own Device (BYOD) scenarios. This lattermost condition is particularly important for office based businesses, as it can be difficult to keep track of network authorised component and devices. This is advantageous, because virtual machines present a number of new security issues and limitations.

Adaptability – with the support of enhanced adaptability (as a result of SDN structures), businesses are able to construct customised networking services safely without the need for anything more sophisticated than basic development resources.

Ease of Use – as a result of greater network configurability, businesses are able to create innovative applications with the potential to keep enhancing assimilation, interaction, ease of use, and a host of other valuable attributes. Ultimately, a company does not have to rely on external resources to improve any of these factors.

Centrally Positioned Automation and Regulation – the term 'Deep Packet Inspection' refers to a type of networking program, which scans packets for signs of faults, errors and unwanted behaviours. So, for example, it is a good way to detect spam, viruses, and other forms of unstable activity. Generally, any action which deviates from the matched command in the flow table can potentially be identified as unwanted or dangerous to security. It is one of the most effective ways to implement sophisticated network automation, security structures, and non-compliance policies. When SDN and DPI are brought together, they further the centralised management of network protocols and facilitate fast, successful automation. To reiterate, automation and streamlined policy management can be implemented across the entire network. The support of a centrally positioned DPI is a recommended way to transmit data to all of the affiliated network components, as opposed to expecting every single one to handle its own DPI requirements. The benefit of this is that the network can be perceived as a unified, cohesive entity, rather than a series of distinct components, when increasingly in depth information about network traffic is collected.

Virtual Pressures – it should be noted that the combination of physical and virtual systems usually leads to greater versatility, scalability, and reliability. SDN gives network managers the opportunity to set up more than one network, rather than always having to modify individual networking components manually to meet changing conditions [26],[4]. In fact, the highest quality control strategies have the power to allocate bandwidth for the specific purpose of moving data traffic to the right locations.

2.5. **Traffic Engineering Analysis**

Software Defined Networking represents the next generation of network arrangements and interactions. Consequently, it has an enormous amount of potential and it could end up being used to do some very sophisticated things. As aforementioned, it is based on the primary goal of isolating the control and forwarding levels within a network structure. This is particularly valuable when it comes to systems, whereby network managers can configure packet forwarding responses to enhance the development abilities of applications substantially.

One of the most helpful types of network application is Traffic Engineering, which scrutinises the quantification and maintenance of network traffic. It is also used to construct optimal channel processes and direct traffic more skilfully so as to maximise the value of key resources [2], [46]. Overall, the goal is almost always to improve quality of service and to help the network fulfil user demands with less effort and strain. When analysed next to older, more conventional systems, it is clear to see that SDN is a marked improvement.

To start with, the fundamental principles of flow management within SDNs dictate that, when a flow reaches a switch and it cannot identify any matching commands, it is handled with the following four step process. Initially (A), the ingress switch delivers the first packet of the flow to the controller and then, (B) the controller determines the right forwarding channel for the flow. Next, (C) the controller directs the flow to the correct flow tables along the optimal channel and finally, (D) any other packets linked to the flow (sometimes from different flows with matching features) are moved across the data plane and require no further contribution from the control plane. It should be noted that forwarding protocols can take some time to be established, so there is likely to be some degree of inactivity. In the event that the accumulated traffic contains a large amount of new flows, a substantial overhead may be generated across the data and control planes. Figure 2-9 demonstrates the four core thrusts that modern traffic engineering processes are concerned

with and shaped to fit, these being: (1) topology upgrades; (2) flow management; (3) traffic evaluation (incorporating characterisation); and (4) error tolerance.

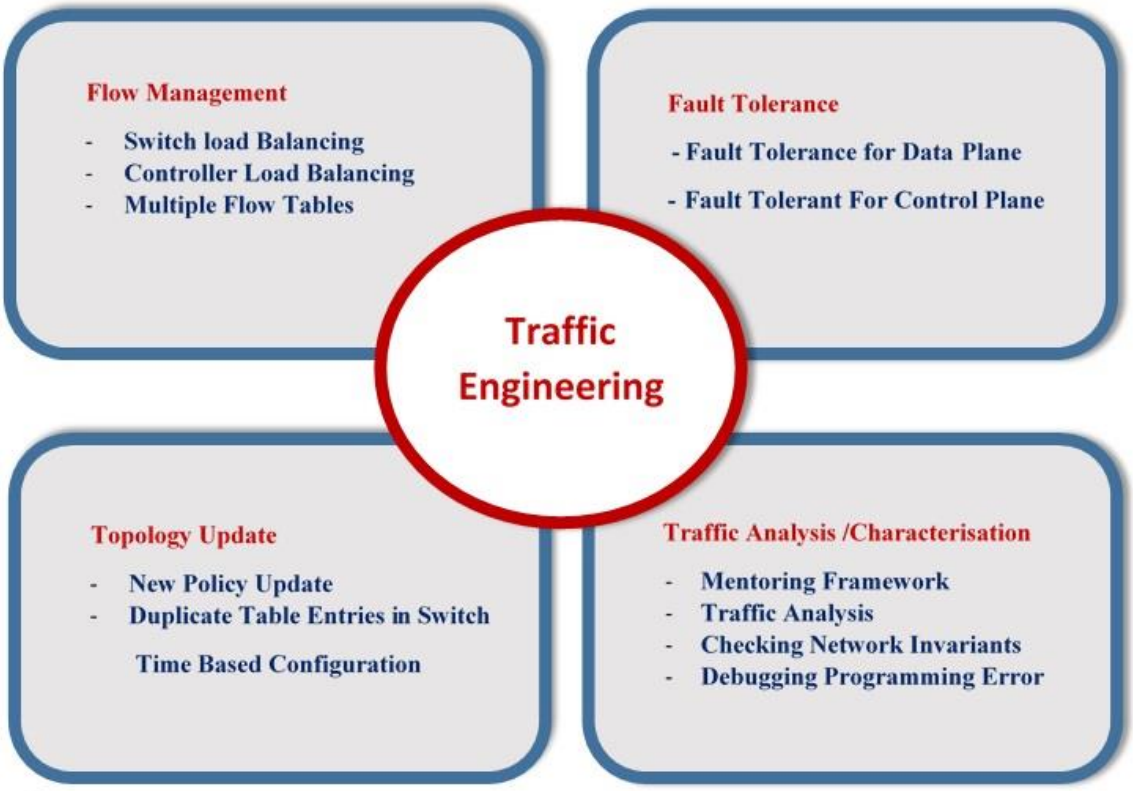

Figure 2-9 The scope of traffic engineering approaches in current SDNs

The issue of unwanted inactivity may be solved by using traffic engineering strategies to enhance flow maintenance and efficiency. If this is the case, TE needs to be structured in a way that allows it to minimise the compromise between load balance and dynamism [47]. Furthermore, to guarantee optimal network integrity, SDN must be able to carry out failure recovery processes in an open, clear, and skilful manner. The goal and general rule of thumb is to respond to and solve node and individual link faults within a fifty millisecond window. Version 1.1 of OF has attempted to enhance overall networking integrity by incorporating a rapid response system for node and link faults. It allows a substitute channel to be allocated, so that the switch can alter the forwarding route without having to consult the controller first. Yet, it should be explained that, even though improvements have been made with the introduction of centralised approaches, getting fault response times right is a big task. To achieve it, the central controller has to identify alternative channels and warn all of the relevant switches about the fault before it can take any kind of corrective response, with all of

these processes taking time. Nevertheless, the standard of failure responses has to be extremely high every time that an error develops (whether it is associated with links, controllers, switches, or anything else) [48]. Not only this, but all fault responses must take into account the restrictions on flow table resources and memory within switches.

The topology modification processes of the SDN are primarily concerned with forecasted shifts and alterations (like tweaks to network protocols and regulations). They are much better at handling changes to conditions that they have been able to prepare for, but this doesn't really help when it comes to fault tolerance. As centralised controllers regulate each of the switches featured in an SDN/OF network via fast paced adjustment of the universal network principles, a reasonable degree of stability and predictability is important for all of them. This allows separate packets and flows to be managed either by the original rules or by the updated ones, but it makes sure that the two do not clash and try to influence the same entity at the same time. While rules and regulations are being changed and improved, there is a risk of more flows being disregarded and eliminated. Others are likely to be disrupted and reach their intended destinations at a later time than predicted, which ultimately is bad for the integrity of the network and it can lead to a hugely inefficient use of resources. Ideally, the fault response needs to be carried out in as close to real time as possible for the system to be successful. It is not an easy thing to achieve, particularly in accordance with a very big SDN/OF system. Often, there are some switches that cannot be directly linked to the central controller and this is a tricky limitation. Thus, the major obstacle for topology upgrades is determining how best to ensure that the SDN controller is able to modify the network without compromising on quality, productivity, integrity, or efficiency.

Finally, all traffic evaluation tools need to offer suitable methods of traffic tracking and reporting, invariant testing techniques, analytics/extraction of trends/attributes, flow/state data procurement, fault debugging functions, and more. In fact, traffic tracking and recording processes are some of the most essential elements of broader traffic evaluation. They are a great way to identify link and network faults and prepare for link overloads and 'pile ups.'

One particularly pressing concern is the amount of SDN networks, which continue to utilise older, less efficient flow oriented tracking processes [49]. Many of these have more in common with the earlier forms of IP network. As such, they can result in inflated monitoring costs and a much greater rate of resource consumption than is preferred. Plus, there is plenty of evidence to suggest that introducing a controller with an overly intricate tracking and monitoring system will only end up unnecessarily obfuscating the network. While Version

1.3 of OF did launch upgraded flow metering functions, many of the existing controllers (Floodlight, NOX, POX, etc.) and easily accessible switches fail to offer sufficient support for variable flows and diverse, dynamic statistical data [50]. Consequently, there is a need for improved, enhanced tracking features, which should be as simple as possible, offer ease of use, demand minimal expenses as well as reliable and precise traffic calculations.

2.6. **Dynamic Update of the Forwarding Rules**

OpenFlow networks provide another distinct advantage, whereby they facilitate the fast paced modification and improvement of forwarding protocols. Crucially, no form of human intervention is needed and the controller has the power to alter entries in the flow tables at any point. When the system is correctly configured, it can handle a huge variety of topological alterations at quick speeds. The actions are made according to judgements elicited by the software controller, which is the 'brain' of the network.

The next section will explore the interactions between OpenFlow controllers and switches. The OpenFlow framework is based on three core ideas [51], these being: (A) the network is reinforced and supported by switches that are compatible with the OpenFlow interface; (B) the control plane is made up of a single one or multiple controllers; and (C) the control plane is linked to the switches via a robust control channel. When the phrase OpenFlow compatible switch is used, it refers to a simple forwarding component, which directs packets as it is told to by the relevant flow table entries. Each table contains a series of entries, with its own match credentials, commands, and requests. In some contexts, these entries are known as 'flow rules.' Every entry is attached to a header field and this determines which packets are a match and which are not viable for the table. They consist of a wildcard-capable match over specified header fields of packets. Ternary content addressable memory TCAM is an important part of supporting rapid fast wildcard matches.

The purpose of the counters is to gather statistical data about the flows [51]. They record the amount of packets processed and the time it took for the delivery to be completed, whilst the 'actions' refer to what commands were used to deal with a particular packet. For instance, the most useful commands are 'Deliver To,' 'Eliminate,' and 'Alter Field.' The switch needs Ternary Content Addressable Memory if it is to facilitate the rapid transportation of packets to their intended locations. In accordance with the specific OpenFlow policies, the header fields can prioritise different types of matching credential (for example, MPLS, IPv4, IPv6, etc.). Refer to figure 2-10 for more details.

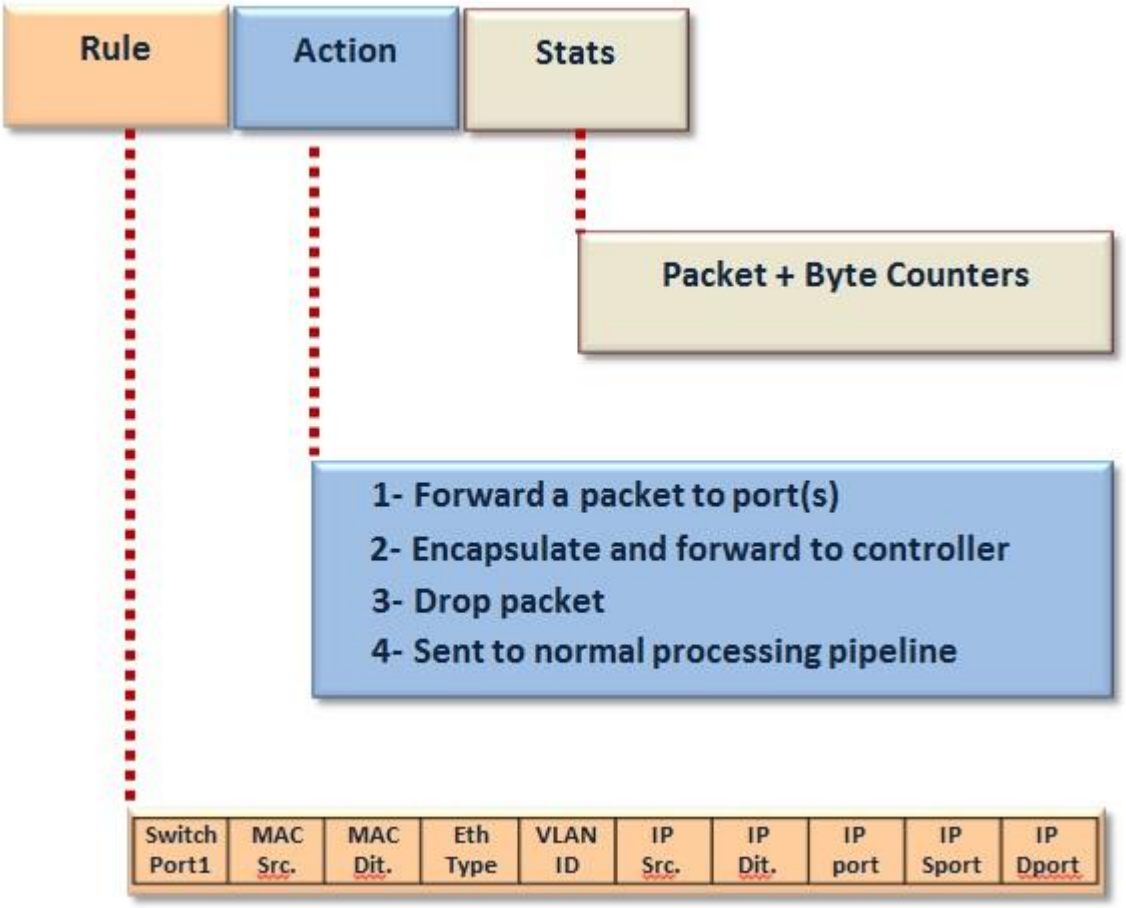

Figure 2-10 Flow table entry for OpenFlow

The controller is a software based resource tasked with the job of filling and regulating the flow tables of each switch. It has the power to alter the forwarding actions of a switch by eliminating, adding, or changing entries. The OpenFlow system shapes and refines the rules which give the controller the ability to influence switches, which depends upon the maintenance and monitoring of robust data channels to achieve this. Three key types of interaction are observed within the OpenFlow network structure. The first sees information passed from the controller to the switch. This type of interaction plays a vital role in resource identification, channel modifications, switch configurations, and the extraction of data. The second is symmetric messaging, which does not rely on approval from the controller [41]. In fact, this form of connection needs no contribution from either the switch or the controller; whichever one triggers the interaction, it does not have to involve the other. For instance, 'echo' interactions are a good example. They are a clever way of testing whether the controller is activated and ready to receive packets. The third and last type of interaction is a

form of balanced and asynchronous connection. It is triggered by OpenFlow compatible switches and it does not require any contact with or approval from the controller. The aim is to warn the controller about the imminent receipt of packets, as well as to help it prepare for changing conditions, and switch faults.

2.7. Challenges of OpenFlow-based Networks

Despite their many benefits, OpenFlow applications do present some unique obstacles, so it is important for these issues to be discussed and explored. While some of the associated limitations cannot be investigated as part of this study (due to time and resource constraints), this section outlines some of issues relating to general performance, reliability, the scalability and security.

2.7.1. Performance

The reality is that most network creators and managers have embraced SDN as a superior replacement for older, more rigid protocol arrangements. It is not difficult to see why this would be the case. It provides adaptable, dynamic, efficient, and stable networking processes, alongside a whole host of other benefits. They include ease of use, smaller set up and maintenance expenses, an enhanced user experience, and fully centralised controls. Yet, it is just as important to point out that SDN structures also carry some fundamental performance limitations. For example, there are negative effects on performance, if all elements of the SDN are not meticulously configured and maintained, which potentially include lost packets, traffic congestion, inactivity or a lower throughout. Similarly, the control level can end up creating a kind of bottleneck within the network and severely disrupting traffic flows [52], which is a particular risk for big enterprises relying on just one controller.

In recent years, a number of studies have explored some of these risks and limitations in more depth. For instance, the work of Luo [53] and Tanyingyong [54] proposes new methods of improving the efficiency of OpenFlow systems. In fact, they recommend an updated network structure as a way to bolster lookup abilities. When network accelerator tools were applied, a 20% drop in inactivity was observed, in comparison with standard configurations. According to Luo, incorporating Linux and opting for a basic commodity network interface card is the most viable option. This strategy was also tested and the results

indicate a rise in throughput of as much as 25% on the productivity levels normally associated with OpenFlow switches.

According to Yeganeh and his research group [55], the best solution is Kandoo. The approach appears to be a highly effective way of improving general scalability and helping a network to handle greater amounts of traffic. Essentially, it is a network resource designed to decrease the number of events processed at the control plane and it works by applying two levels of controllers. The bottom level is made up of multiple controllers with no awareness of the state of the network and no links to one another. It deals with the majority of events so that the pressure on the top level is substantially alleviated. This top level is responsible for sustaining network wide conditions. However, other researchers recommend alternative methods to maximise interactions between switches and the CPU.

Mogul and his group [56] recommended the application of software oriented counters, but the problem with this study is that the researchers never actually tested the theory in experimental conditions. So, it is not known whether the recommendation is a viable one. What they did do is discuss the strengths and limitations of software based counters at length. However, without clear test results, the proposal can be nothing more than an unsupported suggestion. Hopefully, future studies will remedy this. Nevertheless, it should be borne in mind that OpenFlow switches gather statistical information relating to every new flow and according to the researchers, this information is securely held inside the switch via the use of Application Specific Integrated Circuits. They argue that holding the information in a CPU is a better idea, because it has a greater capacity and is more readily able to interpret diverse, dynamic data.

According to Lu [57], one possible answer might be to integrate CPU and ASIC processes. The researchers have explored weaknesses of using both approaches in isolation in detail. For one thing, the size packet buffer and size forwarding tables are significantly restricted and they believe that a combination of the two is a good way to minimise these negative impacts. They actually did go on to formulate a prototype design, testing their system by directing 50k bidirectional TCP flows between a total of four servers. It achieved 3.9 Gb/s of forwarding with the added support of a CPU. They also discovered that sizeable TCP traffic movements can be processed without any lost packets.

The published work of Vanbever and his group [58] recommended HotSwap. This is a resource which facilitates fast, accurate, and reliable modifications of SDN controllers. The

primary objective of the tool is to jump quickly between controllers without causing problems or faults across the network. Rather cleverly, it takes note of valuable data being passed among the switches and the controller. It is able to bootstrap the updated controller by mimicking earlier network activities. Once the updated controller finally gets up and running, the status of the network is exactly as it was before the changes were made. HotSwap is particularly useful at times when specific controllers are due a targeted update. According to the researchers, a solution of this kind is very important, because moving quickly between old controllers and updated ones almost always leads to traffic congestion and lost packets as well as there being a greater risk of faults.

As part of the recommended strategy aimed at solving key performance challenges in this thesis, a soft computing tool is incorporated. The resource turned out to be extremely good at providing valuable data and this information was used to streamline performance, alleviate congestion along with reducing the amount of lost packets. It was added to the OpenFlow controller and eventually transitioned to become an integral element of the system.

2.7.2. Reliability

When it comes to reliance on the controller, there are additional problems relating to accessibility [59]. To ensure the maximum degree of network access and avoid manual faults, the SDN controller must be able to modify and approve certain network topologies skilfully. Yet, it should be noted that, often, the amount of skill demonstrated is limited, due to conflicts caused by the 'brain/split' dilemma [39],[60]. This places too much weight and pressure on just one individual region, thus leading to potential failure. One possible solution is an OpenFlow compatible switch with the ability to forward packets via cached commands and protocols. However, even this does not provide a permanent fix, because interaction with an active controller is essential at some point. The 'legacy' network is defined by the fact that, if a single one or multiple components malfunction, network traffic gets redirected towards substitute components that are in close enough proximity to sustain healthy movement. The success of OpenFlow systems depends upon interaction with the controller, which is a necessity. Yet, the problem is that, when the SDN malfunctions and there is no substitute controller, this leaves just a single centralised controller to carry out the entire sequence. The destruction of the entire network is guaranteed if this lone controller fails, because it is the only place left for the system to try and tolerate a serious fault [59]. Hence, as there are no other controllers to share the load, the outcome is poor. Fortunately, IT

providers can decrease the likelihood of a total meltdown by diversifying and expanding the central controllers.

The SDN controller must be able to facilitate dual path commands or support the redirection of fast moving traffic into other, substitute links, if it wants to increase its tolerance of link and path faults. When it comes to controller faults specifically, it must allow the grouping of two or more SDN controllers when they are activated, but not currently in use [59]. For this to work though, memory and storage links between active and operational controllers must be established and kept open.

According to some researchers [17], there is an effective way to minimise concerns about network dependability. Assimilation between control congestion and unstructured multi-pathing has its roots in an innovative, highly adaptable load balancing multi-pathing strategy. It presents a distributed algorithm as a way to tolerate possible controller faults. The switches are consistently 'informed' by the algorithm, which alerts them to any alterations to the 'path load' within the relevant channels. On the other hand, according to [61], centralised controller processes are likely to disrupt network traffic and flow commands, if a fault develops with the controller. As one possible answer to the problem, they recommend the use of a distributed network structure (or SiBF) made up of a legion of rack managers (RMs). There is one manager assigned to every rack and they have the ability to function as controllers, in the event that one or more collapses. That is, if a central controller malfunctions, flow commands are processed by the temporary substitute RM, which remains the situation until the central controller can be brought back online and made fully functional again. If a switch malfunctions, SiBF introduces novel mappings (new backup flow commands) to the ToR switches for each activated entry. The only difference is that the packets get to their intended destinations by taking different routes, with the important thing being that they are successfully delivered as per the original plan.

It should be explained that managing the degree of latency requited for new flow generation is a tricky task too. If a switch processes a packet with no identifiable match, the initial two hundred bytes of data are delivered to the controller, which gives it a chance to implement a fresh or modified forwarding command. The problem is that the amount of inactivity (while the new rule is being formulated) must strike a perfect balance between efficiency and maximum functionality. For instance, if it takes too long for the new rule to be formulated, there is a higher chance that the accessibility criteria of a network will not be fulfilled [41] (refer to [59], for a more detailed example of this). In the case of OpenFlow

systems, link faults are quickly brought to the attention of the controller, so that it can start to create a solution and alternative route. This happens fairly quickly, but it does not happen fast enough. That is, there is still an unwanted degree of latency and it can take some time for the controller to be brought up to speed on the situation.

Within a distributed network, a multipath proposal for OpenFlow is designed to work out how to bounce back from faults with the minimum amount of latency. If the switch believes that a particular port has been deactivated, it applies a restorative backup flow. It might not be a completely 'real time' response, but it means that the controller does not have to give explicit approval, instructions, or interactions before a solution can be implemented. Without the need to travel to the controller and ask for guidance, the recovery can begin right away. Many high end carriers demand that their networks have the ability to recover from faults within a fifty second window. Yet, according to the experiment in [62], a high proportion of networks fail to achieve this target. It should be pointed out that, despite its limitations, centralised control systems do present some big benefits, even in terms of network protection and resilience.

2.7.3. Scalability

It should be noted that separating the control and data planes is not without its problems either. Quite apart from the general difficulty of defining standard APIs between both planes, there are likely to be limitations regarding adaptability. As Voellmy [63] explains, 'the SDN controller is at risk of becoming a tight, congested bottleneck if the network sharply increases the amount of end hosts and switches.' Nevertheless, the SDN can be separated and differentiated from older, more conventional networks by isolating the control and data planes from one another [64]. One feature of SDN is that both planes are able to develop and operate without the presence of the other. This is possible just so long as some form of API sustains a link between the two. Such a centralised perspective on the network serves to speed up modifications across the control plane.

When a great many packets are being delivered to one controller, the likelihood of serious traffic congestion rises. In fact, there are inevitable consequences for performance at any point that an overabundance of packets is being pushed through the channel. Strong, successful networks guarantee efficient congestion regulation at the switches and they don't require explicit commands or guidance from the controller to respond to faults [41]. Ultimately, no matter what arrangement the network takes, it is necessary to consider the

possibility of congestion whenever the amount of nodes is increased. The number of commands lined up to be processed by the controller steadily grows as the amount of flows, switches, and bandwidth increases. There is always a possibility that the components will become overloaded, so careful monitoring is essential.

The reality is that it continues to be very difficult for OpenFlow networks to support the movement of abundant flows. It can take an unsuitable long time to modify packets at the control plane too and this is another problem. In light of these weaknesses, it can be argued that there is uncertainty surrounding the question of whether OpenFlow applications can be used to regulate core network processes [41]. This is one of the reasons why OpenFlow is normally utilised for the outer portions of a network and not the middle functions. Research by Heller and his group [65] explained how just one controller can be used to maintain an optimal degree of inactivity. The researchers suggest that the implementation of *k* controllers will decrease inactivity by *k*. In regards to OpenFlow structures, there are two key scalability obstacles. The first is the fact that there are hardware related restrictions on the rate at which new flows can be formulated. The second is that, due to restrictions on hardware capabilities and table size, the amount of flows that can be accommodated is relatively low.

On top of the possibility of controller overload, the flow set up process often introduces new restrictions to network adaptability and flexibility [63]. This process is defined by the four steps outlined below:

1. The switch receives a packet, but it can find no matching table entry;
2. The switch delivers an alert to the controller and asks for further information on how to handle the packet;
3. The controller generates a brand new command and delivers it to the switch;
4. The switch uses the updated information to change the entry and update the command.

Preliminary tests on SDN switches and controllers proved that the controller is able to react to the alert within a one millisecond window [18]. Even with the support of high end components, the tests found that switches were able to 'accommodate a few thousand installations per second with a sub-10 millisecond delay at best.' So, while the reaction times of the controller are impressive, they hold little value, if the hardware switches cannot match or surpass this level of speed. Ultimately, the overall efficacy and productivity of the set up process is determined by the quality of its controller software and components (memory,

CPU, etc.) The forwarding information base (FIB) for a switch must introduce a degree of inactivity while it sends an alert to the controllers and asks for more detailed instructions.

In some cases, SDN unintentionally conceals a proportion of network traffic and this can make dealing with faults very difficult. When utilising older, more conventional systems, it would not take a network manager long to figure out, for instance, that the cause of congestion was most likely a faulty backup. The fastest response would be to reorient the backup to a period that was less hectic and overcrowded. However, when it comes to SDN, the only fully transparent elements related to User Datagram Protocol traffic are the tunnel endpoint and tunnel source, which means that the network manager cannot get a clear picture of how the tunnel is being utilised. That is, it is very tricky to figure out whether congestion is a result of an email, replication, or other malfunction. Moreover, the UDP tunnels hide the identity of the biggest communicator.

When congestion is causing havoc and users are experiencing serious issues, narrowing the fault down to just one area is unreasonably hard. As the degree of transparency decreases, it gets even harder to find viable solutions. Additional faults develop and the flexibility of the network is compromised [18], [64]. The reality is that, the longer it takes to find a workable solution, the more likely it is that the network will become untenable and its associated enterprise will lose money. It is clear then that inactivity and congestion both present serious problems for network flexibility [59]. The expenses associated with broadcast overheads and the propagation of flow entries both impose restrictions on the inherent flexibility of SDN networks. Therefore, to regulate and control the rapid expansion of flow entries, the controller should be permitted to apply header modifications to the core components. The targeted switches are the egress and ingress.

Another way to enhance the flexibility of the network is to introduce the use of virtual storage migration and virtual machines in the spaces between key locations. This usually takes the form of IaaS software and middleware solutions, with their roots in OpenFlow and Crossroads. The latter is a type of network fabric closely connected with OpenFlow. This issue has been explored a little in earlier chapters [66],[67] and alternatively, DIFANE can be employed, as [68] suggests. The term refers to a distributed flow maintenance structure with the ability to range in size, depending on the needs of the user (amount of hosts, protocols, flows, etc.) In fact, there are various software oriented proposals that could be used to solve many of the bigger limitations to SDN flexibility. For instance, CORONET is a system which is able to respond extremely quickly to faults [69].

It lends itself well to vast, sprawling networks due to the VLAN components attached to its local switches. The advantage of CORONET is that it can bounce back from link and switch faults with a minimum of downtime. Plus, it is compatible with modifiable, adjustable networks. It employs substitute multipath routine strategies when needed and it can be combined with almost any type of network topology. If that were not enough, it is very skilled at using centralised controllers to direct and deliver packets, as required. CORONET is defined by a series of modules that are tasked with route mapping, traffic control, topology exploration, and finding the optimal (fastest) channel for packets. One of its strongest elements is the employment of VLANs, because this is an effective way to standardise packet movements and ensure that processes do not become unnecessarily overcomplicated. They are also great at regulating the amount of flow commands and facilitating the upkeep of a fully adaptable and flexible system.

To reference one more proposal, DEVOFLOW, refers to a type of miniature flow regulated by the data plane, at the same time as much bigger flows are being handled by the central controller. The idea is that, by dividing the two, the workload for the primary controller becomes much smaller and there is less strain on the core components of the network [70], [71]. The strategy mitigates the impact of poor controller transparency and ensures that flow scheduling overheads remain as modest as possible.

2.7.4. Security

According to statistical research conducted by IT experts, approximately 31% of corporate users are uncertain whether they think SDN is a safer or more risky network structure than some of the other mainstream options [63]. Around 12% of corporate users believe that SDN presents serious security problems.

The controller contains invaluable insights about the network and therefore, is a prime target for invaders. For this reason, appropriate safety strategies need to be implemented to guarantee the accessibility of the controller, whilst protecting it from any threats. If an invader manages to get into the controller, the likelihood of the rest of the system being infiltrated is very high and the network becomes useless. The channel running from the switches to the controller is particularly sensitive. To some extent, Transport Layer Security can be employed as a way to protect communication. The problem is that this feature is not currently a mandatory part of SDN networks. Additionally, most permit the use of plain text traffic between the switches and controller. As a result, whilst TLS is able to increase the

safety of the channel, its overall impact is determined by the shape of the system and how the manager chooses to operate it. That person is under no obligation to incorporate a feature of this kind. To reiterate, heavy reliance on controllers is one of the biggest limitations for OpenFlow networks. It presents a significant security concern, as information thieves are aware of how important it is for a business.

Like the packet channels, flow tables can be particularly vulnerable to invaders. It is interesting to note that there is scant literature on this subject and hence, there is no a clear picture of just how sensitive this component may be. Nevertheless, multiple controllers can be used to regulate one flow table and in this scenario, one is usually experimental (not essential), whilst the other is the primary component. Crucially, however, the experimental controller is still at risk from the same security concerns as the primary component. Therefore, the network manager must sustain the stability and reliability of the flow table to guarantee that the two cannot be easily distinguished [41]. This reduces the likelihood that a destructive (invading) command originating from one of the controllers will go on to manipulate flow entries relating to the other as well. Presently, flow visor is the most common way of managing controller security, but OpenFlow may be able to offer superior solutions, seeing as it is a much newer technology.

There are some interesting benefits to be enjoyed with the support of a centralised software oriented controller. When it comes to distributed networks, it is necessary to deal with security concerns and inherent risks by making modifications to individual components and protocols, in turn. Obviously, this is very time consuming and it makes it harder to build a cohesive, unified system. Hence, using a controller located outside of the data plane is a better way to streamline security processes.

According to most researchers, strengthening servers by hardening network components, rather than superficially securing them, is always the superior option. Some IT companies invest a lot of time and money in what they judge to be serious security risks and weaknesses. Yet, it should be remembered that, according to extant literature, most SDN dangers arise because the components have not been fully or skilfully assimilated with one other. In other words, the network does not fit together seamlessly, so it is hard for operators to move around it quickly. It must be understood that complexity is rarely a positive thing for controller processes. Not only does it increase the expense associated with maintenance and reduce ease of use, for it also makes the likelihood of an attack greater, because the manager does not have a full picture of the network processes.

As aforementioned, if invaders get into the central controller, they have the freedom to destroy almost every component in the system [72], which essentially means the end of the network. It is within the power of the network owner to apply an intelligent access control list. This is a detailed breakdown of which packets are authorised and which should be denied and eliminated as way to keep the network safe from malicious commands. To enhance SDN security significantly, the controller must be able to tolerate and facilitate the authorisation and authentication credentials established by the network operators. To be precise, it must have the ability to recognise which actions can and cannot be permitted. Furthermore, to maximise security, the network operator has to be able to employ the same principles used to regulate traffic for the purposes of monitoring and protecting SDN movements.

To conclude, the controller must have the ability to warn network managers of any imminent invasion or infiltration [60]. It should also have the resources to restrict interactions to emergency messages and alerts only during a security breach. The problem is that the existing technology offers no guidelines on how to build up a deeper knowledge of network topology or the impact of latency and lost packets. Similarly, SDN lacks loop detection capabilities and it cannot respond to faults when certain conditions are present [73]. Furthermore, SDN does not offer enough provision for successful horizontal interactions between network nodes and this places substantial limits on how the network components function. While it seems clear that SDN has a lot to offer businesses and large data centre networks, its weaknesses cannot be ignored. In particular, it is in need of stronger, more cohesive standardisation regulations.

2.8. **Using Soft Computing in SDN**

Over the last decade, there has been a growing interest in the advantages of SDN architectures, with a particular focus on OpenFlow structures. The core component responsible for driving this new technology forward is the OpenFlow agent. This is a software program that a switch activates to facilitate and sustain remote, configurable links to forwarding entries. Overall, it is a positive transition, because it significantly raises the possibility of dramatic extensibility changes, enhanced network processes, and more manageable expenses.

It should be pointed out that, when experimenting with different degrees of network capability and functionality, it is easy to underestimate the importance of compatible, collaborative interactions and responses from network components. The problem is that there

is no official or formalised method for determining the value of a component. Network operators can find out whether a component is active, but this does not tell them much about its role and connections to other parts of the system. Additionally, the fact that OpenFlow principles and protocols are so variable makes it hard to determine whether a component is responding in an inefficient manner [74].

From a broad perspective, SDN has the potential to eliminate some of the biggest obstacles to applying and maintaining innovative processes within networks. Presently, the primary thrust of SDN is OpenFlow. Within this structure, software operates according to a centrally positioned controller. In this regard, it is useful to think of the controller as the 'brain' of the network. It is involved in all of the decisions and it also has the ability to make choices through the switches. If this were not possible, the network would run at a much slower speed [33]. The controller regulates and monitors the actions of a series of switches, which are themselves tasked with regulating and maintaining an assortment of forwarding tables.

The dependability of the network is imperative. This is true no matter what system architecture is being used, because stability and consistency result in faster, more efficient processes. It is not always easy to achieve though, especially with the invention of ever more sophisticated network configurations and components [75]. As a system grows more complex, the chance of software malfunction and bugs substantially increases. The good news is that many researchers are now turning their attention to the construction of stronger, faster and superior debugging tools.

It is possible to optimise load balance, cognitive network processes, and network safety by incorporating a variety of artificial intelligence oriented strategies [76], There are a number of studies that have sought to combine SDN networks with artificial intelligence features. While the results have mostly been inconsistent, there is evidence to suggest that hybridised AI strategies are the secret to facilitating more sophisticated activities within SDN architectures.

Soft-Computing (SC) can be used for developing intelligent system and learning machines. The main goal of SC is modelled and provided intelligent solutions for complicated problems that cannot be modelled mathematically. Therefore, SC will fulfil a strong, manageable and not expensive solution from approximate reasoning and uncertainty. Recently, the SC techniques are being used successfully in several applications [77].

Several experts recommend soft computing strategies as a way to further test and streamline networks. For instance, the work by Yilan [78] proposed a genetic algorithm designed to tackle bandwidth related multi path optimisation issues. Similarly, Gao and his group [79] argued that a Particle Swarm algorithm is the best way to deal with control placement challenges, because it accounts for both the constraints on controller abilities and the impact of forced inactivity.

Risidianto and his research group [80] spent time analysing the efficiency of an SDN virtual system, in relation to its responses to various virtualisation conditions. The researchers performed tests on par virtualisation, OS based virtualisation, and hardware facilitated virtualisation. Victor-Valeriu and Ionita [81] suggest a possible strategy for circumventing DDos invasions. They employed a sophisticated cyber protection structure with the ability to analyse dangers at a rapid speed. The resource was shaped around the strengths of a neural network and took inspiration from concepts of biological danger. Interestingly, the test managed to demonstrate full packet capture and minimise the impact of malicious invasions. However, the researchers point out that, for this to work, the central controller must first be alerted to the threat and recognise that it needs eliminating or avoiding. One of the biggest advantages of the proposed system is the fact that it can be used to gather identifying data about the invader. The work of Zhang and Fumin [82] investigate the efficacy of SDN applications when network management protocols are used for the purposes of automation. They also tested the capacity for IP load bearing, network virtualisation, the cohesive regulation of optical deliveries, Qos guarantees, and fast switching across wireless networks. Sgambelluri [83] focussed on particular common proposals designed to enhance backup and operational flows, particularly when a variety of urgencies are attached to commands and packets.

Ognjen and his group [84] implemented and evaluated an ant colony optimisation (ACO) approach to flow table rules in SDN environment. This design improves the quality of experience (QoE) by reducing the number of packets loss. The ACO-based heuristic algorithm includes calculation of new route path depending on traffic load and the network restrictions.

Yao and his team [85] proposed a new method to enhance the global feasible traffic in SDN network, this method called PSO based Maximum Flow (PSO-MF) algorithm. The goal of this algorithm was to maximise the number of forwarding rules by using PSO

technique to solve the size limitation problem with resources in TCMA-based SDN switches. This problem will be affected the performance of the network during high traffic demand.

S. Kim, in his PhD research [86] proposed cognitive network management design called CogMan for SDN. The proposed method offers reactive, loops deliberative and, reflective where the reflective loop is planned to use learning algorithms for thinking, and impact planning and decisions performed in the deliberative loop.

Mehdi and Liyuan in [87] demonstrated a method working based on repeated the same information every day in the data centres. The method is included implementation of an ANN network in the switch or router to choose a multipath route rather than single path route. Multipath routing with ANN can reach dynamic load balancing and optimized routing without involvement from the SDN controller.

Finally, the work of Wu and Huang [88] recommends the introduction of an innovative open flow controller. It is shaped to fit sophisticated intracerebral neural architectures. According to the researchers, it is an effective way to facilitate the robust but flexible transmission of media across a wireless network. By employing a modified PSO, the researchers were able to 'teach' the core controller equation to become more efficient, particularly when transmitting media distributed within the sigmoid and radial activation functions of the intracerebral neural network. Thus, whilst binary responses are determined by the controller, it is subject to the influence of link control protocols. These guidelines and principles are produced as well as being organised by a highly efficient neural architecture, which is itself sensitive to the impact of wireless link performance and mobile connection speeds.

The primary aim of this section was to discuss some of the most interesting solutions for overcoming key SDN limitations. Most come with some weaknesses of their own, but they still indicate a very promising future for the development of SDN networks.

2.9. **Summary**

Currently, the Internet of Things lies right at the heart of the biggest, broadest, most versatile networking solutions, with at present, over fifteen billion devices being linked to the internet. The figure is predicted to reach more than fifty billion by the year 2020. The amount of information produced by these devices is unfathomably large and the problem is that a successful IoT must be able to process and interpret it. As the reliance on networking technology grows, so too do the obstacles relating to resource distribution, traffic flows, and

fundamental network strength and safety. According to some experts, centralised controllers and increased configurability is the right way to tackle many of these challenges. Regarding which, one of the most promising recommendations is Software Defined Networking. It is hugely beneficial, because it offers a configurable and centralised system of control, but it does not need to carry out extensive modifications on existing networks to achieve these goals.

Throughout this chapter, SDN has been explored in great depth and considered in relation to a broad variety of conditions and scenarios. To summarise, SDN is believed, by most experts, to be a viable method of streamlining network maintenance and regulation. It is also a great way to support future research endeavours and attempts to improve interactions between the data and control planes. It should be noted that, while the aim in this chapter has been to cover as many different aspects of SDN as possible, it is not the most comprehensive account available. There are plenty of other resources of this matter and many investigate different issues.

3. Chapter Three: Performance Prediction of a Software Defined Network

3.1. Introduction

With the aim of achieving enhanced data transmission in software defined networking (SDN) an Artificial Neural Network has been proposed, in which the global network approach to SDN was employed to predict the performance of the Software Defined Network according to effective traffic parameters. Quality of service (QoS) is recognised as a significant and vital consideration in networks. In SDN/OpenFlow networks, the capacity for intelligent recognition of flow requirements (e.g. bandwidth, delay, jitter, and throughput) is an essential specification for the network components, if QoS is to be guaranteed. The two categories of components in SDN are the switches and controllers, with the latter instructing the former regarding how the flows should be routed. An additional component, which will be incorporated into the SDN network, is a data analysis server or cluster designed to store application data as well as to analyse and anticipate flow behaviour.

Those QoS parameters used in this thesis are round-trip time, throughput and the flow table rules for each switch, which are also used in the learning system. A POX controller and OpenFlow switches, which characterise the behaviour of a Software Defined Network, have been modelled and simulated via Mininet and Matlab platforms. An ANN has the ability to provide an excellent input-output relationship for nonlinear and complex processes. The network has been implemented using different topologies of one or two layers in the hidden zone with different numbers of neurons. Generalisation of the prediction model has been tested with new data that are unseen in the training stage. The simulated results show reasonably good performance of the network.

3.2. A comprehensive Survey of Intelligent Control Systems

Human intelligence has been widely used as the blueprint for the development of intelligent systems and humans remain at present considerably more adept than the resulting devices over a range of attributes. These attributes tend to be those specifically needed for attaining control objectives, including adaptability, analytical thinking, associative memory, creativity, the ability to learn, the ability to discern and distinguish items as well as the ability to make observations, either independently or under guidance, when faced with challenging situations [89].

3.2.1. Parallel Processing

The making of a good decision involves the interplay and synthesis of numerous information and learning sources in a strategy known as parallel processing. For example, in the process of selecting an item of clothing for purchase, a shopper is involved in considering numerous factors (colour, style, cost, fabric) which are taken into account along with the individual's previous experience and hearsay, in order to reach a final decision. Similarly, in the management of a dynamic system, the control specialist must begin by deriving the system's behaviour from a range of influential criteria.

The required system behaviour, under any given circumstances, can then be obtained by the appropriate selection and handling of the criteria. Thus, human beings make decisions by the parallel processing of information, such that a wide body of knowledge and data must be taken into account rapidly and simultaneously. This type of strategy has therefore been modelled and applied to networking in order to obtain human-like behaviour [90]. Thus artificial intelligence, such as the Artificial Neural Network (ANN), involves data handling by the interchange of inputs and outputs among numerous units. The data acquired by the decision maker is frequently fragmentary and/or vague (so-called fuzzy data), hence a range of deduction and estimation strategies (fuzzy logic) must be employed to arrive at an appropriate decision.

3.2.2. Learning Procedure

The central aspect of natural intelligence is the learning process, by which human beings are able to become increasingly astute and adept. With respect to artificial intelligence, the learning procedure involves training an intelligence-based system to carry out a specified task. There are three main approaches, as outlined below.

(i) Supervised learning. This is an evaluation-oriented approach in which a set of instructive tasks is presented according to the formula 3.1:

$$\begin{bmatrix} u_1^1 & \cdots & u_i^1 \\ \vdots & \cdots & \vdots \\ u_1^n & \cdots & u_i^n \end{bmatrix} = \begin{bmatrix} y_1^1 & \cdots & y_k^1 \\ \vdots & \cdots & \vdots \\ y_1^n & \cdots & y_k^n \end{bmatrix} \tag{3-1}$$

where, n = example value, i = input (u) value and k = target(y) value.

As these instructive tasks are executed the model parameters are adapted in order to make the model outputs approach the target values. This approach can be employed in intelligence-based control and modelling.

(ii) Unsupervised learning. This approach employs a set of inputs without any specified target values and hence, the model parameters are adapted in response to the input values alone. This approach is primarily employed for clustering and pattern recognition processes. It is also applicable in system analysis and in image processing (feature extraction).

(iii) Associative reinforcement learning. This evaluation-oriented approach differs from supervised learning in that it generates a single score or grade reflecting the level of performance of the model over a given group of inputs, rather than giving an individual output value for every input. This approach is particularly suitable for control software.

3.2.3. Adaptation

The ability to alter the pattern of behaviour to suit the environment, known as adaptation, is possibly the most intriguing aspect of human-like performance. Likewise, the achievement of robust and competent functioning under uncertain conditions necessitates intelligent control or modelling to revise or adjust the parameters and in some cases, the configuration of the system, continuously. The adaptability and robustness of controllers might be assessed by, for instance, a set of challenges designed to monitor changes in the desired set of outputs, performance perturbations, or performance failures [91].

3.3. Artificial Intelligence Methodology

As outlined in previous sections, the primary aim in establishing an artificial intelligence methodology is to generate human-like functioning. The extensive body of literature relevant to this endeavour is so vast that the resulting survey for this could not be completely exhaustive. Instead, the aim here is to highlight the development of practical techniques, with a particular focus on the SDN control and optimisation approaches. Particular attention will be drawn to the ANN methods, and to optimisation techniques, including genetic algorithms (GAs) and particle swarm optimisation (PSO), which are employed in this thesis.

3.4. Artificial Neural Network

The basic requirement of an intelligent system is that it must be capable of performing a range of operations, including machine learning, parallel data processing, generalisation etc, which enable it to generate appropriate decisions under uncertain conditions [89].

Artificial Neural Networks (ANNs) have emerged as a primary aspect of artificial intelligence due to their particularly favourable possibilities. Furthermore, the rapid progress in very-large-scale integration (VLSI) and super-fast computers, along with expanding research into neural system algorithms and architectures, have led to the successful application of ANNs in a considerable variety of interesting utilities. This includes their uses in intelligent control systems, primarily for the modelling and prediction of incomplete nonlinear and dynamic systems and for adaptation and control of uncertain dynamic systems [92]. A large number of exhaustive reviews have been published regarding the use of ANNs for modelling and control applications [93] - [94]. The primary features of ANNs are outlined below.

3.4.1. The Neuron model

The biological neural network uses the neurone as its basic component, with an estimated 10^{11} making up the human brain. As indicated in Figure 3-1, the neurone consists of three parts: the cell body, the axon, and the dendrites. The dendrites consist of branched nerve fibre nets, which are active in passing signals to the cell body. This, in turn, effectively measures and processes the signals to generate an output, which can be transmitted via the single long axon fibre, to other neurones. The axon of one cell connects with others via the accessible dendrites, the point of contact being termed a synapse [95].

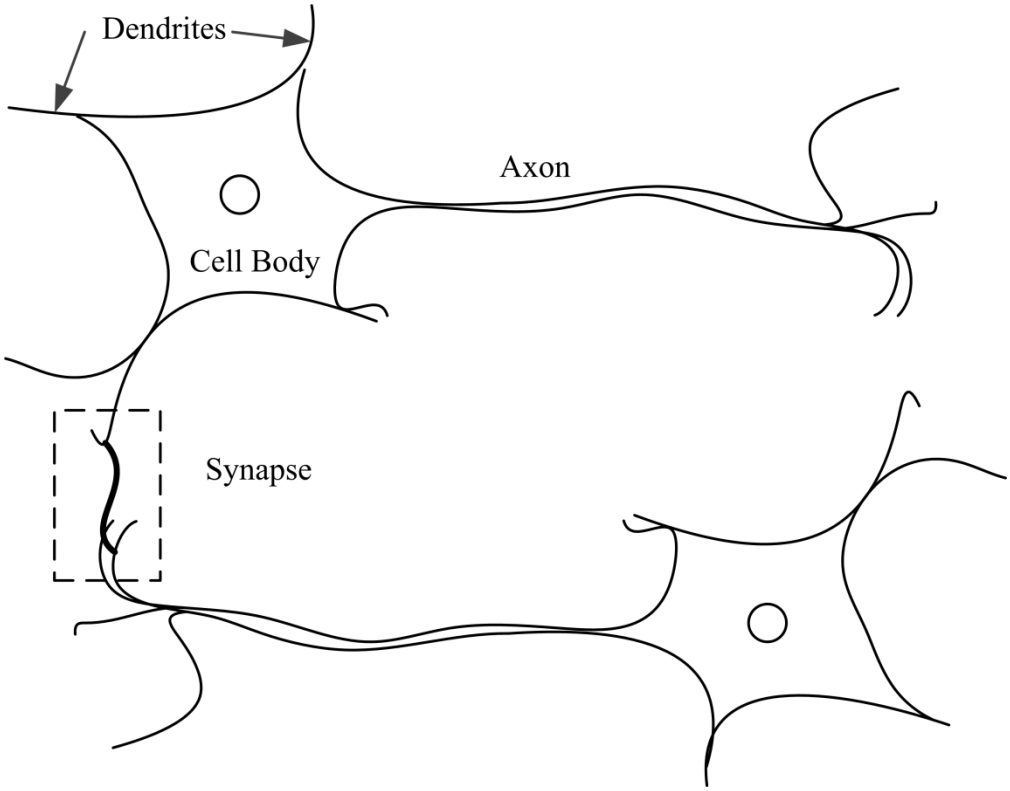

Figure 3-1 A streamlined schematic diagram showing the connection (synapse) between two biological neurones

Artificial neurones are of two fundamental categories: (i) single-input neurones, in which a scalar input is multiplied by a scalar factor (termed the weight) to generate a scalar output; and (ii) multi-input neurones, in which a vector of weighted inputs is received from the outside world or from other neurones and is summed according to equation Eq. (3-2)

$$\alpha_j = w_{1,j}u_1 + w_{2,j}u_2 + \cdots + w_{i,j}u_i + b_{1,j} = \sum_{i=1}^{n} w_{i,j}u_i + b_{1,j} \quad (3\text{-}2)$$

where, α = the activation state of neurone j. This formula shows that the relationship between the input (u) and activation state (α) is a linear combination. As indicated in Figure 3-2 and Eq. (3-3), the activation state is then processed, either nonlinearly or linearly, by an activation function.

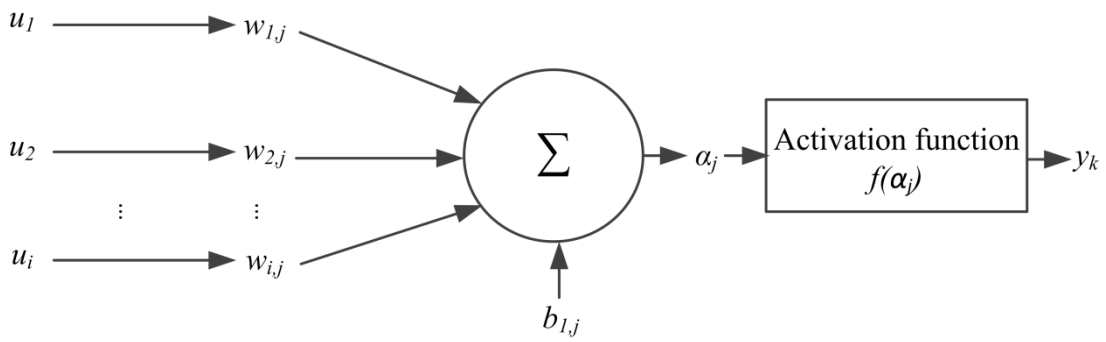

Figure 3-2 Signal processing in a multi-input neurone

$$y_k = f(\alpha_j) \tag{3-3}$$

As described in detail by Hagan [95], there are a large number of activation functions utilised in ANNs, each being selected according to the specific problem that the neurone is tackling.

3.4.2. The Structure of Neural Networks

Parallel operation requires the functioning in concert of a large number of neurones. This active set of neurones, termed a layer, is a standard building block of ANN structure or topology. The specific structural arrangement of the neurones making up the neural network is closely related to the learning algorithms employed in training the network. Three distinct primary categories of network structure can be identified, as outlined below.

I Single-Layer Feedforward Networks

Shown schematically in Figure 3-3, this type of network consists of a single layer of connection weights and can frequently be characterised as an input unit. This unit receives signals from the external environment and transmits them to other network components for processing. This initial layer is not counted in the layer number, as it is not involved in the performance of computations or in the function of the output unit which delivers the network response.

II Multi-layer Feedforward Networks

From Figure 3-4 it can be seen that this category is characterised by the presence of extra hidden layers containing hidden neurone computation nodes (also termed hidden units) which serve the purpose of providing a useful mediation between the external input and the network output. The inclusion of one or more hidden layers makes it possible for the network

to derive higher order statistics [96] and this important type of neural network has been fruitfully employed to elucidate a broad range of challenging issues in numerous scientific and technological areas [97].

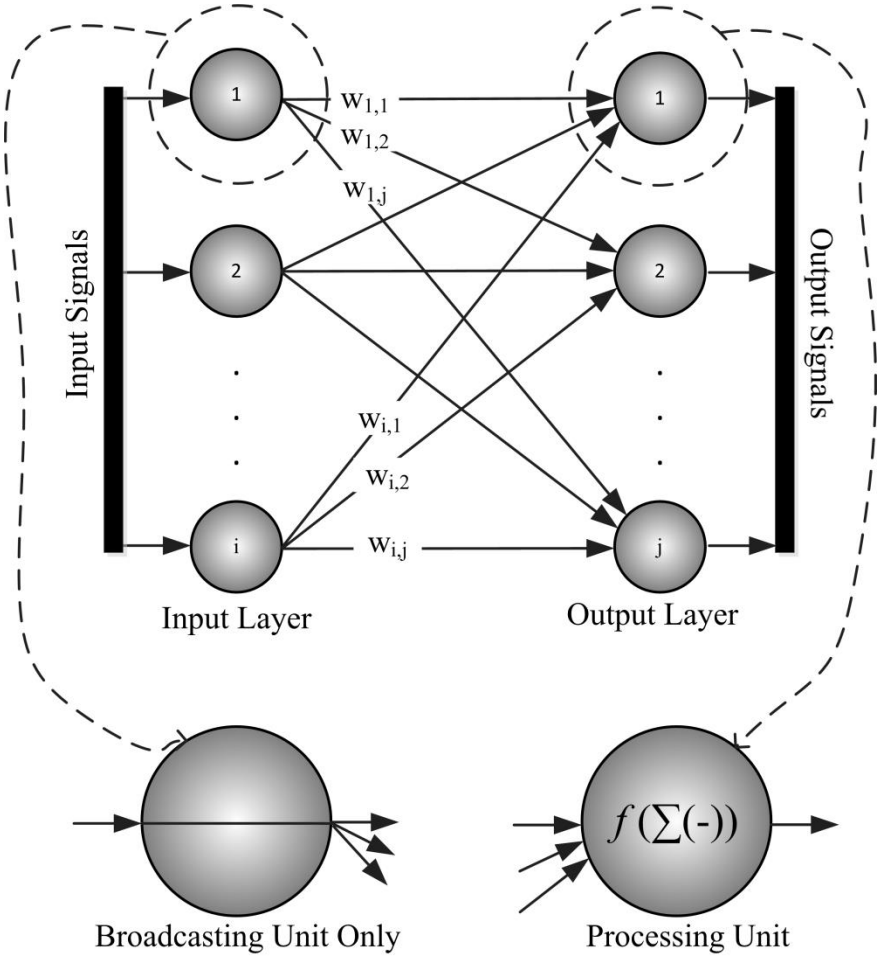

Figure 3-3 Schematic diagram of a single layer feedforward neural network

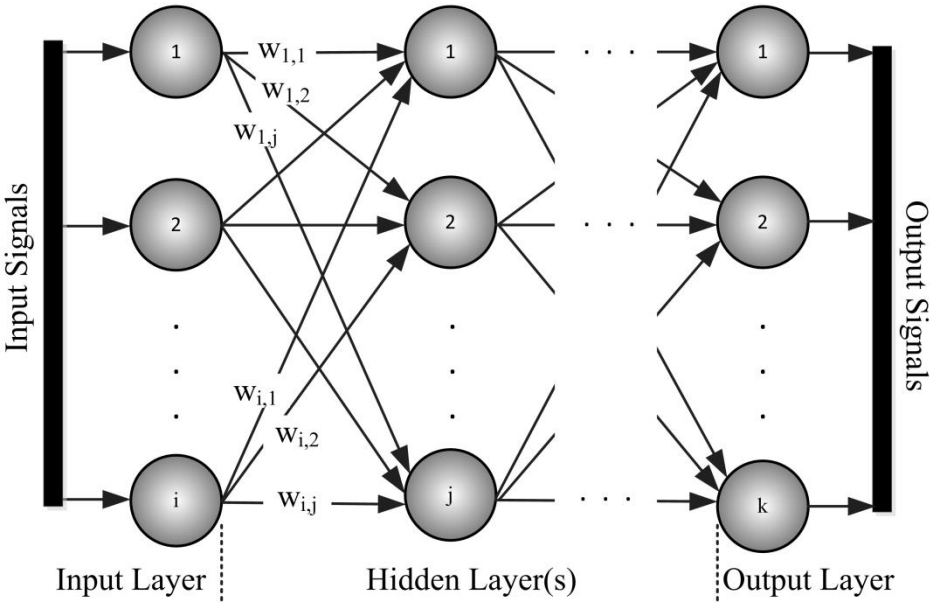

Figure 3-4 Schematic diagram of a multi-layer feedforward neural network

The majority of ANN-based control systems, such as those cited in the present work, make use of the multi-layer feedforward structure. Neural networks are able to estimate functions and mathematical operators to any desired standard. In particular, the feedforward artificial neural networks (FFANNs) can define the relationships between input and output for a range of systems, thus leading to their reputation as universal approximation devices [97].

This structure, consequently, became favoured for the characterisation of nonlinear models and classifiers; finding frequent application in pattern recognition, output prediction for various operations, nonlinear control, etc. The primary learning approach in multi-layer FFANNs is that of error backpropagation (Bp) [98]. This was developed around 1985 by a number of research groups and is generally used alongside an optimisation method, such as gradient descent.

The error Bp approach consists of the following steps:

(i) Random initialisation of the network weight followed by iteration of the subsequent steps until the desired criterion is attained;

(ii) Summation of the weighted input and computation of the hidden layer(s) output using an activation function Eq.(3-4):

$$h_j = f(\sum w_{i,j} u_i) \tag{3-4}$$

Where, h_j: the actual output of hidden neurone j, u_i is the input signal of input neurone i, $w_{i,j}$ is the weight between input neurone i and hidden neurone j and f is the activation function.

(iii) Summation of the weighted output of the hidden layer and computation of the layer output using an activation function Eq. (3-):

$$\hat{y}_k = f(\sum h_j w_{j,k}) \tag{3-5}$$

Where, $\hat{y}_k$ is the output of neuron k,

(iv) Calculation of the backpropagation error according to Eq.(3-6):

$$\delta_k = \acute{f}(\hat{y}_j)(y_k - \hat{y}_k) \tag{3-6}$$

Where, $\acute{f}$ = activation function derivative, and y_k = desired output (actual output) of the neuron k.

(v) Calculation of the weight correction term according to Eq.(3-7):

$$\Delta w_{j,k}(t) = \eta \delta_k h_j + \alpha \Delta w_{j,k}(t-1) \tag{3-7}$$

where, t is a time step, η is an appositive constant called the learning rate coefficient and α is another constant called a momentum factor;

(vi) Summation of the change in input (delta) for each hidden neurone and computation of the error term according to Eq.(3-8):

$$\delta_j = \sum_k \delta_k w_{j,k} f'(\sum_i u_i w_{i,j}) \tag{3-8}$$

(vii) Calculation of the weight correction term according to Eq.(3-9):

$$\Delta w_{i,j}(t) = \eta \delta_j u_i + \alpha \Delta w_{i,j}(t-1) \tag{3-9}$$

(ix) Revision of weights according to Eq.(3-10) and Eq.(3-11):

$$\Delta w_{j,k}(t+1) = \Delta w_{j,k}(t) + \Delta w_{j,k} \tag{3-10}$$

$$\Delta w_{i,j}(t+1) = \Delta w_{i,j}(t) + \Delta w_{i,j} \tag{3-11}$$

III Recurrent Neural Networks

In contrast to FFANNs, recurrent neural networks (RNNs) incorporate one or more feedback loops, i.e. they are structured with internal connections that feed back to another layer or to themselves. This structure makes it possible for the network to handle unidirectional data flow (as in biological systems) and to discriminate between separate input patterns from the same input sequence (thus identifying time-dependent patterns). A feedforward network is incapable of performing the latter function without pre-processing of the inputs [99].

Furthermore, RNNs possess a dynamic memory, such that their outputs at any specific time are shaped not only by the present input, but also the previous inputs and outputs. The primary categories of RNNs are the fully recurrent network, the Hopfield network, recursive neural networks and the Elman network.

3.5. Simulation Analysis

Due to the highly diversified nature of the network, the behaviour of software-defined networking systems is significantly non-linear. Consequently, the QoS of the network can be impacted upon by a significant number of factors and the operational correlation between the various performance parameters cannot be expressed by any existing near-standard statistical evaluation. However, an ANN can be used to predict the optimal performance of an SDN network based on the rule table inputs and QoS parameters as outputs. The ANN has the clear advantages of economy, simplicity, and efficiency and, most of all, the ability to learn from examples. The neural network has been modelled using the Levenberg – Marquardt training algorithms, with the absolute percentage error of predicted and computed QoS parameters being obtained by the steps outlined in this chapter.

To start with, the network simulation experiments are detailed using Software Defined Networks (SDN) with the Python-based POX Controller in a simulated environment with a

Mininet network emulation platform. Then, the Matlab programme is used to pre-process all the collected data sets and to prepare them for implementation in an ANN. The model is developed according to the following steps.

1. Simulate the SDN by Mininet. Three different SDN network topologies are used, Mesh topology with 5 switches, Mesh topology with 15 switches and custom topology with 9 switches. For each switch two hosts are connected.
2. Database collection. A wide range of data have been collected from SDN network included flow rules and network performance parameters. The network program is run for each topology for 180 minutes and collects all the data for every 5 minutes to build the learning intelligent system. As is well known, QoS has been a very important and indispensable issue in networks. In order to ensure it in SDN/OpenFlow networks, it is absolutely essential to improve the controller or the switch performance. Hence, the important requirements of flows, such as throughput and delay (RRT) are collected and used as data sets output in the ANN association with the flow rules that generate these parameters.
3. Analysing and pre-processing of the data. The input for the ANN will be a set of flow rule and the corresponding throughput and delay for that rule. In order to begin with ANN modelling, pre-processing (normalising) of data is vitally important to make all input parameters lie within a specific range [-1, 1].
4. Training of the ANN using all the data that are collected from three different scenarios, which are run for three hours. Train the network with specific properties like a network topology, training algorithm, and a minimum of accepted error between the predicted and actual output.
5. Testing of the trained ANN in order to check for model generality: the trained network is tested by unseen data and then the output compared with actual one.
6. Post-processing of the output data: after an acceptable result has been obtained by testing network, the normal value of output is calculated by post processing of the predicted value.
7. Using the ANN, once trained, for simulation and prediction of the performance of SDN.

3.5.1. Creating the training dataset from SDN Topology Selection

All experiments were conducted using a Dell PC, with the processor Intel Core (TM)i7-4500U CPU @ 2.4GHz, and with 8 GB RAM, which runs a 64-bit

Windows 7 operating system and a VMware Player virtual box. The guest OS installed in the VM was Linux OS Ubuntu 14.04 32bit with 1GB RAM. A Mininet 2.2.1 emulator was installed on this VM, with a POX 2.0 controller. The aforementioned system was utilised with a POX controller with three different topologies: mesh topology, which has five switches and two hosts in each switch; Mesh topology, with also fifteen switches and two hosts for each switch; and finally, custom topology, which has nine switches with two hosts for each switch. The capacity of all the links between switches was 10Mbps) and the link capacity between the hosts and corresponding switches was also10Mbps.

The experiment involved making use of the Python-based POX OpenFlow controller, which arose from the original NOX OpenFlow controller [100]. The POX controller was selected, because it is compatible with the spanning tree protocol (STP), which is essential when setting up a mesh topology with logical loop-free pathways. Once the network topology had been formulated, the POX controller needed to run on a terminal. On start-up of the POX controller, the OpenFlow switch was directly connected to the network in readiness for receiving OpenFlow messages. The module used in the present work is intended to run in conjunction with the POX forwarding L2_learning component.

In order to validate the proposed scheme, the network traffic was generated using the latest version of the network performance and measurement tool iperf (version 2.0.5). This open source tool is presently available on Linux, Unix and Windows platforms with identical command options. It is designed to evaluate the trace performance of the Transmission Control Protocol (TCP) and the User Datagram Protocol (UDP) with both unicast and multicast transmission [101] and was selected because TCP is used in the applied program. The iperf tool follows the client-server model and is compatible with both IPv4 and IPv6[102]. The iperf tool measures a range of statistical values including datagram loss, delay jitter, and throughput. Its output typically contains a time stamped report of the quantity of data transferred.

To make the simulation closer to the real environment and more reliable, a link capacity of between 10 Mbps-25 Mbps was deployed because L2 switches were used [103], [104]. This meant that each host in the network communicated with other hosts by sending a traffic rate of 10-25Mbps. On the client side, traffic was generated by means of TCP traffic, while on the receiver side, the incoming data size, throughput, delay, and the loss packets were recorded. Every 5 minutes all traffic parameters were measured and

recorded, with the scenario being run for a time period of 180 minutes. Traffic was generated from the hosts 80% of which represented clients, whilst the rest were servers.

A. Design

Due to the design of OpenFlow, flow entries were stored in the switch flow tables and then packets were forwarded to their destinations based on the instructions received from the OpenFlow controller. When the program was run, as explained above, 20% of the hosts were servers and the rest were clients. The server hosts received the data from all other clients. Starting with the first host, the remaining servers followed on from this constituting the designed 20% and then the rest were clients. The program was run for 180 minutes and by using ipef traffic generator, three lots of traffic are generated. The first traffic flow lasted three hours, the second was of two hours duration and the third lasted one hour. There was stress on the network for the first hour, medium traffic in the second and low traffic during the last hour.

B. Development

As previously noted, the controller makes forwarding decisions that are informed by the network state. Since an ANN has the capability of generating extremely high accuracy predictions, the controller can make forwarding decisions based upon precise bandwidth utilisation values. Messages regarding the current state of each switch are passed to the controller at intervals of time, thus providing current values of the network state, including detailed statistics relating to each port, flow and queue, along with the bandwidth utilisation values. The specific time interval at which messages are passed must be carefully chosen, such that it is not too long for the controller to be able to keep up-to-date with the network state and not so short as to generate excessive traffic in the controller. A time interval of five minutes was selected for the present research programme.

In order to create the presented design, the python programming languages was used as the POX OpenFlow controller is python-based. In this module, a network topology is developed to create traffic as well as to measure it and other network parameters.

1. Run POX controller.
2. Start the script for network simulation and performance measurement.

3. When the packets are transmitted over Ethernet networks, the packets will arrive at an ordinary Ethernet switch, which usually looks up the MAC address of the destination in its MAC table. In the MAC table, the information for mapping from the MAC address to the output switch port is stored and the switch will use this information to forward the frame to the specified switch port. For other subsequent frames, the communication rules from the source MAC address to the ingress port are learned and stored in the MAC table.
4. Determine the total run time, interval measurement time and choose the topology including the number of switches.
5. Collect the statistics data which include:
    - For the controller (number of packets sent to the controller, response time for the controller and average flow table creation, both in seconds);
    - For the switch (average delay in the switch);
    - Performance (RTT, throughput, total packets sent, dropped packets and bandwidth utilisation).
6. For each source and destination, all possible paths with information about the number of hops, delay, and bandwidth will be collected.
7. libpcap API is used in a C++ shared libraryto collect data. This list shows the type of protocol used to generate the traffic, source IP address, destination IP address and some extra information about the time interval.
8. Finally, for each path in the network, the flow rules that represent a path must be saved in the database. This data will be used in the learning system to build the cognitive system.
9. The program will generate three different high levels of traffic load to stress the network, and for monitoring all the scenarios of traffic load. Moreover, based on these different scenarios the cognitive agent has been built to study the different environments and to make correct decisions.

3.5.2. Artificial Neural Network Simulation

For this work, the dataset of the inputs and outputs was divided randomly into three subsets:

1) Training set (70%);

2) Valediction set (15%);

3) Testing set (15%).

The first subset was for establishing the gradient as well as updating the network weights and biases. The error in the second subset was observed during the training development.

The validation error is usually reduced in the initial training phase, as is the training set error. Nevertheless, when the network overfits the data, the error in the validation set invariably starts to rise. In the current case, the network parameters were saved at the minimum of the validation set error [17]. As mentioned earlier, the optimal topology giving the best performance was selected by conducting an exhaustive search for the number of neurons in both the first and second hidden layers. Figure 3-5 shows the nested loop architecture for choosing the optimal ANN topology. During each run, the number of neurons (i, j) was selected by conducting all those possible from 1 to 20 in hidden layer(s). Due to the random initialising of the parameters (weights and biases), the network training process (k) was repeated ten times with random initial weights for each single run. The ANN was trained in a nested loop and the recorded performance index, namely, the Mean Square Error (MSE), was saved in the external matrix containing all the parameters (i, j, k) gathered during the loop. The optimal structure of the network was chosen depending on the best performance (minimum error between the predicted and actual matrix).

The transfer function used in the hidden zone is log sigmoid, given by Eq. (3-12):

$$Yi = \frac{1}{1+e^{-x}} \tag{3-12}$$

where, Xi represents the input for the neuron in the hidden layer, whilst Yi is the neuron output. For calculating Xi , the input values need to be normalised in the range [-1, 1], which corresponds to the minimum and maximum actual values. Subsequently, testing the ANN requires a new independent set (test sets) in order to validate the generalisation capacity of the prediction model.

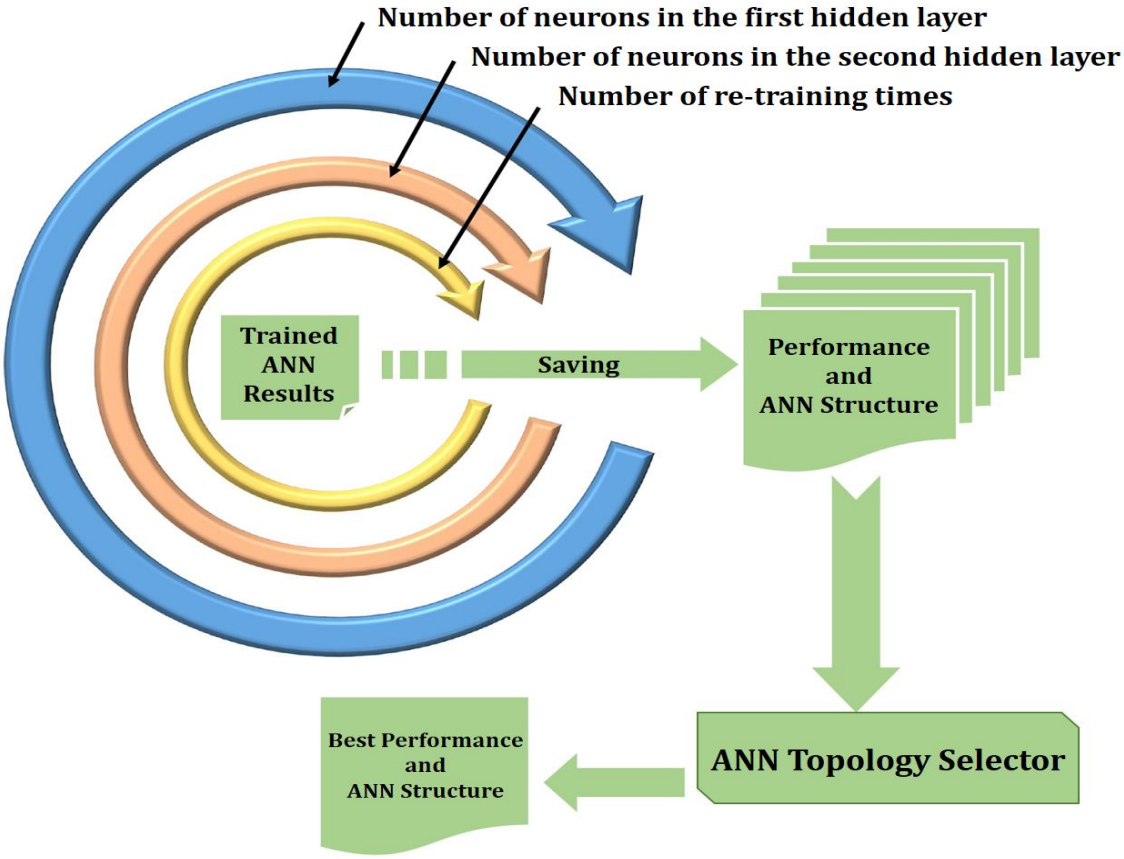

Figure 3-5 Exhaustive search of optimal NN topology

3.6. Simulation Results and Discussion

According to the selection strategy, three different scenarios were used to train the ANN network:

1- Mesh SDN topology using five switches.
2- Mesh SDN topology using fifteen switches.
3- Custom SDN topology using nine switches.

When considering which Mesh SDN topologies to implement, the five switches were selected to be the minimum, with the maximum set to be fifteen. The reason for that is the limiting effect of using Mininet emulation; which restricted the maximum number of Mesh SDN topology implemented in this system to fifteen switches. Each topology using a POX controller and each switch in every topology was connected to two hosts. Every time, the program was run for three hours to collect the data that were used in the ANN network learning system. The data that were collected include the flow rule tables, which were sent from the controller to each switch and these were used as inputs to the ANN network, plus the QoS parameters for the SDN network which represented the output for this network.

The structure of the ANN network includes:

1. Seven inputs parameters, which are the switch number, input port, Ethernet source, Ethernet destination, TCP source, TCP destination and action;
2. One hidden layer with a symmetric sigmoid transfer function;
3. One output layer including two parameters, throughput and round trip, as QoS performance.

The collected input-output dataset was then randomly split into subsets of 70%, 15% and 15% for respective use in the training, validation and testing. The performance of SDN was evaluated by implementation of a feedforward multilayer network. The Levenberg-Marquardt training algorithm (LMA) was run ten times for each network architecture using various random initial weights and biases. The exhaustive search method was used to evaluate the performance of various architectures under three scenarios. The optimum result was achieved for the best trained ANN with one hidden layer, in comparison to that with two.

The following are the results for the three scenarios trained in the program.

1- Mesh topology with five switches and ten hosts:

Regarding the selection strategy, Figure 3-6 shows the results of the network architecture. This consists of seven input neurons in the first layer, 18 in the hidden layers with a logarithmic sigmoid nonlinear activation function and two output neurons in the output layer. The training of the network shows the best performance of the ANN structure gives comparably better performance of MSE, with 2.5716×10^{-6}, as shown in Figure 3-7.

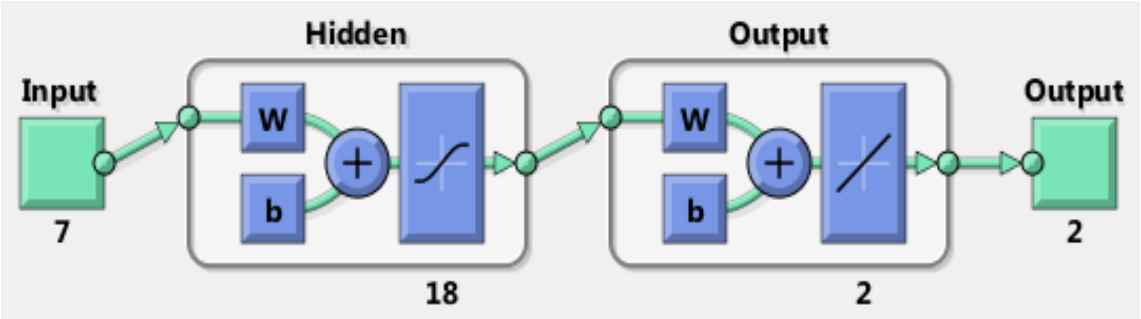

Figure 3-6 The ANN structure for an SDN topology with five switches

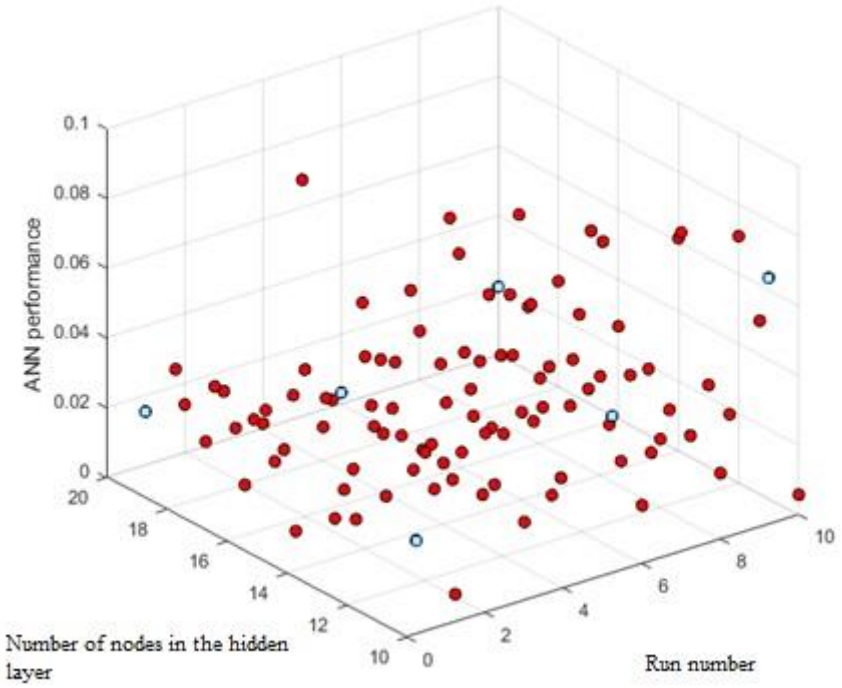

Figure 3-7 Performance of ANN for SDN topology with five switches

After examining the performance of different architectures according to selection strategy as shown in Figure 3-5, a network trained by Levenberg-Marquardt algorithm showed good performance indicated by a Mean Square Error of less than 10^{-6}. Figure 3-8, on the next page, illustrates the training session for an SDN five switches mesh topology, where the number of iterations (epochs) is determined when the training error decreases to the target figure. The validation set, demonstrated in the next figure, is used to minimise overfitting. Usually, the network is trained with the training set in order to adjust the weights. The validation set must also be inputted to the network to prevent its overfitting and to calibrate the models. While this would also confirm if the error is within a certain range, the set is not directly used in adjusting the weights, but to obtain the optimal number of hidden units. The model's accuracy on the test data indicates a realistic estimatation of the model performance on unused data to confirm the network's predictive power. If the accuracy over the training data set increases but the accuracy over then validation data set remains the same or decreases, then the overfitting of the neural network is occurring and the training must cease.

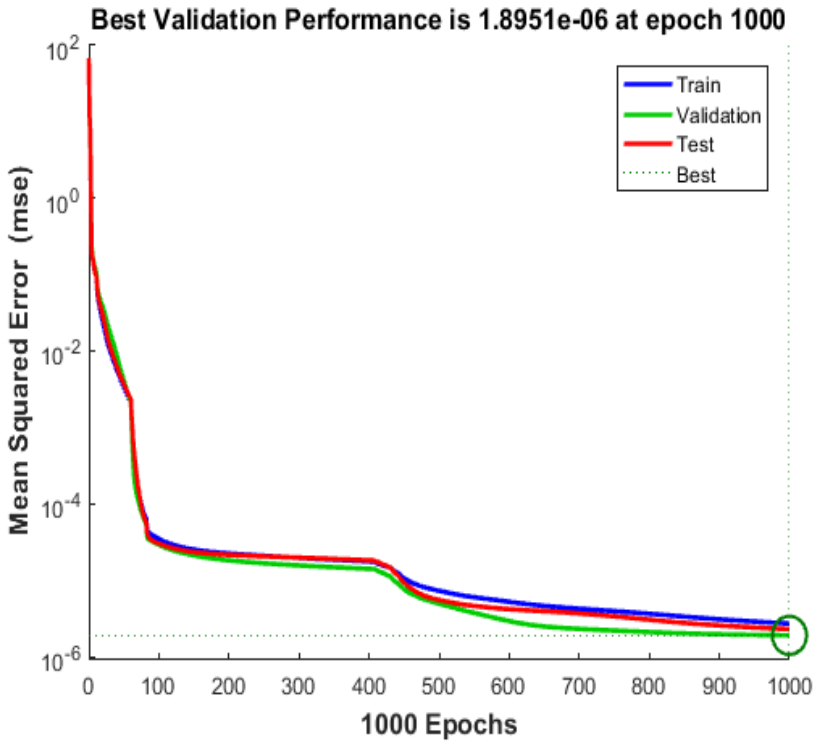

Figure 3-8 Training session for an SDN five switch topology

Whilst the effectiveness of the developed neural network model for the prediction of SDN performance must be estimated to some degree from the error in the training and/or testing data sets, it is frequently beneficial to examine the network behaviour more closely. For example, a regression analysis between the network response (predicted output) and the corresponding target output can provide more detail. As shown in Figure 3-9, the optimum linear regression relating the target to the network response in an SDN network is indicated by a straight line. A perfect fit, with the predicted outputs precisely matching the actual values, would give a straight line with a gradient of one and an intercept of zero. Figure 3-9 indicates that the values obtained were near optimal, i.e. the response to the training sets was good.

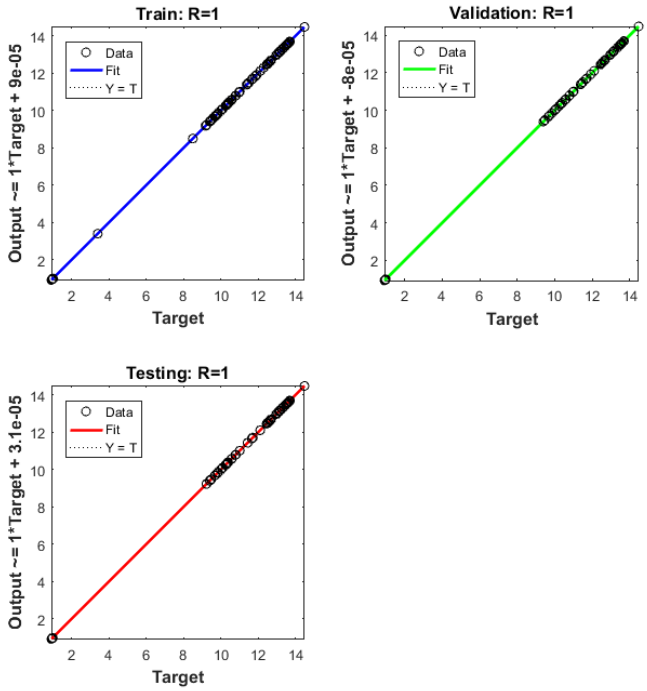

Figure 3-9 The linear regression of the targets relative to outputs for the SDN performance in training set.

Figure 3-10 and Figure 3-11 show the mean square errors between actual and predicted SDN output performance, which are 10^{-4} and 10^{-5} for each output, thus indicating a close relationship between these values.

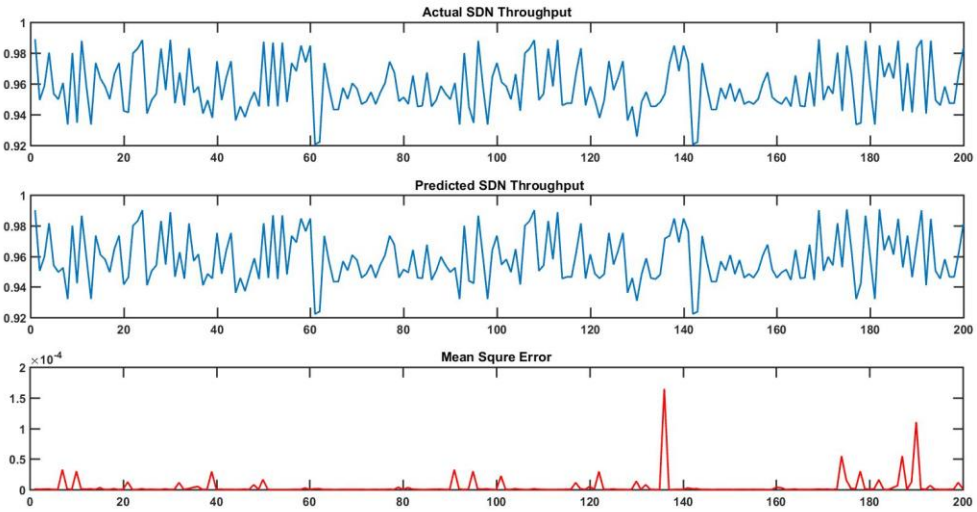

Figure 3-10 The actual and predicted Throughput performance

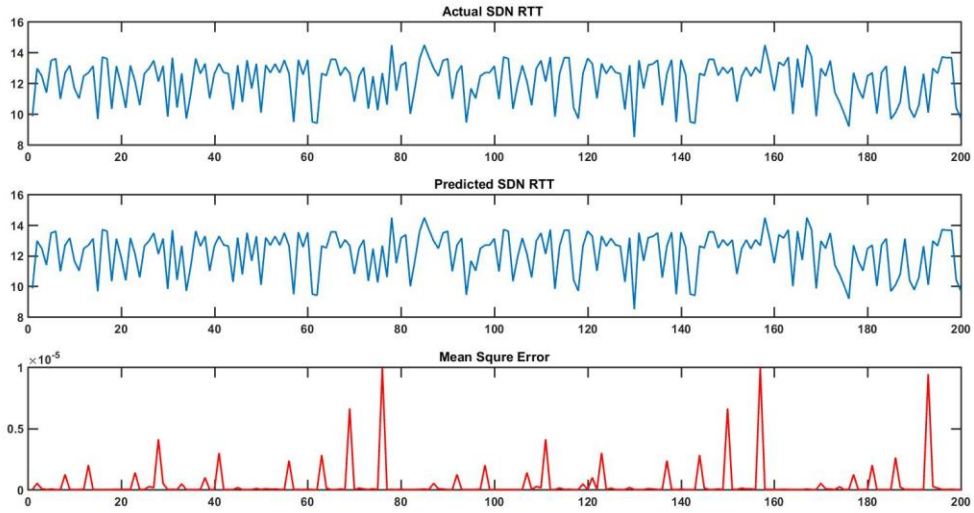

Figure 3-11 The actual and predicted RRT performance

2- Mesh topology with fifteen switches and thirty hosts:

The ANN structure for an SDN topology with 15 switches consisting of 19 neurons in the hidden layer during the second interaction, as shown in Figure 3-12, gives comparably better performance of MSE, with $9.629e^{-13}$, as shown in Figure 3-13 and Figure 3-14 which illustrate the training session.

The predicted and actual SDN performance for two outputs the throughput and RRT are compared and depicted in Figure 3-15 and Figure 3-16, while the linear regression of the date is shown in Figure 3-17.

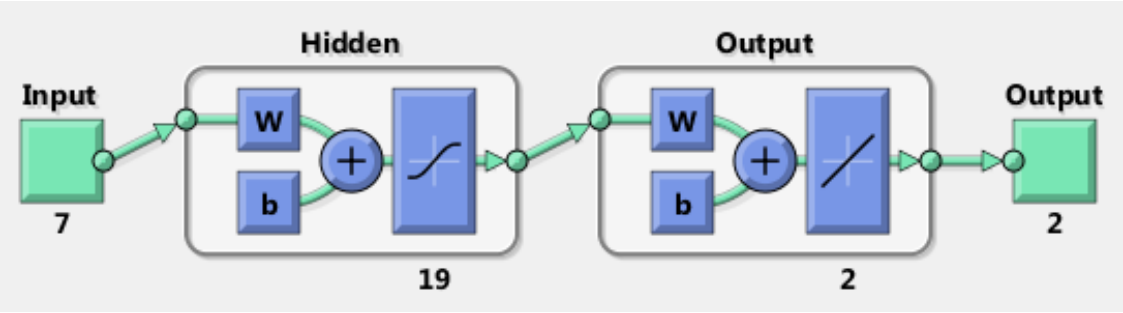

Figure 3-12 The ANN structure for an SDN topology with 15 switches

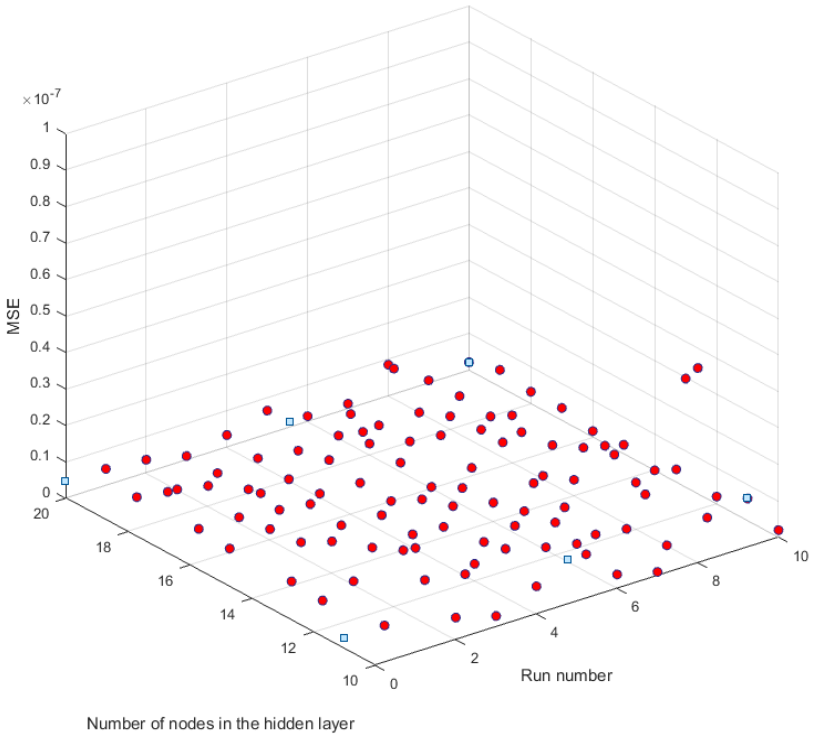

Figure 3-13 Performance of an ANN for an SDN topology with 15 switches

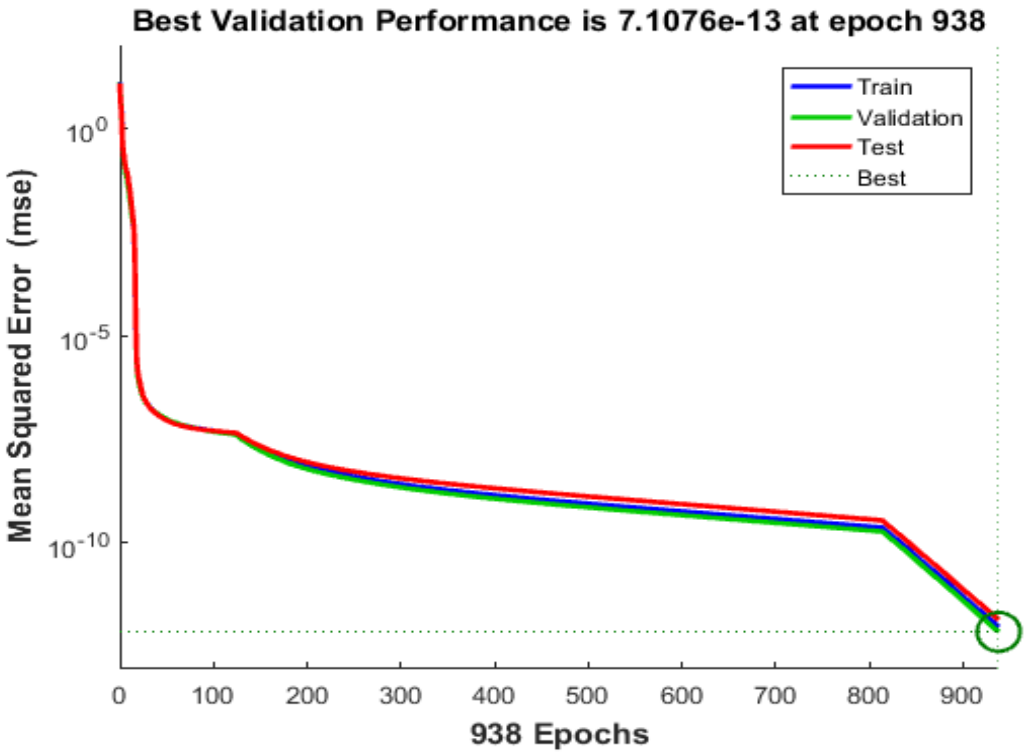

Figure 3-14 Training session for an SDN 15 switch topology

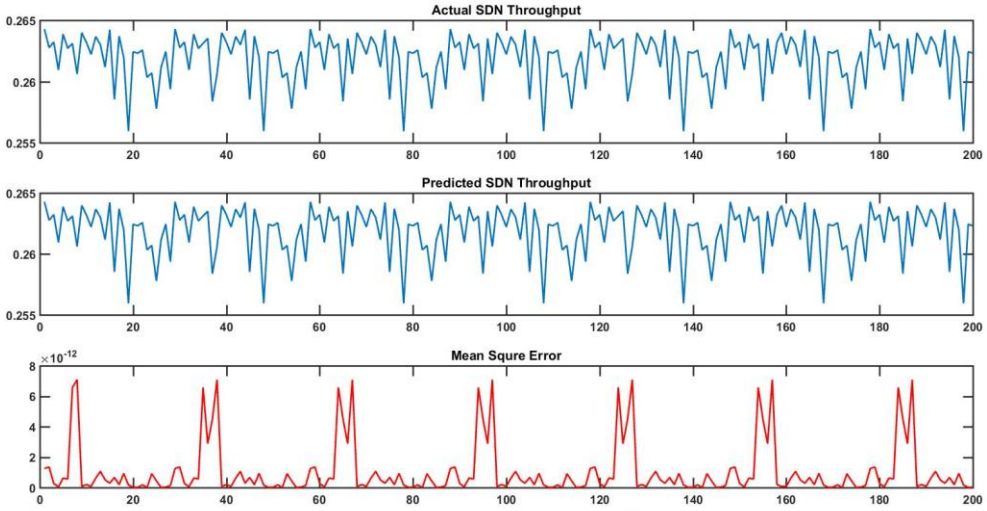

Figure 3-15 The actual and predicted Throughput performance

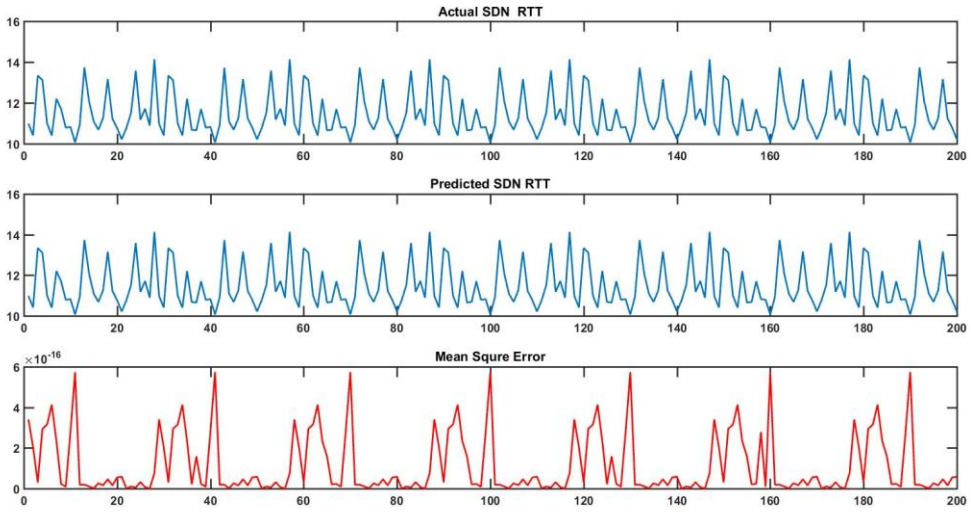

Figure 3-16 The actual and predicted RRT performance

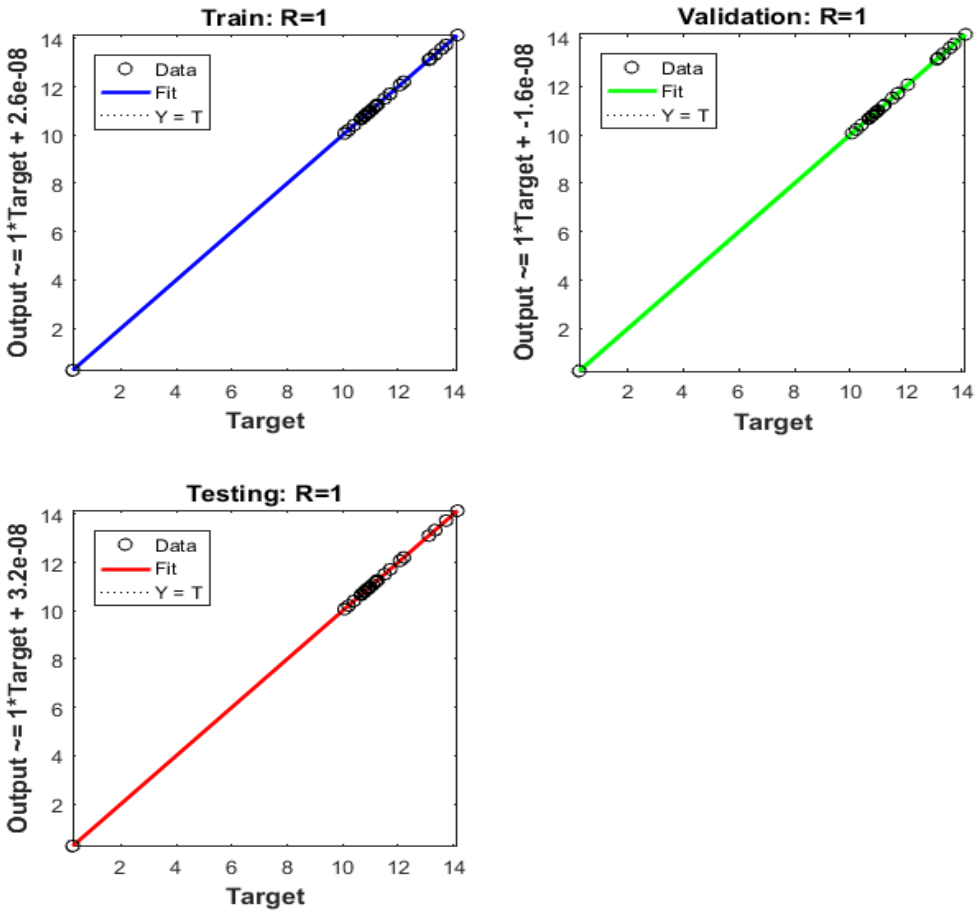

Figure 3-17 Linear regression of the targets relative to outputs for the SDN performance in the training set

3- Custom topology with nine switches and eighteen hosts:

In accordance with the selection strategy, Figure 3-18, show the results of the network architecture. This consists of seven input neurons in the first layer, 19 in the hidden layers, with a nonlinear activation function (logarithmical sigmoid) and two output neurons in the output layer. The training of the network shows the best performance of the ANN structure gives a comparably better performance with a MSE of 0.4660, as shown in Figure 3-19.

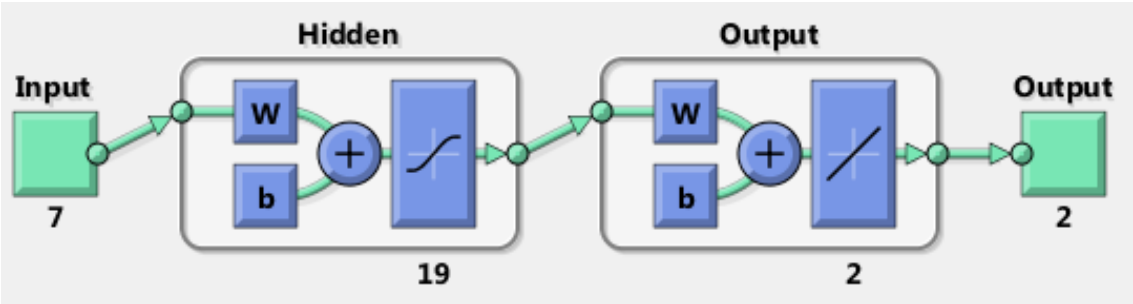

Figure 3-18 The ANN structure for an SDN topology with nine switches

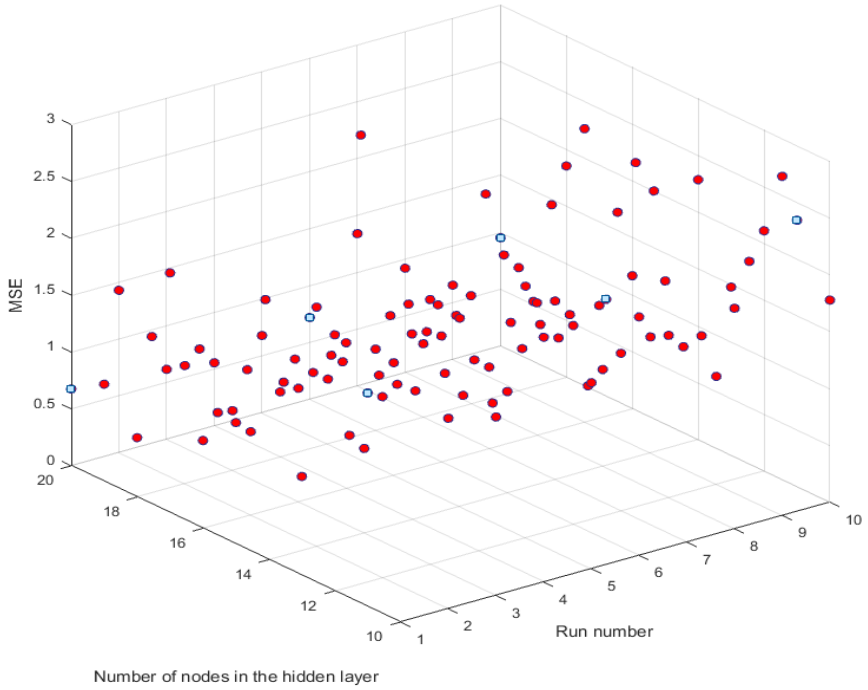

Figure 3-19 Performance of an ANN for an SDN Custom topology

Figure 3-20 illustrates the training session for SDN custom topology, number of iterations (epochs) is determined by when the training error decreases desired amount.

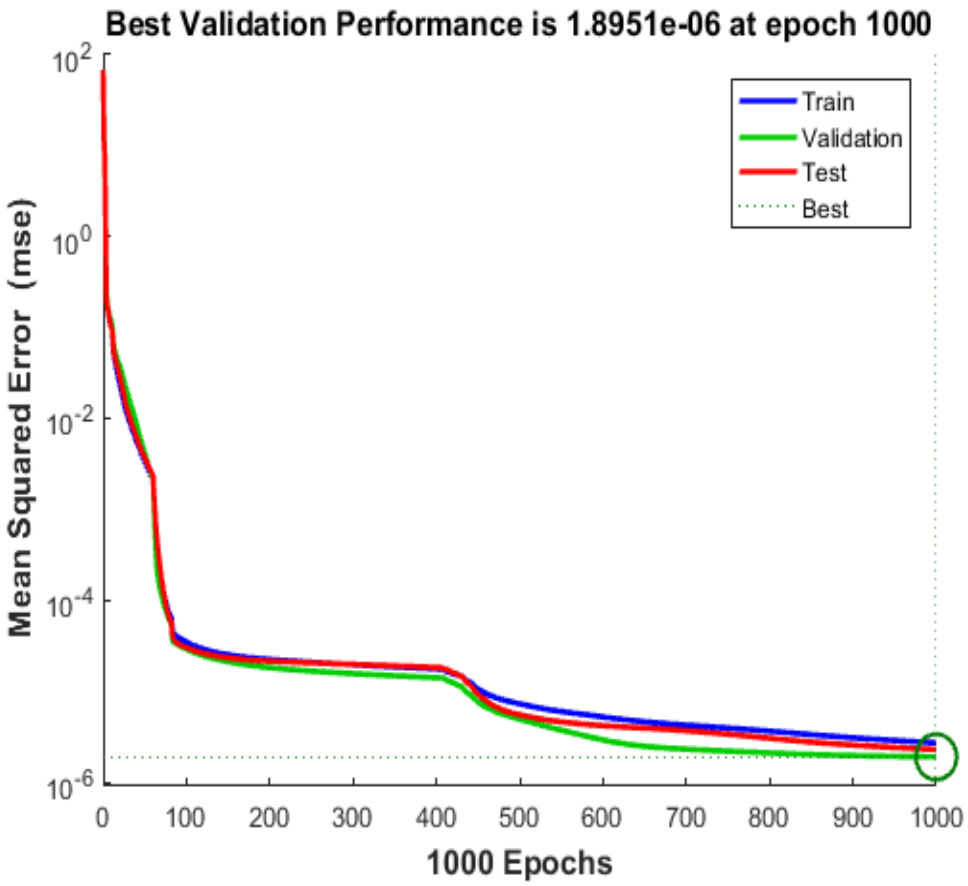

Figure 3-20 Training session for an SDN custom topology

Figure 3-21 illustrates a straight line representing the best linear regression relating the target to the network response in an SDN network.

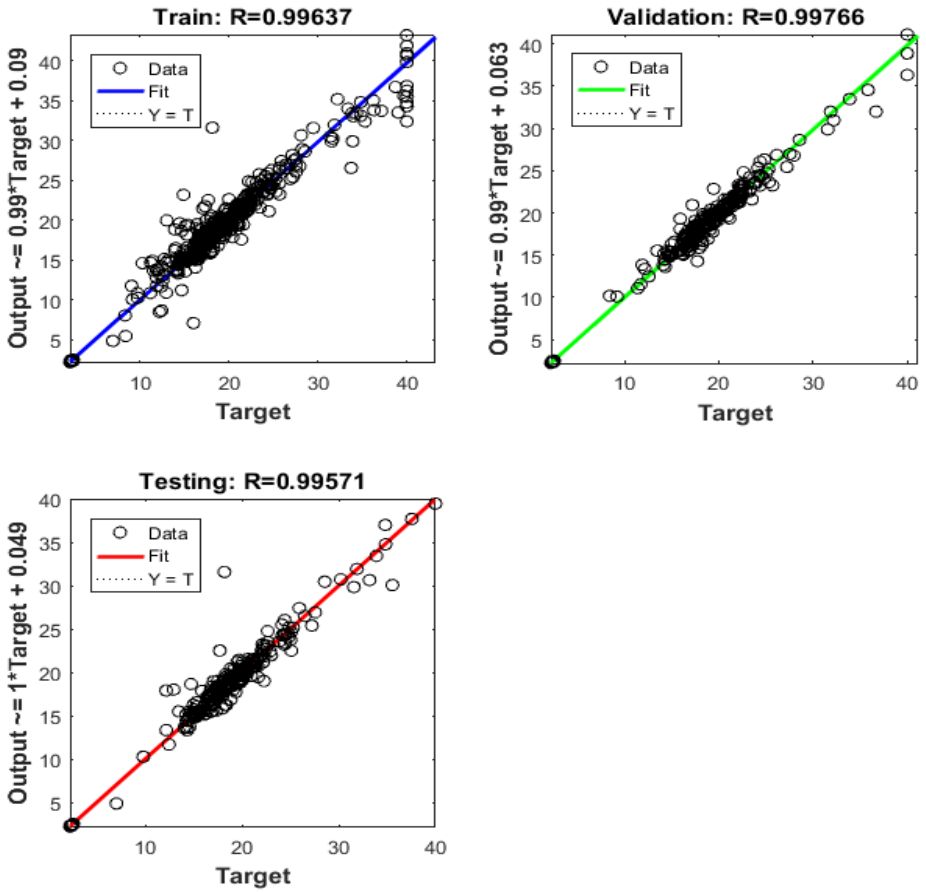

Figure 3-21 Linear regression of the targets relative to the outputs for the SDN performance in the training set

Figure 3-22 and Figure 3-23 show that the Mean Square Errors between the actual and predicted SDN output performance are 0.01 and 30 for each output, which indicates that there is a close relationship between these values.

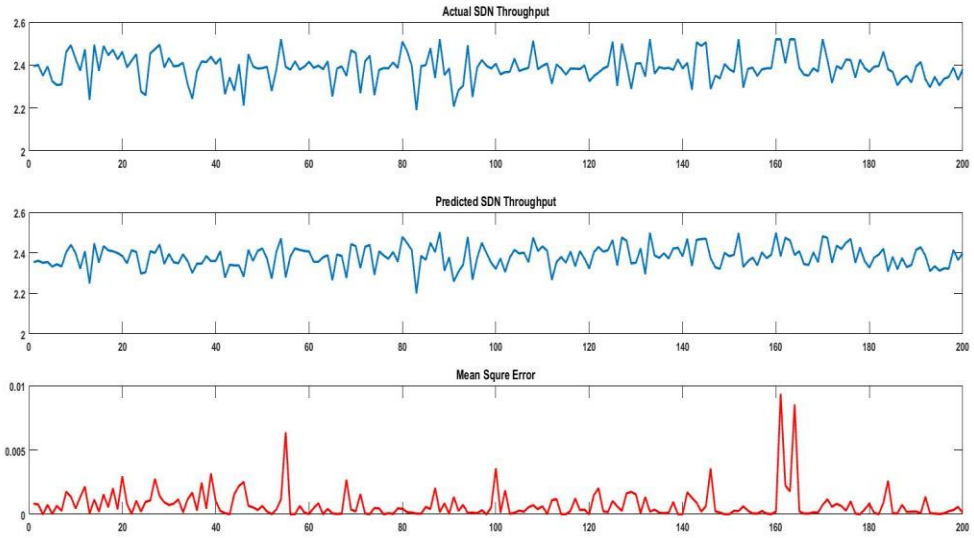

Figure 3-22 The actual and predicted Throughput performance

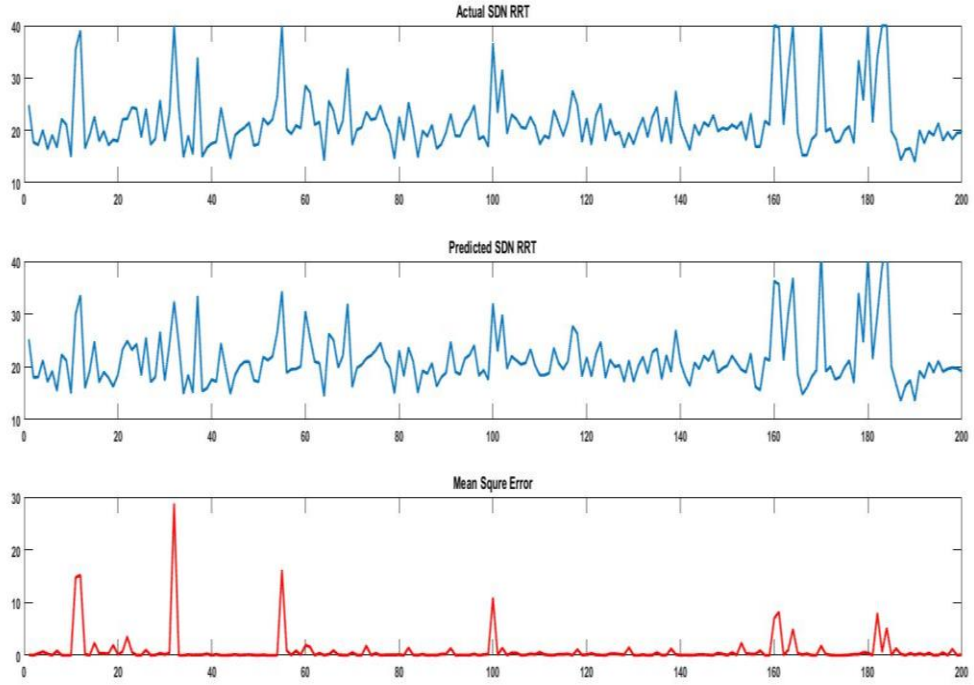

Figure 3-23 The actual and predicted Delay performance

3.7. **Summary**

In this chapter, an ANN has been proposed to find the correlation between the most effective input factors and performance output, with the former being the round-trip time, throughput and the flow table rules for each switch as well as a POX controller and OpenFlow switches. The ANN can be used very efficiently as a predictor, especially in an SDN controller with different traffic load, when the nature of the network is complex and highly nonlinear. The various topologies of SDN were trained and the ANN was tested by applying one hidden layer with different numbers of neurons. Moreover, LMA was used as learning algorithm in the feed-forward ANN structure. The results have shown that the network with one hidden layer gives a very acceptable MSE for all topologies. The proposed ANN could be used efficiently to improve the performance of an SDN by selecting the best input parameters, which is to be the subject of future work.

The dataset was divided randomly into training, validation, and testing sets, as 70%, 15% and 15%, respectively. The performances of the training algorithm, with different topologies of the network were trained to decide which structure performs better than the others, with Levenberg - Marquardt chosen as the training algorithm. The optimal structure of the network was chosen according to the best performance (minimum error between the predicted and actual matrix).

4. Chapter Four: Optimisation of a hybrid intelligent system for Software Defined Network

4.1. Introduction

Significant work has been carried out in the recent years aimed at generating systems demonstrating intelligence for realising optimised routing in networks. In this chapter, an Intelligent Agent based on two levels is proposed. At the first level, an ANN network is built to predict the performance of the SDN network, while at second, depending on the ANN output, the optimal path is explored using PSO.

A new hybrid intelligent approach for optimising the performance of Software-Defined Networks (SDN), based on heuristic optimisation methods integrated with the neural network paradigm, is presented. Evolutionary Optimisation techniques, such as Particle Swarm Optimisation (PSO) and Genetic Algorithms (GAs), are employed to find the best set of inputs that give the maximum performance of an SDN. The neural network model is trained and applied as an approximator of SDN behaviour. An analytical investigation has been conducted to distinguish the optimal optimisation approach based on SDN performance as an objective function as well as the computational time. After getting the general model of the neural network through testing it with unseen data, this model has been implemented with PSO and GAs to find the best performance of the SDN. The PSO approach combined with SDN, represented by an ANN, is identified as a comparatively better configuration regarding its performance index as well as its computational efficiency. The simulation has been run on different three topology structures.

4.2. Evolutionary-based Optimisation Methods

A vital aspect of modelling and control is the optimisation tool used to ascertain the best or near-best solution to a given problem, based upon a predetermined criterion known as the objective function or fitness. During the last 40 years, a set of algorithms has been developed that mimic the biological processes of evolution by natural selection. This evolutionary-based optimisation or evolutionary computing (EC) methodology provides a set of generation-based optimisation tools that use a higher-level (metaheuristic) trial and error process to generate a partial search algorithm (heuristic) in an attempt to solve stochastic problems [105], [106].

The original set of EC methodologies consisted of three exemplary forms, which were developed to tackle different problems, these being: (i) genetic algorithms (GAs) [107]; (ii) evolutionary programming [108]; and (iii) evolution strategies [109]. The subsequent generation of EC methodologies included a number of novel and outstanding forms, the most notable examples being those that evolved generations of data structures instead of string representations [110], and the GAs that advanced populations of programmes (genetic programming). A new generation of evolutionary-based optimisation methodologies has emerged in the last ten years including, for example, cultural algorithms, particle swarm optimisation (PSO) [111], and ant colony optimisation [112]. Every generation of EC optimisation tools has found a broad range of applications, particularly in conjunction with other intelligence-based approaches to the development of hybrid intelligent systems for advanced modelling and control.

There is insufficient space in the present thesis to examine all of these optimisation methodologies, hence the most popular ECs have been selected for attention. These are the genetic algorithms (GAs) and the particle swarm optimisation algorithm (PSO) methodologies, which are described in the following sections.

4.2.1. Genetic Algorithms

Genetic algorithms (GAs) are based on the Darwinian theory of biological evolution by natural selection and were first introduced by Holland in 1975 [98], GAs can be defined as self-adjusting global optimisation search algorithms; they function by mimicking the biological developments that are observed to arise in nature from genetic and evolutionary mechanisms. By association with the concept of natural selection, GAs employ code procedure and reproduction methods, involving bio-inspired operators, such as mutation, crossover and selection, to generate solutions to convoluted problems [113]. Each potential solution to the problem is treated as an individual characterised by data configuration consisting of two parts, namely, a chromosome and an objective function. The chromosome consists of a set of genes, each of which is allocated values termed alleles. The set of all individuals is termed a generation or population. The majority of GAs uses a population size that remains constant over the search period.

The objective function is used as the basis for selecting a set of individuals from the present generation, the parents, which are permitted to generate offspring. Hence, individuals with an above average objective function will have an above average likelihood of being

selected. Following selection, the reproduction processes of crossover and mutation are applied to the surviving individuals (the parents). While the crossover mechanism involves replication of genes from more than one parent to generate chromosomes for the offspring, the mutation mechanism only requires one parent. Consequently, the offspring arising by mutation generally displays a close resemblance to the parent, being distinguished only by a small number of altered genes[114].

The offspring (or children) are regarded as possible solutions to the problem and are therefore evaluated on the basis of the objective function, such that each child is assigned a fitness function. Individuals are eliminated on the basis of their fitness function, whereby those with a below average fitness function have an above average likelihood of being eliminated (or dying); this procedure being termed natural selection. Thus, the dead individuals are removed from the present generation to make room for the offspring. An intriguing aspect of GAs (and of ECs in general) is that the initial generation of individuals (possible solutions) does not need to be particularly good and in practice, it is usually made up of a set of arbitrarily assigned possible solutions. The search mechanism employed by GAs involves a number of iterations of the procedure represented by the flowchart in Figure 4-1.

Along with other intelligent-based approaches for the development of hybrid intelligent systems, GAs have been widely used for a significant variety of applications in modelling, estimation and control [115], [116].

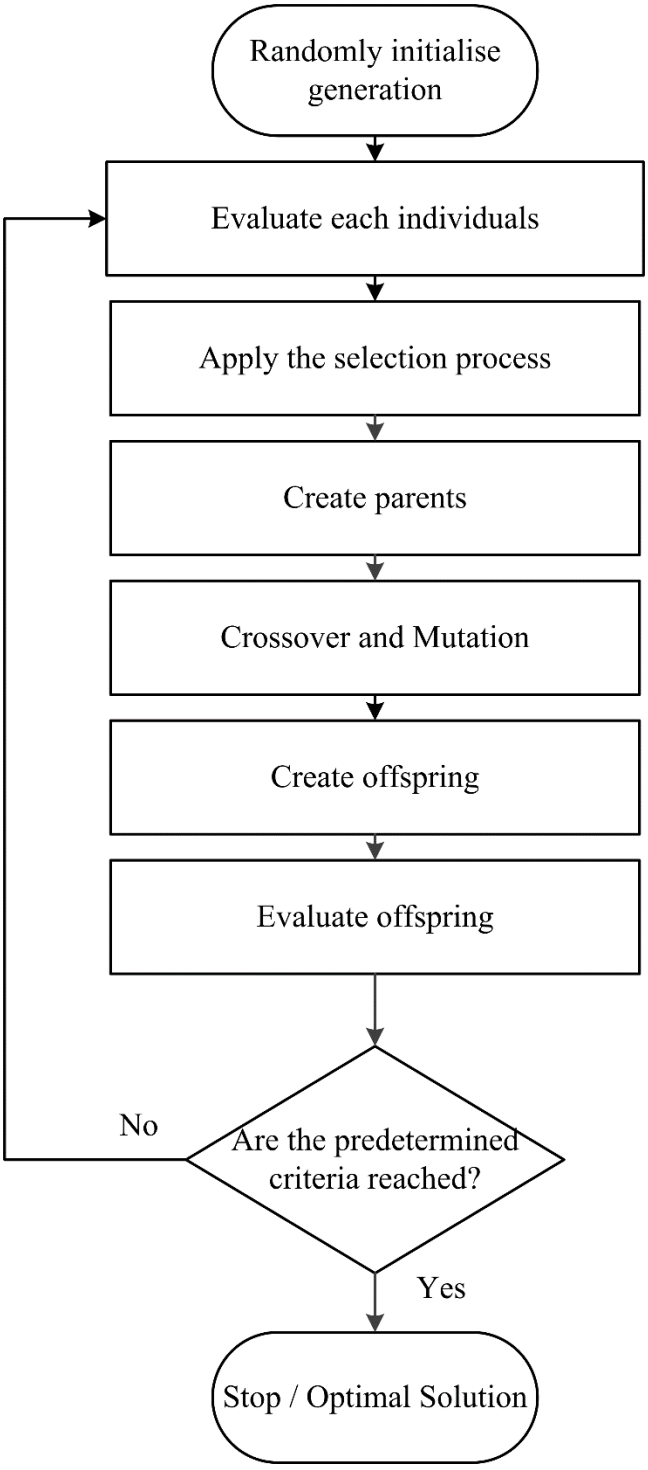

Figure 4-1 Flowchart of the search procedure employed in GAs

4.2.2. Particle Swarm Optimisation Algorithm

The Particle Swarm Optimisation (PSO) heuristic methodology was first put forward in 1995 and was revised in 2001 [117], [118]. As a relatively new subset of EC, PSO has proven extremely productive. The concerted functioning of decentralised and self-organised natural or artificial systems is examined by swarm intelligence (SI), generally by moving

towards a concession amongst individuals or particles within a population that are interacting in close proximity to one another as well as with others in the area. This type of simulation is usually drawn from examples in nature, especially biological systems, with particularly favoured examples being the group behaviours of ant colonies, animal herds, flocks of birds, schools of fish, and growing bacterial colonies [119].

Among the most intriguing aspects of PSO, as well as SI-based algorithms, in general, is the process of self-organisation, whereby overall order emerges out of an initially stochastic system via local interactions among individuals. Rather than being regulated by any internal or external agency, the process occurs completely spontaneously. A number of researchers have suggested improvements to the established PSO algorithm by incorporating useful concepts from other methodologies, examples of which include: quantum-behaved particle swarm optimisation (QPSO), an improved version developed by Jan and his team [120]; which incorporates the concepts of quantum mechanics theory; Kennedy's bare bones particle swarm optimisation (BBPSO) in which some of the operators have been changed [121]. and chaos-based particle swarm optimisation (CPSO), which combines PSO with the concepts of chaos theory [122]. While such attempts to enhance search performance continue, with more researchers suggesting novel modifications, the established PSO methodology also continues to find extensive application for a broad range of engineering challenges, particularly with regard to modelling and control [123], Consequently, the established PSO methodology has been applied to modelling and control in the present thesis, as detailed in the following chapter.

The procedural aspects of PSO can be outlined as follows. All candidate solutions (particles) are assigned random locations and random routes or trajectories within the search field. The direction of travel of each particle then gradually alters with confidence in the direction of its optimum previous position, so as to seek an even better one, with respect to the pre-set criteria or objective function. While the initial location and velocity of each particle is randomly assigned, the ensuing velocities are revised according to Eq. (4-1):

$$V_{i+1} = wV_i + C_1R_1 \times (Pb_i - x_i) + C_2R_2 \times (Gb - x_i) \tag{4-1}$$

where, V = particle velocity, x = particle position and w = inertial weight. R1 and R2 are independent random factors uniformly distributed from 0 to 1, and C1 and C2 are acceleration coefficients.

The new position of the particle is then calculated by addition of this velocity to the previous position, according to Eq. (4-2):

$$x_{i+1} = x_i + V_{i+1} \tag{4-2}$$

Thus, Eq. (4-1) evaluates the new velocity of the particle with reference to its previous velocity and with respect to its actual position relative to both its local and global ideal positions (Pb and GB, respectively). The particle then moves to a new location within the search space, according to the formula in Eq. (4-2). This is the established PSO search methodology, which is presented as a flow chart in Figure 4-2.

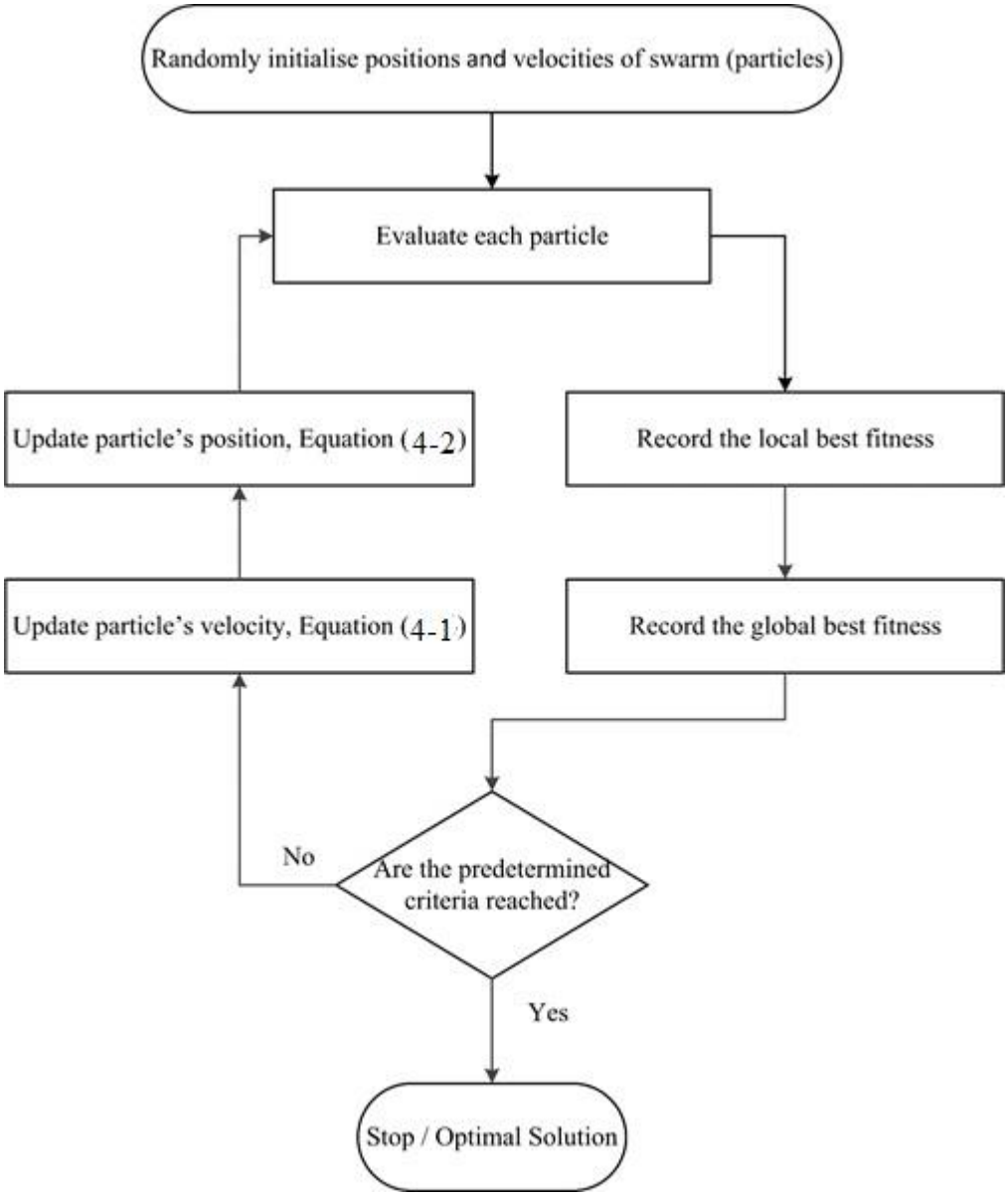

Figure 4-2 Flowchart of the established PSO approach

4.3. Simulation Test Procedure

Significant research has been carried out in recent years to generate systems exhibiting intelligence for realising optimised routing in networks.

A. SDN Simulation

SDN simulation was performed using the Mininet platform to collect all datasets of inputs and outputs. Three different topologies for SDN were used, as described in the previous chapter. In addition, simulation experiments were carried out using a POX controller and monitoring of the flows was required for all events. Regarding which, in order to build the learning system, the SDN controller gives the orders to an SDN switch to monitor the

flows, and it keeps on monitoring them to detect the events. When the flows are monitored, the entire event, whether it occurs frequently or periodically, is stored in the database to be used in the ANN learning system. The flows/inputs include the rules coming from the controller. In turn, the output will be represented by the system throughput and the network delay. Consequently, the data that are collected from the ANN learning system are considered as being efficient inputs into optimisation algorithms. This work presents a new method for minimising the load of the SDN switch by the controller changing it adaptively, instead of waiting for the events all the time. This means that the switch will not send unmatched packets to the controller all the time, because the latter will be able to send the rules directly owing the intelligent agent that studies the behaviour of the network for optimising the routing rules all the time on behalf of the controller.

B. Optimise Identification for SDN performance

This section includes the intelligent layers for identifying the SDN rule parameters so as to achieve the best performance for maximum throughput and reducing the delay as follows.

First, the SDN network behaviour or performance is predicted using an ANN with an exhaustive search for selecting the optimal ANN topology and the Levenberg-Marquardt (LM) learning algorithm. When the ANN training started, the dataset had been pre-processed by normalising it into a range between -1 and 1. The dataset of the inputs and outputs was divided randomly into three subsets: a training set, valediction set and a testing set. The first subset was for establishing the gradient as well as updating the network weights and biases. The error regarding the second subset was observed during the training development. The validation error is usually reduced in the initial training phase, as is the training set error. Nevertheless, when the network overfits the data, the error in the validation set invariably starts to rise. In the current case, the network parameters were saved at the minimum of the validation set error. Input values need to be normalised in the range [-1, 1], which corresponds to the minimum and maximum actual values.

Subsequently, testing the ANN requires a new independent set (test sets) in order to validate the generalisation capacity of the prediction model. A multilayer feedforward network was implemented to estimate the performance of the SDN. In order to obtain a maximum accuracy of prediction, the network was trained in different topologies. For each

network architecture, the training was run ten times for various random initial weights and biases using the Levenberg Marquardt algorithm (LMA). After investigating the performance of different architectures using the exhaustive search method, the best trained ANN with one hidden layer was found as giving the comparably better performance of MSE, as explained in the previous chapter. It is notable that the ANN model was efficiently accurate and hence, the network was accepted as a general model to be integrated, as the next step, with GA or PSO, so as to produce the proposed intelligent hybrid system.

Second, there is the introduction an intelligent optimiser for the SDN network to maximise the throughput and minimise the delay. Two evolutionary algorithms (EA) are used to find the optimum rule parameter set for the OpenFlow rules. Both the EA algorithms, namely, PSO and GA, are compared in terms of effectiveness (finding the true global optimal solution) and computational efficiency. The performance comparison of the GA and PSO is implemented using MATLAB.

4.4. Optimisation for SDN controller using PSO against GA

GA and PSO were integrated separately with the trained ANN model to select the optimal set of inputs that make the network work as efficient as possible. The metadata of both optimisers were selected in a similar manner, as far as possible, in order to make a fair comparison. These metadata included the number of generations or particles and the number of iterations and so on. The metadata of the PSO are shown in Table 4-1, while those for GA are provided in Table 4-2**.**

Table 4-1 Metadata of PSO used to optimise the SDN controller

Parameter	**Value**
Number of particles (size of the swarm)	100
Maximum number of iterations	100
Cognitive acceleration (C_1)	1.2
Social acceleration (C_2)	0.12
Momentum or inertia (W)	0.9

C1, C2 and *W* were chosen after conducting extensive simulation experiments. PSO was programmed from scratch to work as an optimiser tool in MATLAB®, while the default

of the GA metadata was implemented as provided by the Optimisation Toolbox in MATLAB® platform.

Table 4-2 Metadata of GA used to optimise the SDN controller

Parameter	Value
Population Size	100
Maximum number of iterations	100
Function tolerance	0.0001
Crossover fraction	0.8
Migration fraction	0.2

As stated earlier, because of the randomisation of the candidate solution at the initial stage, both optimisers, namely GA and PSO, were run 20 times to achieve the best performance optimisation result. In addition, it showed superiority regarding the transient response properties, including the rise time, overshoot, settling time and steady state errors for both test scenarios. Figure 4-3 shows the architecture of the system including the trained ANN model as well as PSO and GA as optimisers.

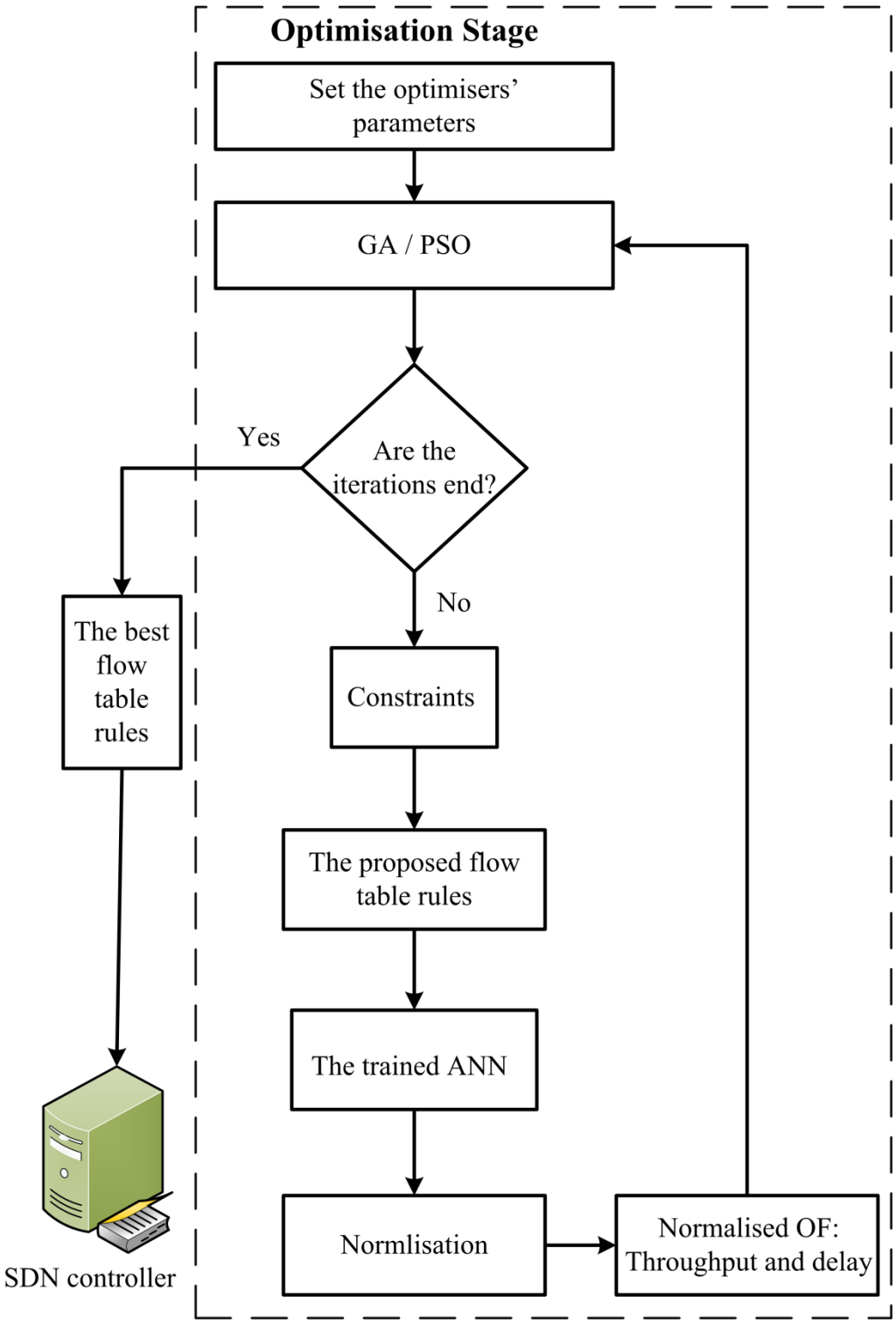

Figure 4-3 the schematic diagram of the proposed intelligent optimisation for SDN performance

The simulation experiments were carried out by the MATLAB platform. PSO was employed to find the optimal structure of the network and the best operational parameters of

the flow table, followed by the same process for the GA algorithm with their performance being compared in the form of a bar chart, as shown in Figure 4-4. It is clear that PSO performed better that GA in 13 out of 20 runs, with the eighth run of PSO showing the best objective function at 96.321%. Whilst GA performed better in just 7 out of the 20 runs and its best objective function is 94.846% in the run number 19, as shown on the bar chart. Also, the average run of PSO took about 3.2 (hours), while GA took approximately 6.7 (hours) to find the optimal set of table flow rules. A comparison of the results of the SDN performance is provided in Table 4-3 and Figure 4-5. PSO outperformed GA regarding the performance and computational time, with the convergence being faster with fewer iterations and the obtained fitness was higher. The results provided were obtained after running PSO/GA 20 times.

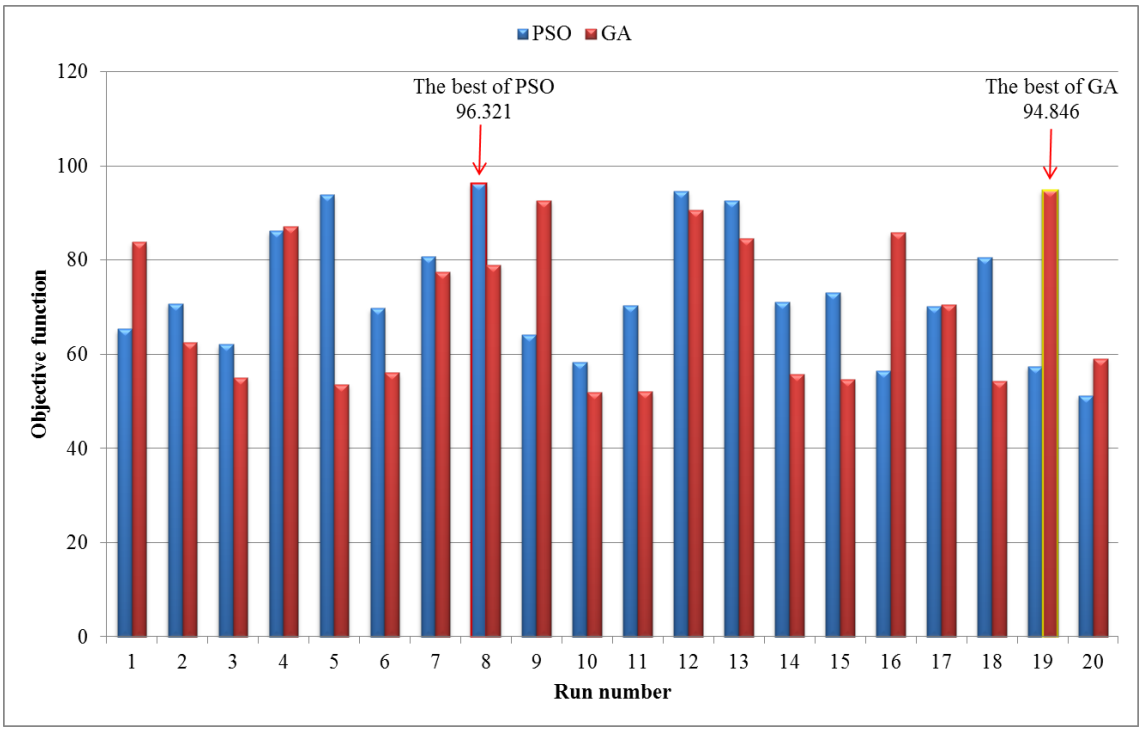

Figure 4-4 Different number of runs of GA and PSO to tune the PID-like fuzzy controllers vs. the objective function

Table 4-3 Performance and computation time comparisons for GA and PSO

Optimisation method	The best objective function %	The average Computation Time
PSO	96.321	3.23
GA	94.846	6.75

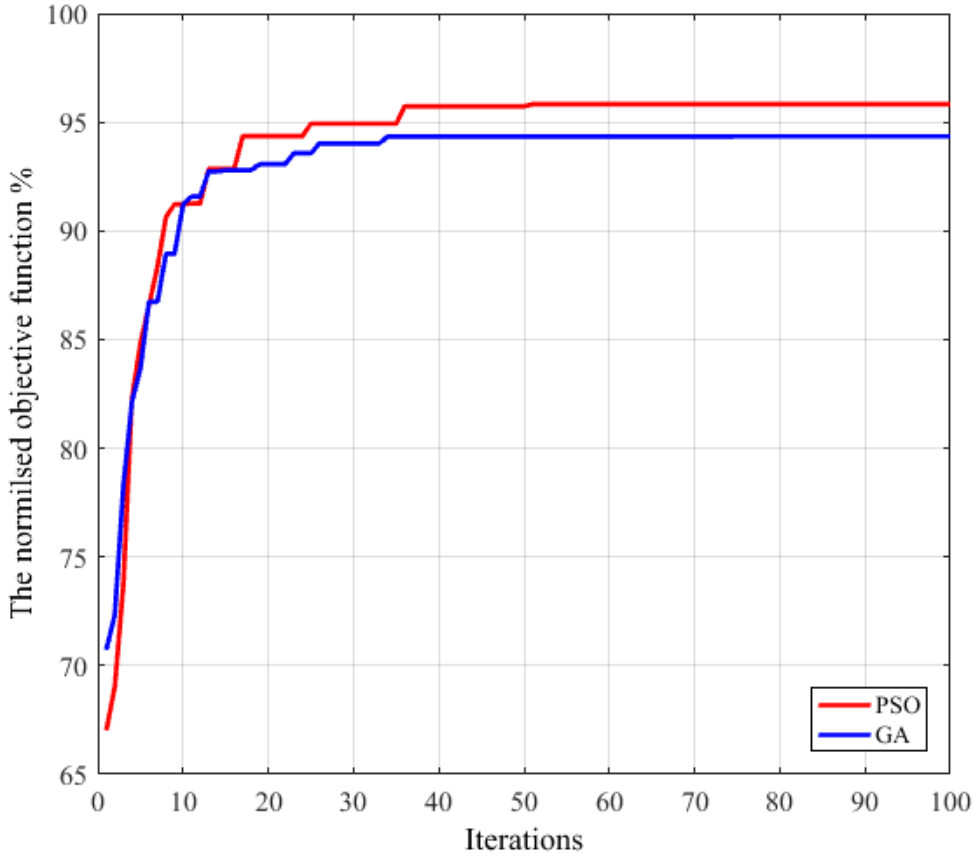

Figure 4-5 Convergence comparison of GA and PSO

This leads to adoption of PSO as the optimiser in the SDN network due to its high performance, as shown in Figure 4-5. The measurements of the operative network serve as inputs to the ANN network to predict the performance of the SDN network represented by the throughput and the delay which are the significant QoS parameters in the network. The optimisation techniques represented by PSO are used to achieve high throughput and minimum delay. The PSO algorithm provides the optimal Flow Rules parameters for achieving these two goals. The measurements program captures the existing load on the

network as well as the performance of the current topology, whereas the ANN model predicts the outcome of the path-selection process running on the switches for a given setting of the SDN network topology.

The optimisation techniques have been used to be allow for identifying the parameter settings of the flow rules in the switches that satisfy the network's performance goals. Once good performance has been identified by the optimiser, the cognitive system can change the configuration of the flow table parameters, depending on the previous knowledge of the network. Then, the program will change or produce new parameter values fof the flow rules to compute new paths for forwarding data packets through the SDN network, as predicted by the model. Finally, the SDN controller receives changes in the flow table parameters from the intelligent agent, which means it can to send these directly to the switches instead of having to waiting for the first packet to react to this action.

4.5. **Summary**

In this chapter, a hybrid intelligent system has been proposed for modeling and optimising a Software-Defined Network (SDN). An artificial neural network (ANN) with a single layer in the hidden zone has been trained to map the inputs and the performance of the network. The results have shown that the network gives a very acceptable MSE. The generality of the trained ANN model has been checked by applying unseen data as a test set and subsequently, this general ANN model has been synergised with evolutionary algorithms (EAs) to produce the proposed intelligent hybrid system That is, GA and PSO have been employed as EA optimisers to find the optimal set of inputs to the SDN based on the maximum performance as a fitness function. According to the obtained results, PSO outperforms GA regarding performance, convergence and computational time. PSO showing the best objective function at 96.321%, whilst GA shows its best objective function at 94.846%. Finally, the average run of PSO took about 3.2 (hours), while GA took approximately 6.7 (hours) to find the optimal set of table flow rules.

5. Chapter Five: Smart Flow Steering Agent in Software Defined Networks

5.1. Introduction

To ensure fault tolerance and distributed management, distributed protocols are employed as one of the major architectural concepts underlying the Internet. Moreover, inefficiency, instability and fragility could be potentially overcome with the help of the novel networking architecture called Software Defined Networking. The main property of this architecture is the separation of the control and data planes. To reduce congestion and thus, improve latency and throughput, there must be homogeneous distribution of the traffic load over the different network paths. This chapter presents new Intelligent Agent we call it smart flow steering agent (SFSA) for data flow routing based on current network conditions.

To enhance throughput and minimise latency, the SFSA distributes network traffic to suitable paths, in addition to supervising link and path loads. A scenario with a minimum spanning tree (MST) routing algorithm and another with open shortest path first (OSPF) routing algorithms were employed to assess the SFSA. In comparison to these two routing algorithms, the suggested SFSA strategy produced a reduction of 28-31% in the packets dropped ratio (PDR), a reduction of 36% in packets send to controller according to the traffic produced, as well as a reduction of 23% in round trip time (RTT). The Mininet emulator and POX controller were employed to conduct the simulation. Another advantage of the SFSA over MST and OSPF is that its implementation and recovery time do not exhibit fluctuations. The smart flow steering agent will open a new horizon for deploying new smart agents in Software Defined Networks that enhance network programmability and management.

5.2. Traffic Engineering and Routing Mechanism

Improvement of network performance and traffic delivery relies greatly on traffic engineering, which in turn depends on route optimisation, involving the discovery of routes that could help attain the target network performance [124][125]. Important elements of traffic engineering in IP networks are the quality of service (QoS) and resilience mechanisms. Alongside bandwidth requirements, QoS assurance (i.e. end-to-end delay, jitter, likelihood of packet loss, and energy efficiency) is also important in many of the novel multimedia applications. Meanwhile, IP networks are often affected by various kinds of failures, including node or link failure, and rapid resilience mechanisms are needed to address these

[126][127][47]. Under such circumstances, preventing failures from disrupting network performance and resource usage is the main objective of traffic engineering solutions. Basic routing models, consisting of shortest path and load-balancing strategies with traffic divided evenly into a specific number of paths of equal cost, represent the cornerstone of the majority of IP-based traffic engineering solutions [47].

In the following part, several popular routing algorithms and their impact on network traffic performance are discussed.

5.2.1. Open Shortest Path First (OSPF)

An OpenFlow interface is included in numerous new Internet routers, which are also compatible with a hybrid OpenFlow/OSPF mode. For optimal forwarding of packets, the distributed legacy routing protocol is usually applied by hybrid control plane frameworks and is augmented with high-priority rules by the SDN controller to allow for more elaborate routing configurations [128]. In the case of IP networks, the OSPF algorithm frequently serves as an interior gateway protocol (IGP) based on the link state. A network topology graph is created by OSPF, with information about the link state derived from existing routers. Furthermore, it applies a hop-counts technique to determine the shortest path for packet routing.

5.2.2. Minimum Spanning Tree (MST)

To prevent the issue of packet broadcast storm in a legacy network with loops, a spanning tree is created with the IEEE 802.11d distributed spanning tree protocol. The same issue can be addressed in an SDN network through the central creation of a spanning tree based on the overall information of network topology gathered by the SDN controller [129]. The basic bridge protocol that was initially devised for loop prevention in a bridge network is known as the Spanning Tree Protocol. Data are disseminated through the tree constructed by this protocol and encompass all network bridges, with only the links included in the tree being allowed. If the bridged network breaks down, it can repair itself through activation of the links that have been previously blocked. However, the algorithm presents limitations in that recovery in the event of failure does not occur fast enough and backup links can only be used for forwarding under conditions of failure [124].

5.2.3. Proposed SmartFlow Steering Mechanism

Quality deterioration is mainly caused by network congestion, which occurs when the traffic is not distributed homogeneously or network resources are insufficient. This issue is compounded by the fact that current routing algorithms are designed to detect just the shortest paths from a source to a destination, and that routing and congestion control do not work together [130]. Packet drop ratio (PDR), latency (end-to-end delay) and error rate are all heightened by the rise in queuing delay due to congestion. Building on the intelligent agent, the approach put forth in this study is aimed at identifying the best path from a source to a destination in an SDN with the best throughput and shortest delay. The information collected by the OpenFlow switches is used by the SDN controller to create a general network topology.

An agent is a computer system that is capable of independent (Autonomous) action on behalf of its user or owner. An intelligent agent contains an architecture, which includes the device that it executes on, and a program that gives it the ability to improve the behaviour of this architecture. As mentioned earlier, in this thesis the SDN network represent the architecture, the performance of which is to be improved using an intelligent agent. The intelligent agent consisting of two parts, namely, an ANN prediction system integrated with a GA or PSO.

The intelligent agent has four main features [131]:

Percepts: the inputs of the system, **Actions:** The outputs of the system, **Goals:** What the agent is expected to achieve and finally the **Environment:** What the agent is interacting with.

In the proposed system, as shown in Figure 5-1, the SFSA percepts the SDN network parameters, which include the flow table rules and the performance parameters and it then sends a new flow table to the controller as an action. The environment is the entire SDN network topology and the performance parameters. Finally, the goal of the SFSA is to enhance the performance of the SDN network, as described in the following sections.

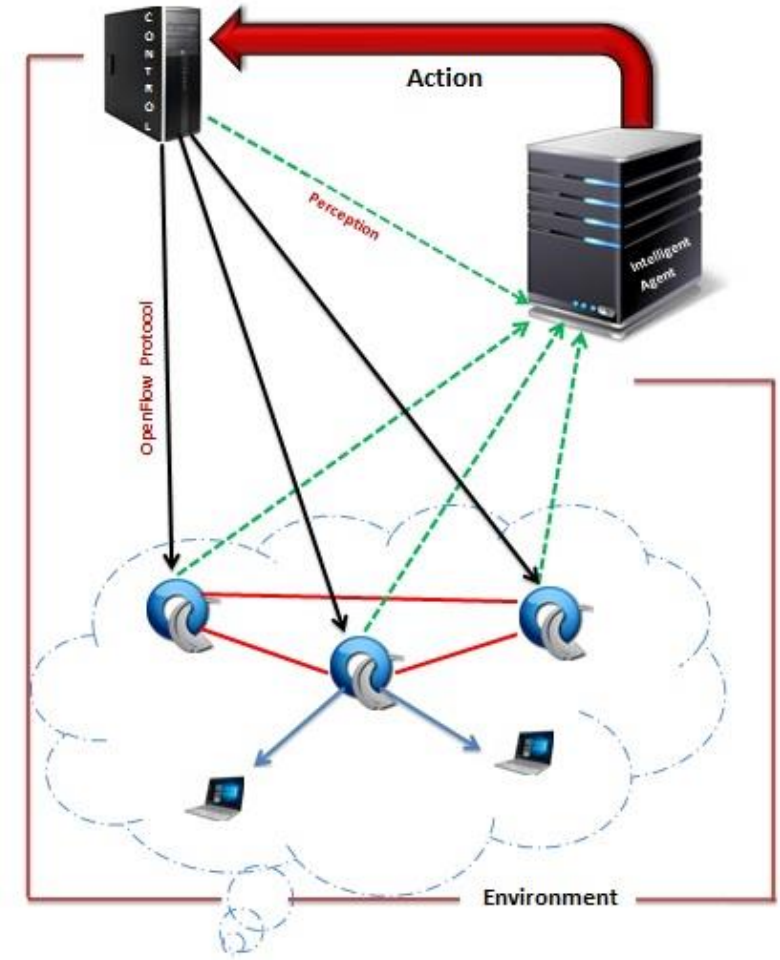

Figure 5-1 The proposed architecture of a smart flow steering agent

The detection information sent by every switch to the SDN controller enables the latter to construct a general network topology. Flow table entries are generated by an agent application with a smart flow steering mechanism working within the SDN controller and subsequently, the flow tables are sent to the OpenFlow switches. Applying a technique underpinned by the intelligent agent, the smart flow steering mechanism determines the shortest path for every route, thus achieving packet routing. For routing to be considered optimal, it must prevent network congestion by employing existing network resources for effective traffic distribution. In SDN, the SFSA can achieve significant improvement in PDR and end-to-end delay, reduce packets sent to the controller and enhance the throughput in the network.

5.3. Evaluation and results Simulation Analysis

In this section, the simulation parameters, simulation screenshots and evaluation of SFSA with other routing protocols are covered. A VMware Player virtual box and MATLAB software for the Mininet emulator and algorithm analysis were employed, respectively, to perform the simulation scenarios. There are two sequential procedures undertaken by the SDN controller. Prior to selection of a routing algorithm, it conducts network discovery to uncover the network topology. The identification of the OpenFlow switches is followed by application of the SFSA in the SDN controller and provision of optimum flow tables to the OpenFlow switches for storage and the initiation of data routing. Three scenarios concerned with end-to-end delay, packets sent to the controller, throughput and PDR are explored in this study via the following steps:

- Creation of the general network topology and execution of the default routing algorithm OSPF or MST in a POX controller by the Mininet;
- Collecting all the data set from the SDN network topology, which include the flow table, the throughput and the latency;
- Analysing the collected data and pre-processing it prior to the first stage of the intelligent agent, which includes the ANN. The data set is divided randomly into three subsets: (70%) representing the training subset, (15%) being the testing subset and (15%) representing the validation subset;
- Training the designed ANN using the collected dataset and testing for the best topology of the network (the number of nodes in the hidden layer);
- Employing PSO to find the best set of flow table rules by integrating it with the trained ANN.

  .

Figure 5-2 shows the first stage of the intelligent agent, which includes percepts of the input from SDN environment. The inputs include the flow table rules for the network after running for 180 minutes, the flow table set containing the switch number, input port, Ethernet source, Ethernet destination, TCP source, TCP destination and action. They also include the performance parameters for every rule, including the throughput and the delay, these two parameters were collected all the time and registered every five minutes. After building the prediction system with an ANN network, the agent will have a global picture about the next performance of the SDN network.

The second stage of the SFSA agent is to choose the optimal flow table rules set to achieve the best performance of the SDN network. This stage includes the PSO algorithm after having compared it with GA in the previous chapter and finding that it delivers superior performance. Since the range of the SDN performance indexes (throughput and delay) varies widely, objective functions will not work properly without pre-processing, such as normalisation. A normalisation process has been used to rescale the range of the two indexes onto a scale ranging between 0 and 1, which is also called feature scaling [132], as presented in the following formula:

$$\acute{x} = \frac{x - \min(x)}{\min(x) - \max(x)} \qquad (5\text{-}1)$$

where, x are the original data and $\acute{x}$ are the normalised data. So, the objective function used in PSO for optimising the performance of the SDN network to find the best throughput and shortest delay can be expressed as:

$$OF = \frac{1}{2} \text{ (Throupghput – Delay)} \qquad (5\text{-}2)$$

Figure 5-3 shows the Optimal OpemFlow rule table selection set scheme of the SDN network. The optimiser suggests that the OpenFlow rules table should be set to reach the maximum throughput and minimum delay. The OpenFlow rules table set was fed into the ANN as input in, while the outputs were throughput and delay. As detailed earlier in this part, the ANN was trained in order to predict the actual output and to prepare the data for the optimisation stage. The objective function aims to to obtain the optimised openFlow rules table set that gives maximum throughput with minimum delay for a given node number.

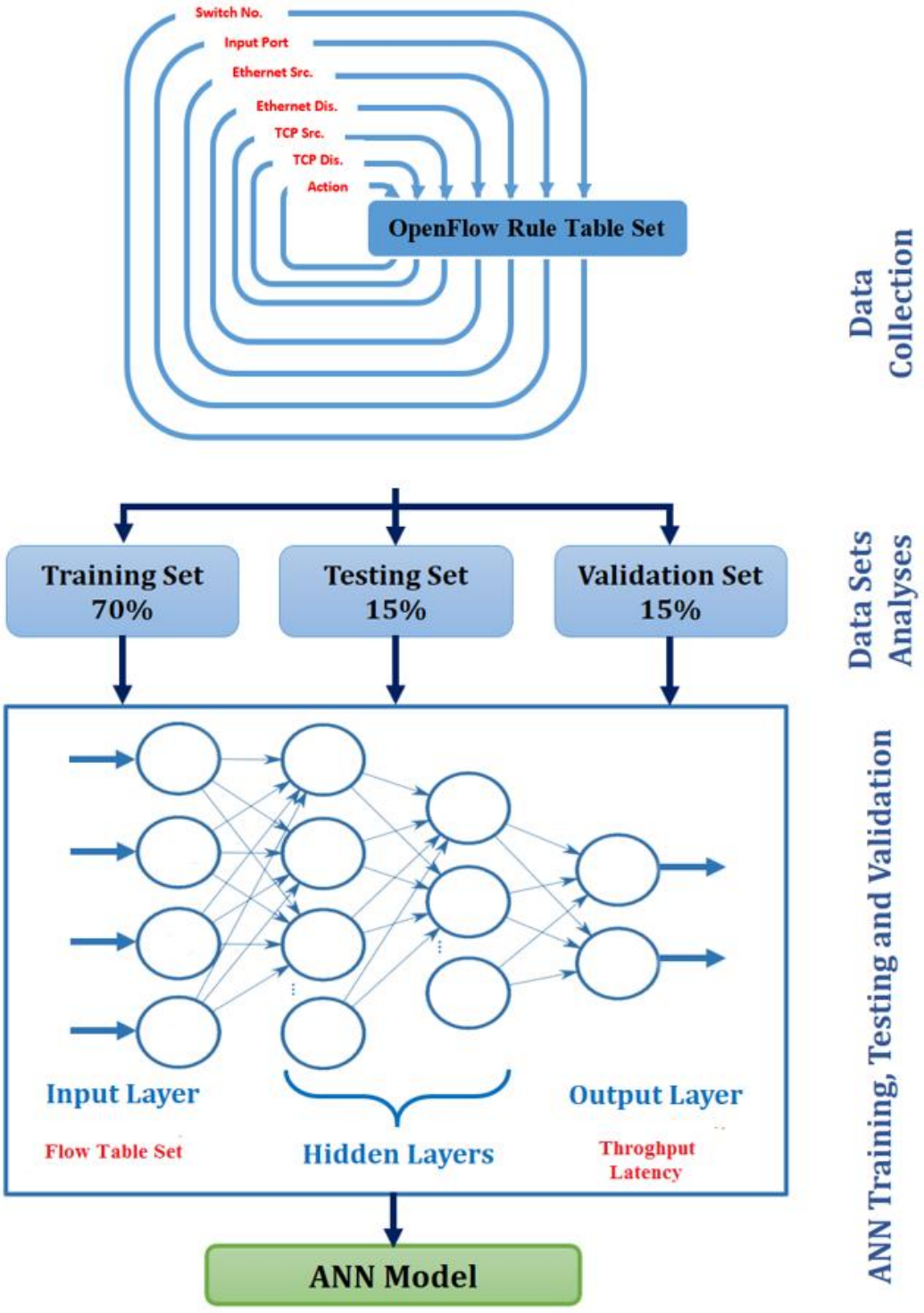

Figure 5-2 First stage of SFSA percept; the input from SDN environment

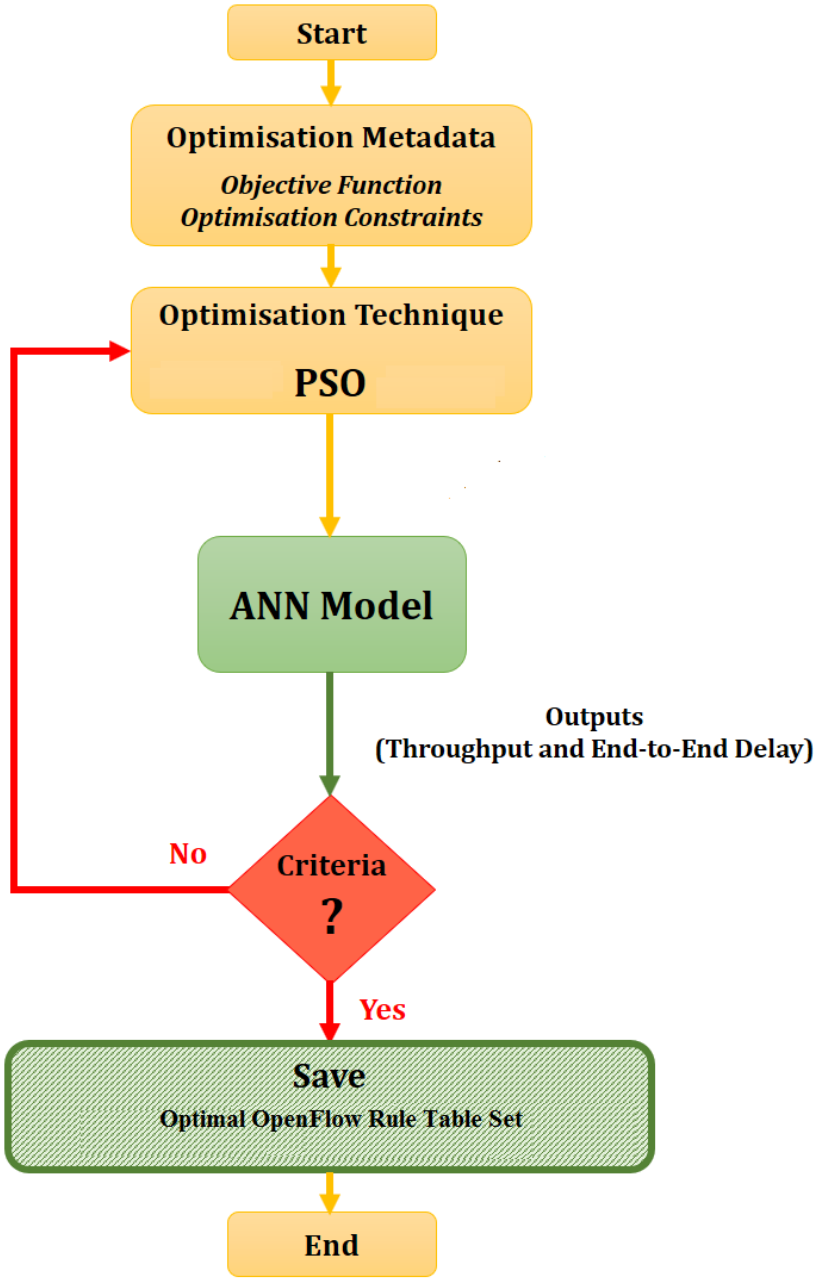

Figure 5-3 OpenFlow Rule Table Set optimisation technique

A. Mesh Topology Scenario

For the Mininet simulation, as shown in Figure 5-4 and Figure 5-5, the mesh topology was made up of five OpenFlow switches with ten links, and fifteen switches with thirty links, thus ensuring their connection.

Figure 5-6 illustrates the round-trip comparison and it can be seen that in contrast to the default Mininet routing algorithms, the SFSA is associated with less delay, constituting a 23% decrease in packet round trip time (RTT) even when the generated packets across the

network have been increased. Moreover, Figure 5-7 presents the algorithm execution time and failure recovery time, indicating that, when compared with the other algorithm, the SFSA can identify the optimal path from all possible paths much faster in the event of network failure.

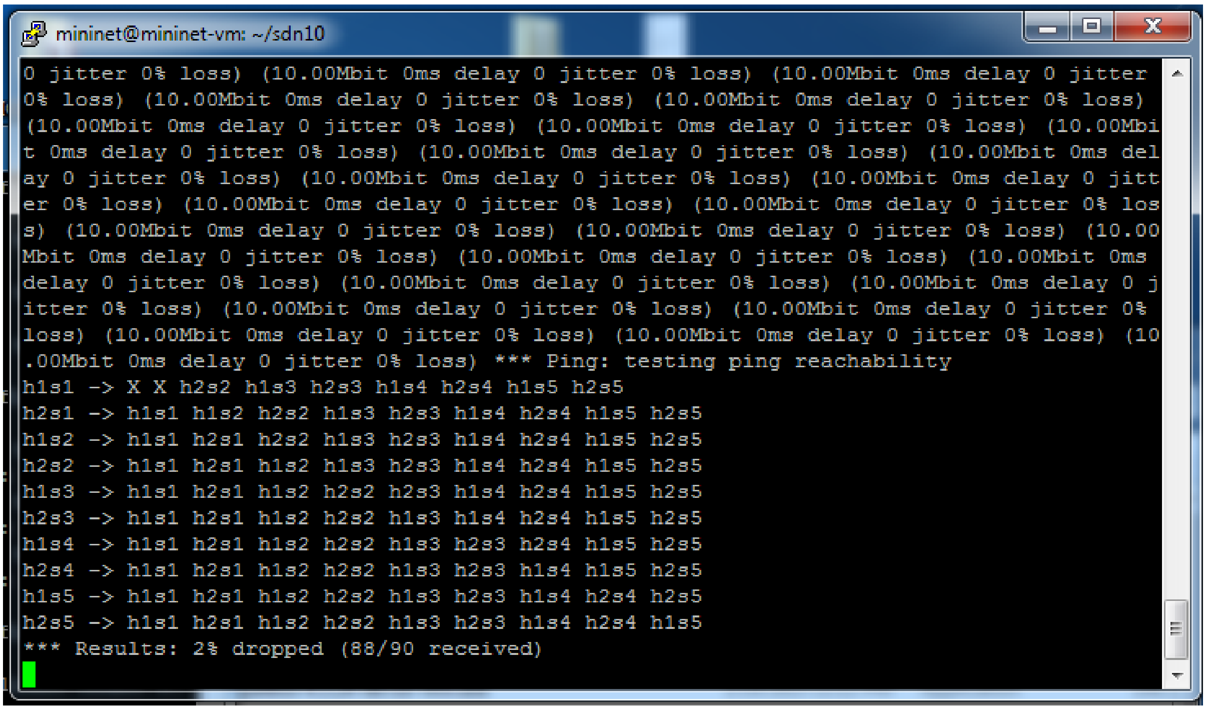

Figure 5-4 Mesh topology simulation run with five switches

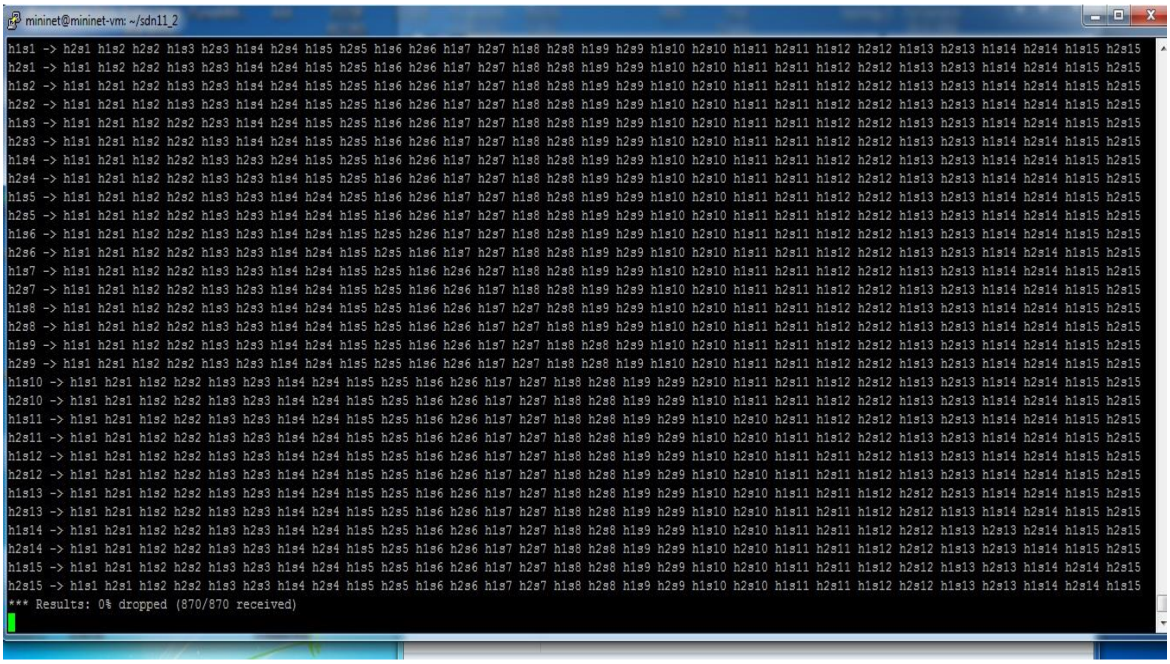

Figure 5-5 Mesh topology simulation run with fifteen switches

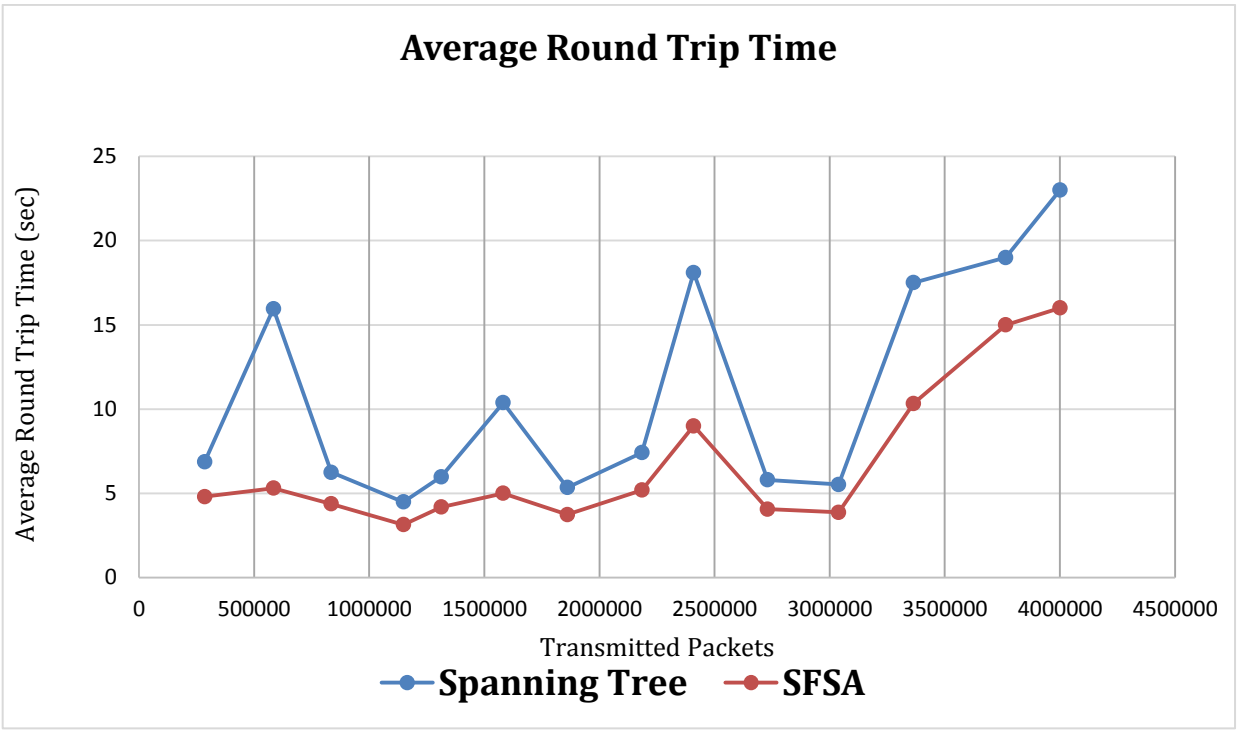

Figure 5-6 Mesh topology round-trip comparison

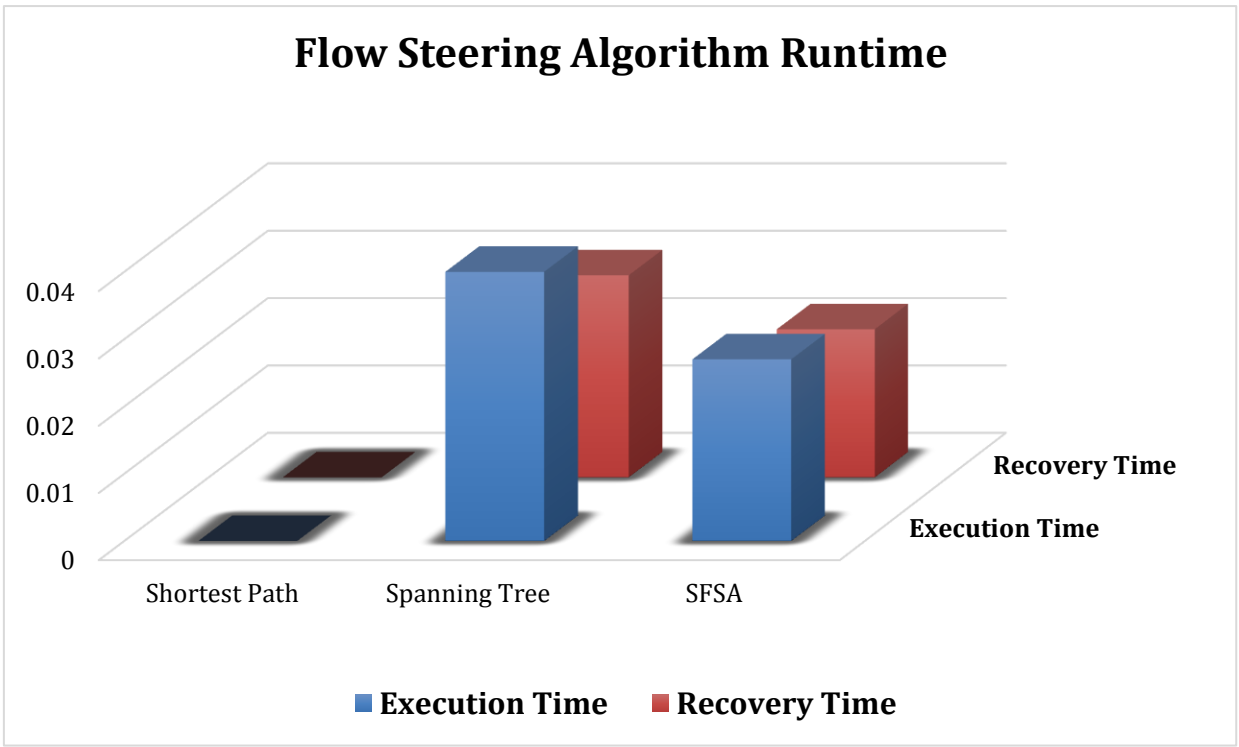

Figure 5-7 Mesh topology scenario for SFSA execution and recovery time

The values associated with the worst-case scenario in mesh topology are listed in Table **5-1**. It can be seen that in the selection of the optimal path between two nodes, the SFSA successfully balances path length and congestion cost. As a result, the end-to-end delay is 15-30% lower than in the case of the default Mininet routing algorithm, which is influenced by the traffic load and node position.

Table 5-1

MESH TOPOLOGY WORST CASE SCENARIO

Source	Destination	Hops Count	Delay Cost	Algorithm
1	4	N/A	N/A	OSPF
1	4	1	4.24	**SFSA**
1	4	2	5.77	MST
3	4	N/A	N/A	OSPF
3	4	2	3.41	**SFSA**
3	4	2	3.41	MST

In SDN architecture, control messages go between the controller and the forwarding devices to form communication networks. If there is only one centralised controller with global network visibility, then there will be excessive control traffic overhead. In addition, the overhead will be further increased by the SDN controller when the events as well as the application specific control traffic are increased. If the overhead is not controlled properly, it can cause various scalability problems for the networking devices, controllers, and the network itself, including slow message processing, potential message drop, delayed root cause analysis, and late response against urgent problems. In the proposed system the control messages are reduced in both topologies by about 35% as shown in Figure 5-8 and Figure 5-9.

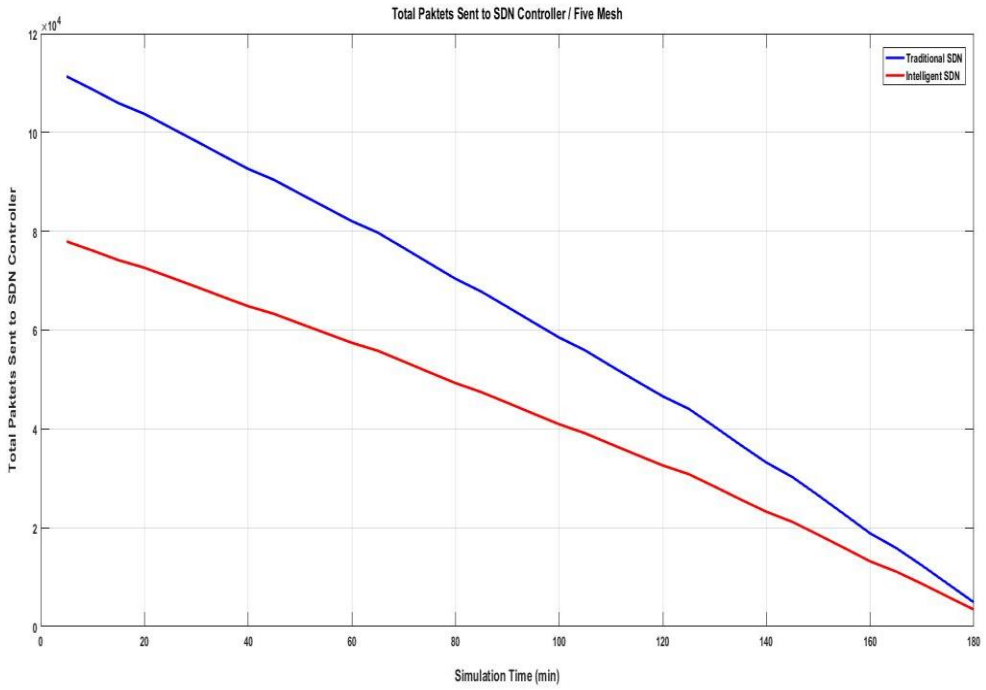

Figure 5-8 Number of messages sent to the controller from the switches in a five switch topology

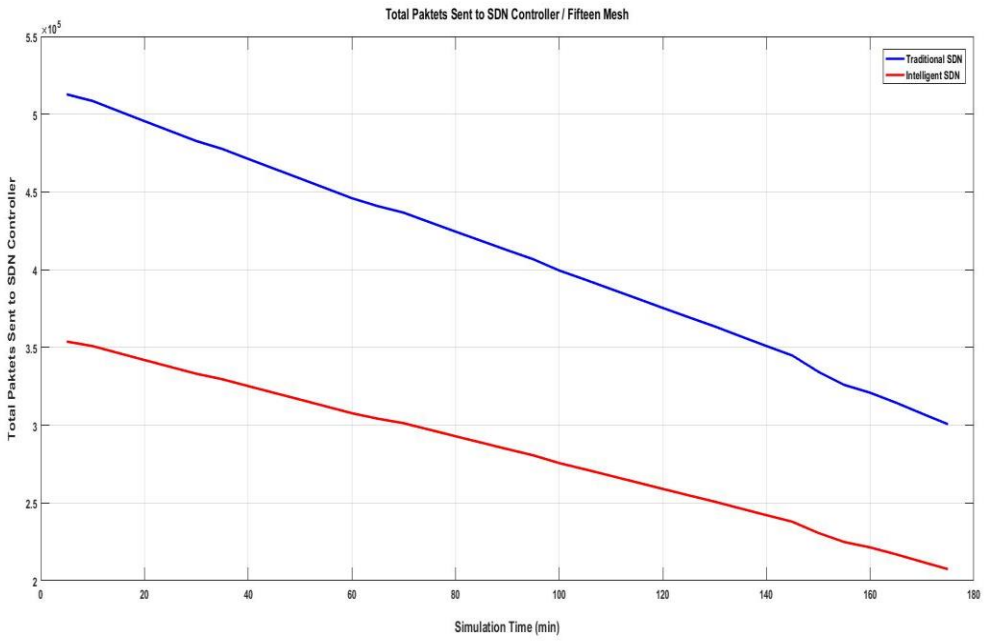

Figure 5-9 number of messages sent from controller to switches in 15 switches topology

Iperf was used as a network bandwidth measurement tool to test the TCP bandwidth performance, with the testing time set to 180 minutes. The two path finding algorithms, MST and SFSA, were executed one after the other in a POX Controller with the two different topologies. The Iperf tool provided the bandwidth usage in the two scenarios where the algorithms were executed by the POX Controller. The results are shown in Figure 5-10 and Figure 5-11.

The maximum throughput was found to be for the SFSA based algorithm rather than the MST. When both are compared, with throughput enhancement of approximately 10 %, it is evident that the SFSA based algorithm performs better than both the other chosen algorithms.

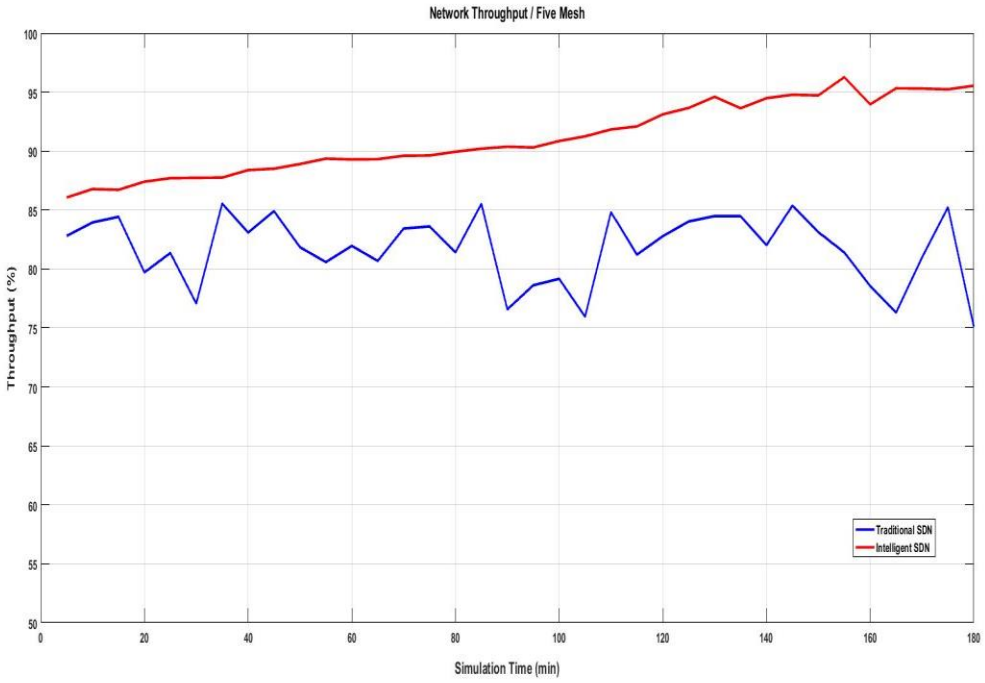

Figure 5-10 The maximum throughput for both algorithms in a five switch topology

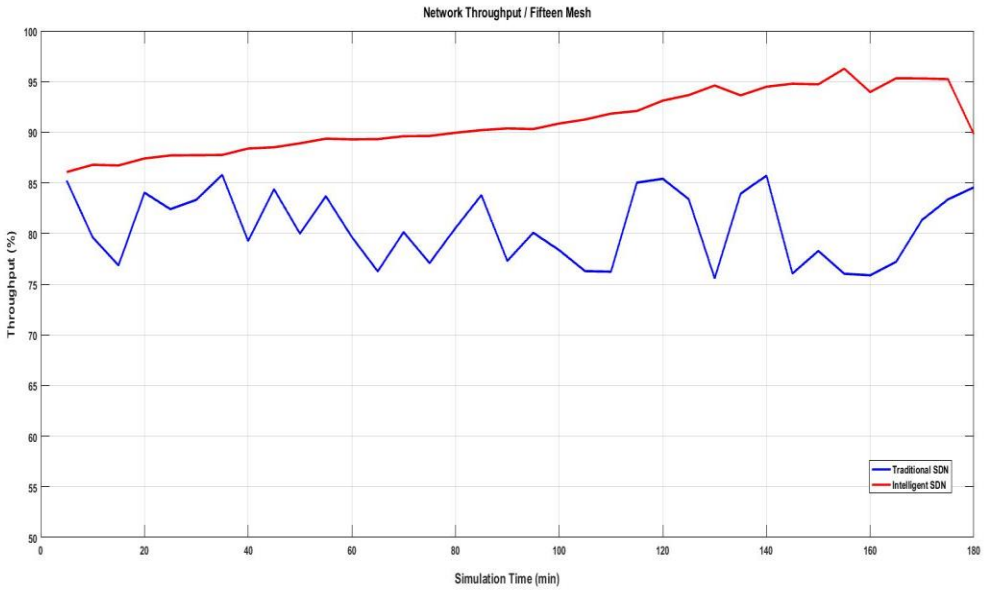

Figure 5-11 The maximum throughput for both algorithms in a 15 switch topology

The simulation results clearly demonstrate the effectiveness of the SFSA based algorithm in finding the optimal path. Another point to be considered is that the dropped packets ratio, as can be seen in Figure 5-12 and Figure 5-13, demonstrates that with the SFSA algorithm the number decreases by about 28% in both topologies. For all the above scenarios, the SFSA algorithm enhances the performance of the SDN network plus giving more reliable network performance by preventing congestion and the bottleneck problem. The SFSA algorithm also provides the network with new intelligent controller such that if the main controller fails, all the network information could retrieved from the learning system and that will save the cost of adding a new controller to the network.

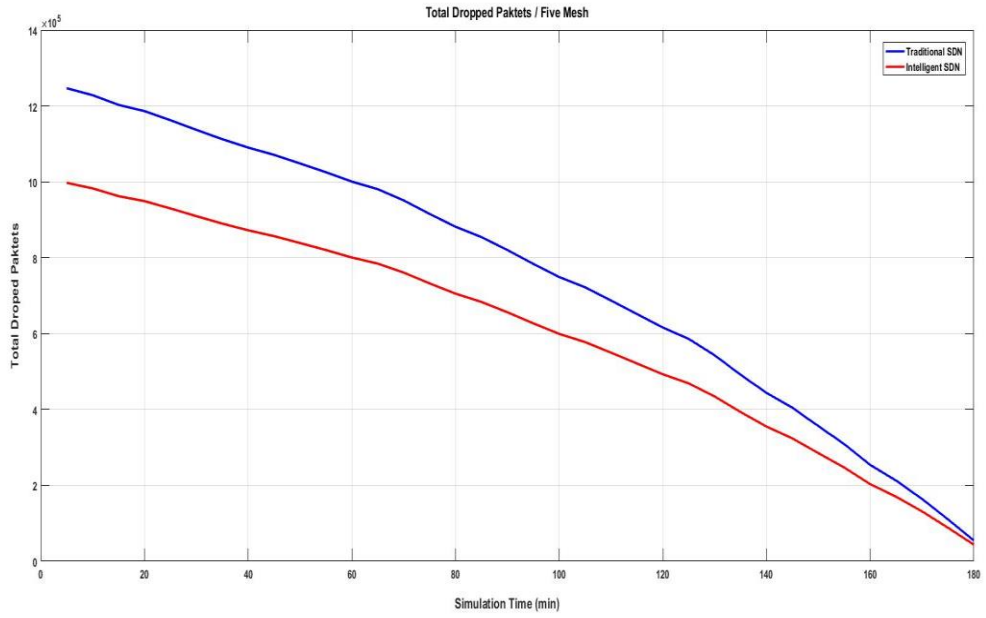

Figure 5-12 Dropped packet numbers in a five switch topology

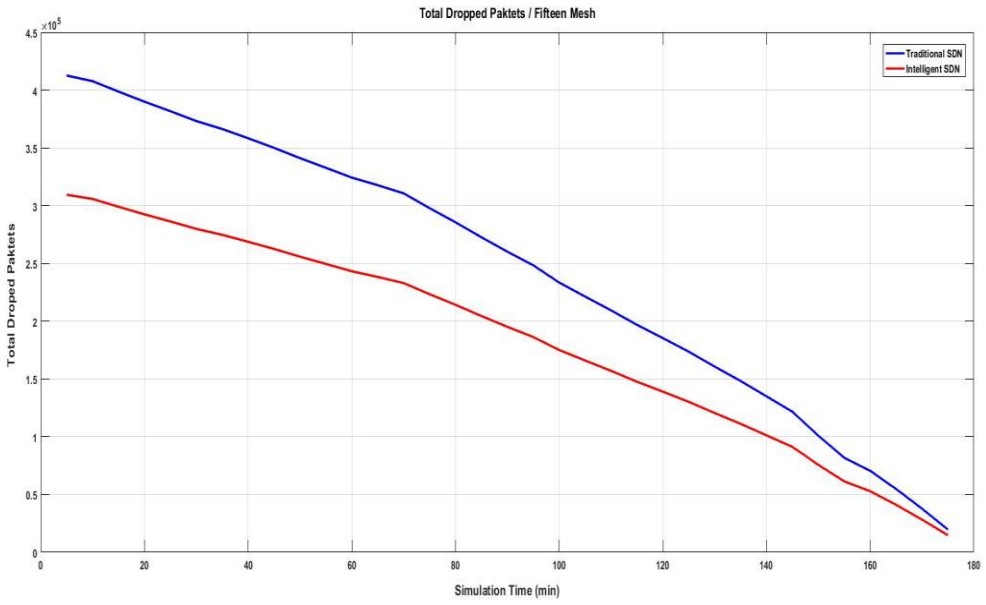

Figure 5-13 Dropped packet numbers in a 15 switch topology

B. Custom Topology Scenario

For the Mininet simulation, as shown in Figure 5-14, the custom topology was made up of nine OpenFlow switches with eleven links, thereby ensuring their connection Figure 5-15 presents the algorithm execution and recovery times indicating that, in comparison to the other algorithms, the SFSA can identify the optimal path from all possible one much faster.

Figure 5-16, illustrates the PDR comparison and it can be seen that, in contrast to the default Mininet routing algorithms, MST and OSPF, SFSA achieves a 31% decrease in this ratio. The capacity of a control plane to prevent additional traffic loss in the event of a network failure by finding a feasible alternative routing with no routing loops or black holes is known as failure recovery.

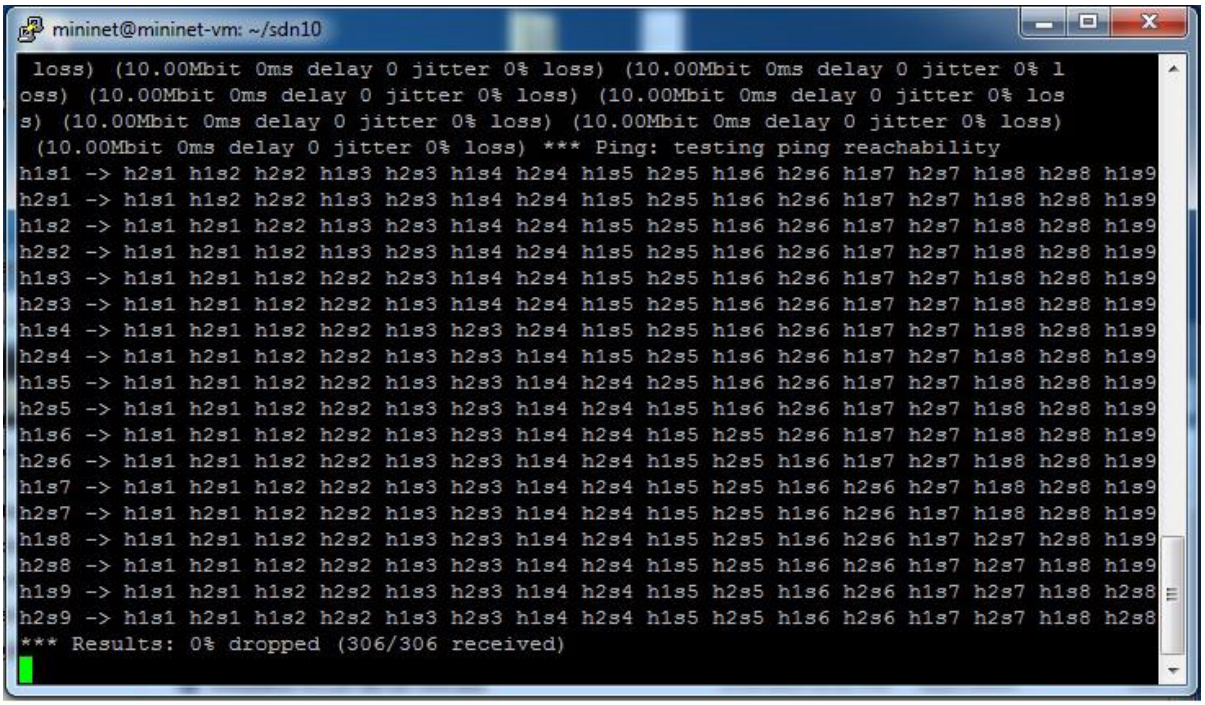

Figure 5-14 Custom topology simulation run for the Mininet

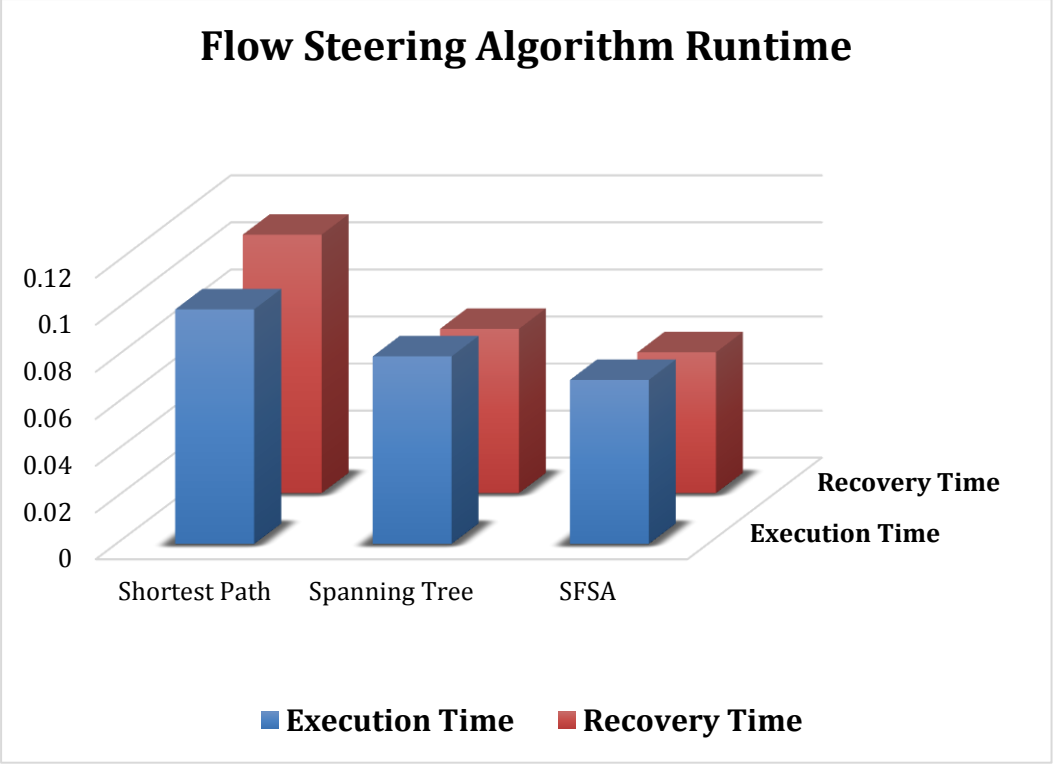

Figure 5-15 Custom topology scenario for SFSA execution and recovery time

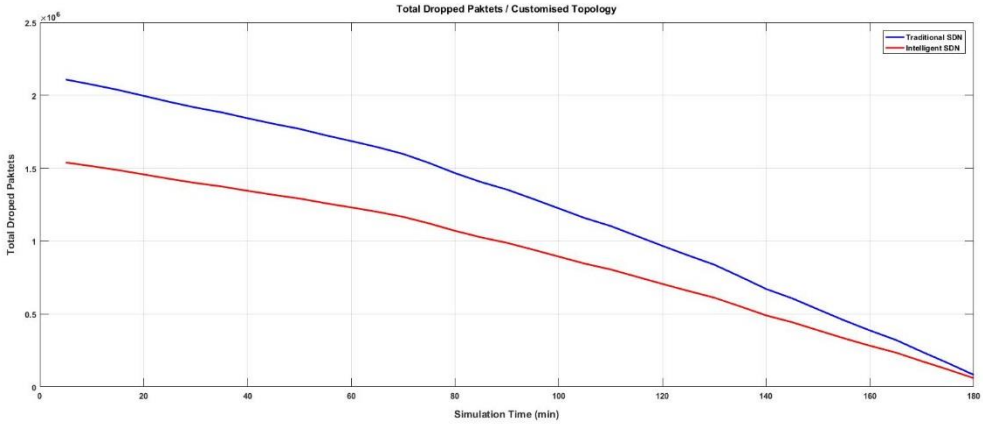

Figure 5-16 Packet Drop Ratio (PDR %) for the custom topology scenario

In the process of optimal path identification, an important factor considered by SFSA is the determination of the congestion of every network link. To achieve route optimisation, it is not the shortest path that is chosen, but that with minimal congestion and delay, this selection process being undertaken by the multi-objective routing algorithm. The values associated with the worst case scenario in custom topology are listed in Table 5-2.

It can be seen that, when selecting the optimal path between two nodes, SFSA successfully balances path length and congestion cost. As a result, the end-to-end delay is 15-45% lower than in the case of the default Mininet routing algorithm, which is influenced by the traffic load of the switches.

Table 5-2
CUSTOM TOPOLOGY WORST CASE SCENARIO

Source	Destination	Hops Count	Delay Cost	Algorithm
1	2	1	10.75	OSPF
1	2	1	10.75	SFSA
1	2	3	19.83	MST
1	4	3	22.38	OSPF
1	4	4	16.61	SFSA
1	4	4	16.61	MST

Also, the bottleneck problem has been solved by reducing the number of packets that were sent to the controller by 36%, as seen in Figure 5-17 and the throughput performance was enhanced by at least 10.50% compared with the other algorithms, as shown in Figure 5-18.

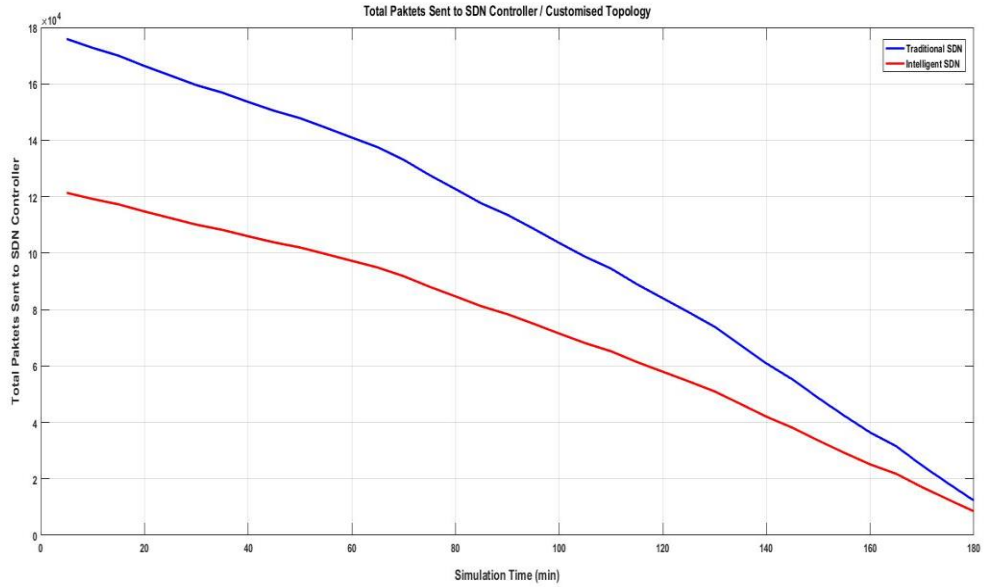

Figure 5-17 Number of messages sent to the controller from switches in a 9 switch topology

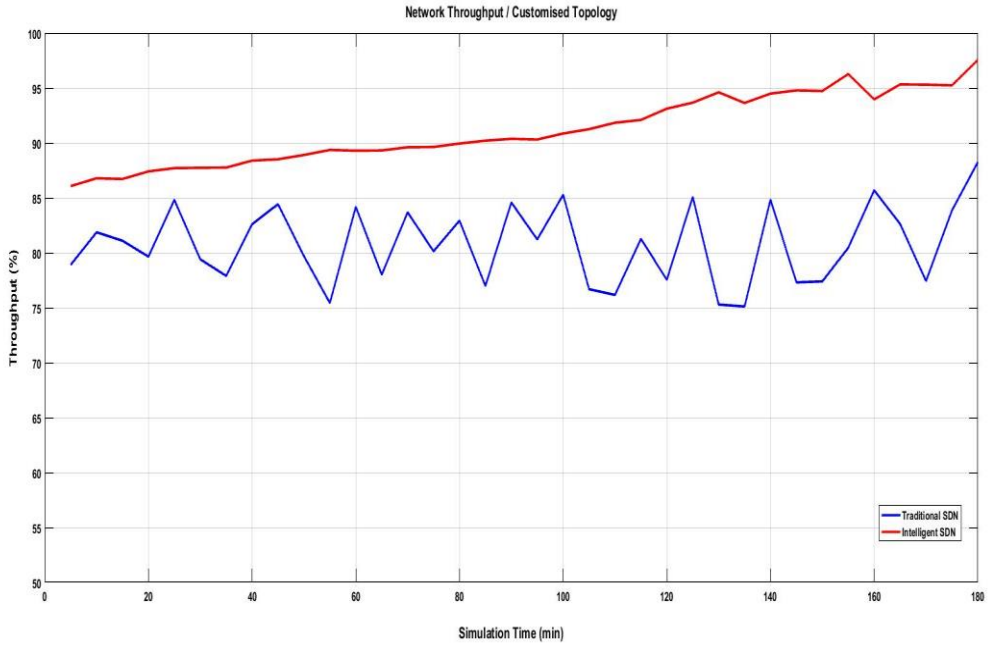

Figure 5-18 The maximum throughput for the normal and intelligent algorithms in a 9 switch topology

5.4. **Summary**

This chapter has reviewed major research related to SDN and a promising approach involving incorporation of a smart agent into the SDN controller has been proposed. SDN is of significant benefit to network and flow management, but it is not without its limitations,

which need further investigation, one major problem being the stability of the control plane. Numerous strategies have been suggested to address this problem, all of which involve measuring the performance of the control plane stability in relation to certain parameters of network QoS (i.e. throughput and latency). In contrast to these strategies, the proposed SFSA approach is geared towards improvement of end-to-end delay, PDR, number of packets sent to the controller and throughput. According to the findings of this study, as the number of requests rose at high traffic, SFSA ensures the stability of the execution and recovery time in the two scenarios employed. In addition, by transferring knowledge to an external agent, it is possible to diminish the number of flow entries in the switches. Reliability of the SDN controller has also been solved by depending on the agent program when the controller fails, whereby it can retrieve all the knowledge about the network. Finally, the results of a simulation confirm that, in comparison to the MST and OSPF algorithms, the SFSA approach performs much better. The Smart Flow Steering Agent SFSA distributes network traffic to suitable paths, in addition to supervising link and path loads. A scenario with a minimum spanning tree (MST) routing algorithm and another with the open shortest path first (OSPF) routing algorithms were employed to assess the SFSA. In comparison to these two routing algorithms, the suggested SFSA strategy produced a reduction of 22-31% in the packets dropped ratio (PDR), a reduction of up to 36% in packets sent to the controller depending on the traffic produced, as well as a reduction of 23% in round-trip time (RTT) and with throughput enhancement of approximately 10 %.

6. Chapter Six: Conclusions and Future works

6.1. Conclusions

For this thesis, different challenges regarding SDN network performance have been investigated. The main purpose has been the proposal of new intelligent agent based approaches to address these issues, with the aim of improving the performance and reliability of SDN networks.

The first approach involved using techniques that rely on an Artificial Neural Network for predicting the performance of a Software Defined Network, according to sensitive traffic parameters. Those used in this study are round-trip time, throughput and the flow table rules for each switch. A POX controller and OpenFlow switches (which characterise the behaviour of the Software Defined Network) have been modelled and simulated via Mininet and Matlab platforms. Three different topologies were used in this study, namely, mesh topology with five switches and fifteen switches along with custom topology with along switches. All these topologies were connected to one POX controller and each switch was connected to two hosts. The network program ran for 180 minutes with three different traffic loads to study the behaviour of the network, namely, when it was low, medium and high.

An ANN has been proposed to find the correlation between the most effective input factors and performance output, with the former being the round-trip time, throughput and the flow table rules for each switch as well as a POX controller and OpenFlow switches. The ANN can be used very efficiently as a predictor, especially in an SDN controller, with different traffic loads, when nature of the network is complex and highly nonlinear. The various topologies of the ANN were tested by applying one or two hidden layers with different numbers of neurons. Moreover, LMA was used as a learning algorithm in the feed-forward ANN structure. The results have shown that the network with one hidden layer gives a very acceptable MSE, with a figure of less than $2.466 \times 10{-}8$. The proposed ANN could be used efficiently to improve the performance of an SDN by selecting the best input parameters, which is to be the subject of future work.

For the second approach, the use of a new hybrid intelligent approach for optimising the performance of Software-Defined Networks (SDN) was introduced, based on heuristic optimisation methods integrated with a neural network paradigm. Evolutionary optimisation

techniques, including particle swarm optimisation (PSO) and genetic algorithms (GA), were employed to find the best set of inputs that give the maximum performance of an SDN. The neural network model was trained and applied as an approximator of SDN behaviour. An analytical investigation was conducted to distinguish the optimal optimisation approach based on SDN performance as an objective function as well as the computational time. After getting the general model of the neural network through testing it with unseen data, the model was implemented with PSO and GAs to find the best performance of the SDN. The PSO approach combined with SDN, represented by an ANN, was identified as having comparatively better configuration regarding its performance index as well as its computational efficiency.

After building the intelligent agent program, (which includes two parts: the ANN predictor and the PSO optimiser) the SDN network showed enhancement in its performance, thereby solving several of the challenges faced. Congestion is a major hindrance to performance in existing communication networks; thus, one of the most significant achievements of this thesis is reducing this, along with delays, the number of packets sent to the controller and packet loss. When the intelligent agent program runs, it depends on the previous learning of the network, so the first packet does not have to go to the controller every time. This is because the intelligent program has already sent the flow table, which includes the path for this packet. In this case, the number of packets that are send to the controller is reduced. This method prevented the bottleneck problem in the controller, thus enhancing the control performance. QoS is also affected by End to End (E2E) Delay, another characteristic of network-related performance that needs to be minimised in the context of real-time communications. Throughput can also affect SDN performance if many packets are lost during communication. Another common network-related performance consideration is packet loss, which can be assessed either per-flow or network-wide. It can be the consequence of crashes in the network (similar to congestion) which may be triggered by forwarding links and devices.

SDN architecture is ultimately reliant on one controller, which commands every network. This can be problematic, as in the event of a malfunction in the central controller, the entire network fails owing to the lack of a replacement controller.

This study has provided robust solutions based on artificial intelligence (AI) techniques, as a means to enhance the performance of data transmission in SDN. This has been achieved by eliciting a global network view of SDN. A third unit, a knowledge analysis server (cluster) has been appended to the SDN network in order to deposit the application

information, as well as for predicting and evaluating traffic patterns. A hybridised intelligence system with an ANN has been proposed as a method of demonstrating and improving the functionalities of SDN. The evolutionary algorithm (EV) optimisers deployed are particle swarm optimisation (PSO) and genetic algorithms (GAs), because they allow the network operator to identify the ideal arrangement of inputs for SDN, according to the uppermost level of performance as a fitness function. Ultimately, the outcomes show that GA is less efficient than PSO, with the latter offering more in terms of convergence, computational response, and overall performance. To map the inputs and efficiency of the system, the ANN was implemented with a single layer positioned in the hidden zone and the outcome is very satisfactory. The network produced a reasonable MSE. In addition, the applicability of the trained ANN was tested by introducing unfamiliar data. As a result, the system was synergised with the use of evolutionary algorithms and the aforementioned hybridised model was formed.

To enhance throughput and minimise latency, the Smart Flow Steering Agent SFSA distributed network traffic to suitable paths, in addition to supervising link and path loads. A scenario with a minimum spanning tree (MST) routing algorithm and another with the open shortest path first (OSPF) routing algorithms were employed to assess the SFSA. In comparison to these two routing algorithms, the suggested SFSA strategy produced a reduction of 22-31% in the packets dropped ratio (PDR), a reduction of up to 36% in packets sent to the controller depending on the traffic produced, as well as a reduction of 23% in round-trip time (RTT). A Mininet emulator and POX controller were employed to conduct the simulation. Another advantage of the SFSA over MST and OSPF is that its implementation and recovery time do not exhibit fluctuations. The smart flow steering agent will open a new horizon for deploying new smart agents in Software Defined Networks that enhance network programmability and management.

Although the Smart Flow Steering Agent (SFSA) has reached its aim, there were some unavoidable limitations such as the absence of unified simulation environments to undertake the experiment on SDN. Mininet was employed to design and obtain the dataset of SDN, while MATLAB was used to design the intelligent agent due to the ready-to-use libraries provided by the Mathworks. In addition, although using ANNs has showen to be a very powerful and interesting tool for approximation and modelling, this use cannot be retrained. If data is later added, it is intolerable to complement to an existing ANN.

6.2. **Future works**

There are a number of recommendations for future study directions for using an intelligent agent in an SDN network with the proposed ANN prediction concept and optimisation technique. The future investigations recommended are as follows:

- The first direction is to use the Application Optimisation program: This prioritises and changes the behaviour of the network based on applications behaviour by analysing the type of the application. By using this, the SDN controller can classify the traffic load depending on the application type, such as video, audio or data, which can also help to assess the amount data according to the application type. This will reduce the amount of information shared in the network and reduce the traffic load. Another AI technique could be used such as fuzzy logic, which could process packets quickly and make the selecting of next forwarding node decisions easier.
- Intelligent Cloud Controller: cloud organisation needs take into account how to utilise the main controller functions in an intelligent controller efficiently, thereby increasing network reliability. In this case, any SDN controller could connect to the intelligent controller to study the network environment on an ongoing basis so that the latter can report back to the former in real time.
- Predict Network Size program: Study and analyse the behaviour of the network environment with an intelligent program and predict the number of controllers the network needs as determined by the previous results and also decide upon the best place for the new controller.
- Intelligent Attacks Detection: Attacks detection based on an intelligent security program could be built in the controller. When the first packet arrives in the network it will be sent directly to the controller, it might be malicious, and could be cause huge harm to the whole system. In order to solve this potential problem, an intelligent attacks detection program could be used to mitigate this by detecting and denying the malicious packets. To this end, intelligent attacks detection program would study the type of packet depending on its source or payload and when the program found that the packet was abnormal it could discard it and all the data coming from that source.
- Design a Simulink model: undertake the experiments of design and implement it in a unified simulation environment for intelligent agent to improve the SDN

performance. The Simulink model can be designed to simulate different network topologies. Moreover, it will be relatively easy to create an intelligent solution to interact with such model through reading the data and proposing an optimal set of inputs that would enable the SDN of performing as ideally as possible.

References

[1] B. A. A. Nunes, M. Mendonca, X. N. Nguyen, K. Obraczka, and T. Turletti, "A survey of software-defined networking: Past, present, and future of programmable networks," *IEEE Commun. Surv. Tutorials*, vol. 16, no. 3, pp. 1617–1634, 2014.

[2] N. Feamster, J. Rexford, and E. Zegura, "The road to SDN," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 2, pp. 87–98, 2014.

[3] H.-M. Huang, T. Tidwell, C. Gill, C. Lu, X. Gao, and S. Dyke, *Cyber-physical systems for real-time hybrid structural testing*, no. 1. 2010.

[4] M. T. Hoske, *Industry 4.0 and Internet of Things tools help streamline factory automation*, vol. 62, no. 2. Springer, 2015.

[5] D. Pascual Serrano, C. Vera Pasamontes, and R. Girón Moreno, *Modelos animales de dolor neuropático*, vol. 31, no. 2. Morgan Kaufmann, 2016.

[6] F. Hu, Q. Hao, and K. Bao, "A survey on software-defined network and OpenFlow: From concept to implementation," *IEEE Commun. Surv. Tutorials*, vol. 16, no. 4, pp. 2181–2206, 2014.

[7] D. B. Rawat and S. R. Reddy, "Software Defined Networking Architecture, Security and Energy Efficiency: A Survey," *IEEE Commun. Surv. Tutorials*, vol. 19, no. 1, pp. 325–346, 2017.

[8] I. Alazzam, I. Alsmadi, and K. M. O. Nahar, "Software design principles to enhance SDN architecture," *IJACSA) Int. J. Adv. Comput. Sci. Appl.*, vol. 7, no. 8, pp. 156–163, 2016.

[9] D. B. Rawat and S. R. Reddy, "Software Defined Networking Architecture, Security and Energy Efficiency: A Survey," *IEEE Commun. Surv. Tutorials*, vol. 19, no. 1, pp. 325–346, 2017.

[10] A. Basta, A. Blenk, K. Hoffmann, H. J. Morper, M. Hoffmann, and W. Kellerer, "Towards a Cost Optimal Design for a 5G Mobile Core Network based on SDN and NFV," *IEEE Trans. Netw. Serv. Manag.*, pp. 1–1, 2017.

[11] D. Kim, J. M. Gil, G. Wang, and S. H. Kim, "Integrated SDN and non-SDN network management approaches for future internet environment," in *Lecture Notes in Electrical Engineering*, vol. 240 LNEE, Springer, 2013, pp. 529–536.

[12] I. Yaqoob, E. Ahmed, I. A. T. Hashem, A. I. A. Ahmed, A. Gani, M. Imran, and M. Guizani, “Internet of Things Architecture: Recent Advances, Taxonomy, Requirements, and Open Challenges,” *IEEE Wirel. Commun.*, vol. 24, no. 3, pp. 10–16, 2017.

[13] E. Tantar, M. R. Palattella, T. Avanesov, M. Kantor, and T. Engel, *EVOLVE - A Bridge between Probability, Set Oriented Numerics, and Evolutionary Computation V*, vol. 288. 2014.

[14] S. Ortiz, “Software-defined networking: On the verge of a breakthrough?,” *Computer (Long. Beach. Calif).*, vol. 46, no. 7, pp. 10–12, 2013.

[15] H. Kim and N. Feamster, “Improving network management with software defined networking,” *IEEE Commun. Mag.*, vol. 51, no. 2, pp. 114–119, 2013.

[16] K. Bakshi, “Considerations for Software Defined Networking (SDN): Approaches and use cases,” in *IEEE Aerospace Conference Proceedings*, 2013, pp. 1–9.

[17] S. Fang, Y. Yu, C. H. Foh, K. Mi, and M. Aung, “A Loss-Free Multipathing Solution for Data Center Network Using Software-De fi ned Networking Approach,” in *APMRC, 2012 Digest*, 2013, vol. 49, no. 6, pp. 2723–2730.

[18] S. H. Yeganeh, A. Tootoonchian, and Y. Ganjali, “On Scalability of Software Defined Networking,” *IEEE Commun. Mag.*, vol. 13, no. February, pp. 136–141, 2013.

[19] S. Das, G. Parulkar, N. McKeown, P. Singh, D. Getachew, and L. Ong, “Packet and circuit network convergence with OpenFlow,” in *Optical Fiber Communication (OFC), collocated National Fiber Optic Engineers Conference, 2010 Conference on (OFC/NFOEC)*, 2010, pp. 2–4.

[20] X. Li, P. Djukic, and H. Zhang, “Zoning for hierarchical network optimization in software defined networks,” in *Network Operations and Management Symposium (NOMS), 2014 IEEE*, 2014, pp. 1–8.

[21] M. Karakus and A. Durresi, “A survey: Control plane scalability issues and approaches in Software-Defined Networking (SDN),” *Comput. Networks*, vol. 112, pp. 279–293, 2017.

[22] B. Fraser, D. Lake, C. Systems, J. Finnegan, N. Viljoen, and S. O. E. N. Etworking, “Are We Ready for SDN ? Implementation Challenges for Software-Defined Networks,” *IEEE Commun. Mag.*, vol. 51, no. July, pp. 36–43, 2013.

[23] A. Voellmy, H. Kim, and N. Feamster, “Procera,” in *Proceedings of the first workshop on Hot topics in software defined networks - HotSDN ’12*, 2012, p. 43.

[24] Y. Jarraya, T. Madi, and M. Debbabi, “A survey and a layered taxonomy of software-defined networking,” *IEEE Commun. Surv. Tutorials*, vol. 16, no. 4, pp. 1955–1980, 2014.

[25] W. Stallings, *Foundations of Modern Networking: SDN, NFV, QoE, IoT, and Cloud*. Addison-Wesley Professional, 2015.

[26] C. E. Rothenberg, A. Vidal, M. R. Salvador, C. N. A. Corrêa, S. Lucena, F. Farias, J. Salvatti, E. Cerqueira, and A. Abelém, “Hybrid Networking Toward a Software-Defined Era,” in *Network Innovation through OpenFlow and SDN: Principles and Design*, CRC Press, 2014, pp. 153–198.

[27] A. J. Fabiano and J. Qiu, *Post-stereotactic radiosurgery brain metastases: A review*, vol. 59, no. 2. CRC Press, 2015.

[28] T. Nadeau and K. Gray, *SDN: Software Defined Networks An Authoritative Review of Network Programmability*. “ O’Reilly Media, Inc.,” 2013.

[29] F. Hu, Q. Hao, and K. Bao, “A survey on software-defined network and OpenFlow: From concept to implementation,” *IEEE Commun. Surv. Tutorials*, vol. 16, no. 4, pp. 2181–2206, 2014.

[30] R. Sherwood, J. Naous, S. Seetharaman, D. Underhill, T. Yabe, K.-K. Yap, Y. Yiakoumis, H. Zeng, G. Appenzeller, R. Johari, N. McKeown, M. Chan, G. Parulkar, A. Covington, G. Gibb, M. Flajslik, N. Handigol, T.-Y. Huang, P. Kazemian, and M. Kobayashi, “Carving research slices out of your production networks with OpenFlow,” *ACM SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 1, p. 129, 2010.

[31] D. Marschke, J. Doyle, and P. Moyer, *Software Defined Networking (SDN): Anatomy of OpenFlow Volume I, Volume 1*, vol. 1. Lulu. com, 2015.

[32] T. Kato, M. Kawakami, T. Myojin, H. Ogawa, K. Hirono, and T. Hasegawa, “Case study of applying SPLE to development of network switch products,” in *Proceedings of the 17th International Software Product Line Conference on - SPLC ’13*, 2013, p. 198.

[33] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, “OpenFlow,” *ACM SIGCOMM Comput. Commun. Rev.*, vol.

38, no. 2, p. 69, 2008.

[34] D. Parniewicz, B. Belter, E. Jacob, K. Pentikousis, R. Doriguzzi Corin, L. Ogrodowczyk, M. Rashidi Fard, J. Matias, M. Gerola, V. Fuentes, U. Toseef, and A. Zaalouk, "Design and implementation of an OpenFlow hardware abstraction layer," in *Proceedings of the 2014 ACM SIGCOMM workshop on Distributed cloud computing - DCC '14*, 2014, pp. 71–76.

[35] B. Belter, D. Parniewicz, L. Ogrodowczyk, A. Binczewski, M. Stroinski, V. Fuentes, J. Matias, M. Huarte, and E. Jacob, "Hardware abstraction layer as an SDN-enabler for non-OpenFlow network equipment," in *Proceedings - 2014 3rd European Workshop on Software-Defined Networks, EWSDN 2014*, 2014, pp. 117–118.

[36] D. Pascual Serrano, C. Vera Pasamontes, and R. Girón Moreno, "Modelos animales de dolor neuropático," 2016.

[37] G. Bianchi, M. Bonola, A. Capone, and C. Cascone, "OpenState," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 2, pp. 44–51, 2014.

[38] B. Belter, A. Binczewski, K. Dombek, A. Juszczyk, L. Ogrodowczyk, D. Parniewicz, M. Stroinski, and I. Olszewski, "Programmable abstraction of datapath: Advanced programmability of heterogeneous datapath elements through hardware abstraction," in *Proceedings - 2014 3rd European Workshop on Software-Defined Networks, EWSDN 2014*, 2014, pp. 7–12.

[39] M. Sune, V. Alvarez, T. Jungel, U. Toseef, and K. Pentikousis, "An OpenFlow implementation for network processors," in *Proceedings - 2014 3rd European Workshop on Software-Defined Networks, EWSDN 2014*, 2014, pp. 123–124.

[40] D. Pascual Serrano, C. Vera Pasamontes, and R. Girón Moreno, "Modelos animales de dolor neuropático," *Dolor*, vol. 31, no. 2, pp. 70–76, 2016.

[41] A. Lara, A. Kolasani, and B. Ramamurthy, "Network innovation using open flow: A survey," *IEEE Commun. Surv. Tutorials*, vol. 16, no. 1, pp. 493–512, 2014.

[42] O. S. Specification, "Version 1.0. 0 (Wire Protocol 0x01)," *Open Netw. Found.*, 2009.

[43] H. Tian, C. Lu, J. Yang, K. Banger, D. N. Huntzinger, C. R. Schwalm, A. M. Michalak, R. Cook, P. Ciais, D. Hayes, M. Huang, A. Ito, A. K. Jain, H. Lei, J. Mao, S. Pan, W. M. Post, S. Peng, B. Poulter, W. Ren, D. Ricciuto, K. Schaefer, X. Shi, B. Tao, W. Wang, Y. Wei, Q. Yang, B. Zhang, and N. Zeng, "Global patterns and

controls of soil organic carbon dynamics as simulated by multiple terrestrial biosphere models: Current status and future directions," *Global Biogeochem. Cycles*, vol. 29, no. 6, pp. 775–792, 2015.

[44] OpenFlow Consortium, "OpenFlow Switch Specification - Version 1.1. 0 Implemented," *Can be accessed http//openflowswitch. org/*, vol. 1, pp. 1–56, 2011.

[45] T. Ren and Y. Xu, "Analysis of the New Features of OpenFlow 1.4," *Proc. 2nd Int. Conf. Information, Electron. Comput.*, no. Icieac, pp. 73–77, 2014.

[46] Z. Shu, J. Wan, J. Lin, S. Wang, D. Li, S. Rho, and C. Yang, "Traffic Engineering in Software-Defined Networking: Measurement and Management," *IEEE Access*, vol. 4, pp. 3246–3256, 2016.

[47] I. F. Akyildiz, A. Lee, P. Wang, M. Luo, and W. Chou, "A roadmap for traffic engineering in software defined networks," *Comput. Networks*, vol. 71, pp. 1–30, 2014.

[48] M. Betts, B. Niven-Jenkins, D. Brungard, N. Sprecher, and S. Ueno, "Requirements of an MPLS Transport Profile Abstract," 2009.

[49] A. R. Curtis, W. Kim, and P. Yalagandula, "Mahout: Low-overhead datacenter traffic management using end-host-based elephant detection," in *Proceedings - IEEE INFOCOM*, 2011, pp. 1629–1637.

[50] S. R. Chowdhury, M. F. Bari, R. Ahmed, and R. Boutaba, "PayLess: A low cost network monitoring framework for Software Defined Networks," in *2014 IEEE Network Operations and Management Symposium (NOMS)*, 2014, pp. 1–9.

[51] W. Braun and M. Menth, "Software-Defined Networking Using OpenFlow: Protocols, Applications and Architectural Design Choices," *Futur. Internet*, vol. 6, no. 2, pp. 302–336, 2014.

[52] A. Gelberger, N. Yemini, and R. Giladi, "Performance analysis of Software-Defined Networking (SDN)," in *Proceedings - IEEE Computer Society's Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems, MASCOTS*, 2013, pp. 389–393.

[53] Y. Luo, P. Cascon, E. Murray, and J. Ortega, "Accelerating OpenFlow switching with network processors," in *Proceedings of the 5th ACM/IEEE Symposium on Architectures for Networking and Communications Systems - ANCS '09*, 2009, p. 70.

[54] V. Tanyingyong, M. Hidell, and P. Sjödin, “Improving PC-based OpenFlow switching performance,” in *Architectures for Networking and Communications Systems (ANCS), 2010 ACM/IEEE Symposium on*, 2010, no. 硕士论文—OpenFlow, pp. 8–9.

[55] S. Hassas Yeganeh and Y. Ganjali, “Kandoo,” in *Proceedings of the first workshop on Hot topics in software defined networks - HotSDN ’12*, 2012, p. 19.

[56] J. C. Mogul and P. Congdon, “Hey, you darned counters!,” in *Proceedings of the first workshop on Hot topics in software defined networks - HotSDN ’12*, 2012, p. 25.

[57] G. Lu, R. Miao, Y. Xiong, and C. Guo, “Using CPU as a traffic co-processing unit in commodity switches,” in *Proceedings of the first workshop on Hot topics in software defined networks - HotSDN ’12*, 2012, p. 31.

[58] L. Vanbever, J. Reich, T. Benson, N. Foster, and J. Rexford, “HotSwap,” in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking - HotSDN ’13*, 2013, p. 133.

[59] A. METZLER and A. Metzler, “Ten Things to Look for in an SDN Controller.” p. 11, 2013.

[60] V. Yazici, M. O. Sunay, and A. O. Ercan, “Controlling a Software-Defined Network via Distributed Controllers,” *arXiv Prepr. arXiv1401.7651*, vol. 90, no. 11580, p. 6, 2014.

[61] C. A. B. Macapuna, C. E. Rothenberg, and M. F. Magalh??es, “In-packet bloom filter based data center networking with distributed OpenFlow controllers,” in *2010 IEEE Globecom Workshops, GC’10*, 2010, vol. 2, pp. 584–588.

[62] R. Raghavendra, J. Lobo, and K.-W. Lee, “Dynamic graph query primitives for SDN-based cloudnetwork management,” in *Proceedings of the first workshop on Hot topics in software defined networks - HotSDN ’12*, 2012, p. 97.

[63] M. Jammal, T. Singh, A. Shami, Y. Li, and B. Telecom, “Software-Defined Networking: State of the Art and Research Challenges,” pp. 1–24.

[64] A. Tavakoli, M. Casado, and S. Shenker, “Applying NOX to the Datacenter,” in *Proceedings of the 8th ACM Workshop on Hot Topics in Networks (HotNets-VIII)*, 2009, pp. 1--6.

[65] B. Heller, R. Sherwood, and N. McKeown, “The controller placement problem,” in

Proceedings of the first workshop on Hot topics in software defined networks - HotSDN '12, 2012, p. 7.

[66] B. Boughzala, R. Ben Ali, M. Lemay, Y. Lemieux, and O. Cherkaoui, "OpenFlow supporting inter-domain virtual machine migration," in *8th IEEE and IFIP International Conference on Wireless and Optical Communications Networks, WOCN2011*, 2011, pp. 1–7.

[67] V. Mann, A. Vishnoi, K. Kannan, and S. Kalyanaraman, "CrossRoads Seamless VM Mobility Across Data Centers through Software Defined Networking.pdf," in *Network Operations and Management Symposium (NOMS), 2012 IEEE*, 2012, pp. 88–96.

[68] M. Yu, J. Rexford, M. J. Freedman, and J. Wang, "Scalable flow-based networking with DIFANE," *Proc. ACM SIGCOMM 2010 Conf. SIGCOMM - SIGCOMM '10*, vol. 40, no. 4, p. 351, 2010.

[69] H. Kim, M. Schlansker, and J. Santos, "CORONET: Fault tolerance for software defined networks - Annotated," in *2012 20th IEEE*, 2012, pp. 1–2.

[70] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "DevoFlow," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 4, p. 254, 2011.

[71] J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, A. R. Curtis, and S. Banerjee, "DevoFlow," in *Proceedings of the Ninth ACM SIGCOMM Workshop on Hot Topics in Networks - Hotnets '10*, 2010, pp. 1–6.

[72] M. Abu Sharkh, A. Ouda, and A. Shami, "A resource scheduling model for cloud computing data centers," *2013 9th Int. Wirel. Commun. Mob. Comput. Conf.*, pp. 213–218, 2013.

[73] M. Kobayashi, S. Seetharaman, G. Parulkar, P. Weissmann, and N. Mckeown, "Maturing of OpenFlow and Software Defined Networking through Deployments," *Comput. Networks*, vol. 61, pp. 151–175, 2014.

[74] B. R. Al-Kaseem, H. S. Al-Raweshidy, Y. Al-Dunainawi, and K. Banitsas, "A New Intelligent Approach for Optimising 6LoWPAN MAC Layer Parameters," *IEEE Access*, pp. 1–1, 2017.

[75] M. Kuzniar, P. Peresini, M. Canini, D. Venzano, and D. Kostic, "A SOFT way for openflow switch interoperability testing," *Proc. 8th Int. Conf. Emerg. Netw. Exp. Technol.*, pp. 265–276, 2012.

[76] M. Latah and L. Toker, “Application of artificial intelligence to software defined networking: A survey,” *Indian J. Sci. Technol.*, vol. 9, no. 44, 2016.

[77] Chaturvedi, *Soft computing -- techniques and its applications in electrical engineering*, vol. 103. Springer, 2008.

[78] Y. Yilan Liu, Y. Yun Pan, M. Muxi Yang, W. Wenqing Wang, C. Chi Fang, and R. Ruijuan Jiang, “The multi-path routing problem in the Software Defined Network,” in *2015 11th International Conference on Natural Computation (ICNC)*, 2015, pp. 250–254.

[79] C. Gao, H. Wang, F. Zhu, L. Zhai, and S. Yi, “A Particle Swarm Optimization Algorithm for Controller Placement Problem in Software Defined Network,” Springer International Publishing, 2015, pp. 44–54.

[80] A. C. Risdianto and E. Mulyana, “Implementation and analysis of control and forwarding plane for SDN,” in *2012 7th International Conference on Telecommunication Systems, Services, and Applications, TSSA 2012*, 2012, pp. 227–231.

[81] I. Mihai-Gabriel and P. Victor-Valeriu, “Achieving DDoS resiliency in a software defined network by intelligent risk assessment based on neural networks and danger theory,” *CINTI 2014 - 15th IEEE Int. Symp. Comput. Intell. Informatics, Proc.*, pp. 319–324, 2014.

[82] S. ZHANG and F. ZOU, “Survey on software defined network research,” *Appl. Res. Comput.*, vol. 30, pp. 2246–2251, 2013.

[83] A. Sgambelluri, A. Giorgetti, F. Cugini, F. Paolucci, and P. Castoldi, “OpenFlow-Based Segment Protection in Ethernet Networks,” *J. Opt. Commun. Netw.*, vol. 5, no. 9, p. 1066, Sep. 2013.

[84] O. Dobrijevic, M. Santl, and M. Matijasevic, “Ant Colony Optimization for QoE-Centric Flow Routing in Software-Defined Networks,” in *Network and Service Management (CNSM), 2015 11th International Conference on*, 2015, pp. 274–278.

[85] X. Yao, H. Wang, C. Gao, and S. Yi, “Maximizing Network Utilization for SDN Based on Particle Swarm Optimization,” in *Tools with Artificial Intelligence (ICTAI), 2016 IEEE 28th International Conference on*, 2016, pp. 921–925.

[86] S. Kim, “Cognitive Model-Based Autonomic Fault Management in SDN,”

Dpnm.Postech.Ac.Kr, p. 84, 2013.

[87] M. Malboubi, L. Wang, and C. Chuah, "Intelligent SDN based Traffic (de) Aggregation and Measurement Paradigm (i STAMP)," in *IEEE INFOCOM 2014 - IEEE Conference on Computer Communications*, 2014, pp. 934–942.

[88] Q. Wu, J. Huang, and O. Yang, "An Open Flow Controller Based on the Intercerebral Neural Network for Media Independent Handover," *Int J Swarm Intel Evol Comput*, 2014.

[89] Y. Lü, *Industrial intelligent control : fundamentals and applications*. John Wiley & Sons, 1996.

[90] D. E. Rumelhart, J. L. McClelland, and P. Group, "others, Parallel distributed processing, vol. 1." IEEE, 1988.

[91] I. This, "Control System Design," *New Jersey*, pp. 39–64, 2002.

[92] W. He, Y. Chen, and Z. Yin, "Adaptive Neural Network Control of an Uncertain Robot with Full-State Constraints," *IEEE Trans. Cybern.*, vol. 46, no. 3, pp. 620–629, 2016.

[93] H. M, "Review of the applications of neural networks in chemical process control -- {Simulation} and on-line implementation," - *Artifi. Intelli. Eng.*, vol. 13, no. 1, pp. 55–68, 1999.

[94] A. Afram, F. Janabi-Sharifi, A. S. Fung, and K. Raahemifar, "Artificial neural network (ANN) based model predictive control (MPC) and optimization of HVAC systems: A state of the art review and case study of a residential HVAC system," *Energy Build.*, vol. 141, no. February, pp. 96–113, 2017.

[95] T. Konjić, A. Jahić, and J. Pihler, "Artificial Neural Network Approach to Photovoltaic System Power Output Forecasting," 2015.

[96] S. S. Haykin, "Kalman Filtering and Neural Networks," p. 304, 2004.

[97] D. W. Patterson, *Artificial neural networks: theory and applications*. Prentice Hall PTR, 1998.

[98] J. H. Holland, "Adaptation in Natural and Artificial Systems : An Introductory Analysis with Application to Biology, Control and Artificial Intelligence," *Ann Arbor, MI Univ. Michigan Press*, 1992.

[99] L. J. Fogel and L. J., *Intelligence through simulated evolution : forty years of evolutionary programming*. John Wiley & Sons, Inc., 1999.

[100] F. Keti and S. Askar, "Emulation of Software Defined Networks Using Mininet in Different Simulation Environments," *Proc. - Int. Conf. Intell. Syst. Model. Simulation, ISMS*, vol. 2015–Octob, pp. 205–210, 2015.

[101] S. S. Kolahi, S. Narayan, D. D. T. Nguyen, and Y. Sunarto, "Performance monitoring of various network traffic generators," *Proc. - 2011 UKSim 13th Int. Conf. Model. Simulation, UKSim 2011*, pp. 501–506, 2011.

[102] A. V Akella and K. Xiong, "Quality of Service (QoS)-Guaranteed Network Resource Allocation via Software Defined Networking (SDN)," *2014 IEEE 12th Int. Conf. Dependable, Auton. Secur. Comput.*, pp. 7–13, 2014.

[103] A. Ishimori, F. Farias, E. Cerqueira, and A. Abelem, "Control of multiple packet schedulers for improving QoS on OpenFlow/SDN networking," *Proc. - 2013 2nd Eur. Work. Softw. Defin. Networks, EWSDN 2013*, pp. 81–86, 2013.

[104] M. Gupta, J. Sommers, and P. Barford, "Fast, accurate simulation for SDN prototyping," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking - HotSDN '13*, 2013, p. 31.

[105] R. T. Marler and J. S. Arora, "Survey of multi-objective optimization methods for engineering," *Struct. Multidiscip. Optim.*, vol. 26, no. 6, pp. 369–395, 2004.

[106] T. C. Burg, *Intelligent Systems: Modeling, Optimization, and Control*. 2008.

[107] J. R. Sampson, *Adaptation in Natural and Artificial Systems (John H. Holland)*, vol. 18, no. 3. 1976.

[108] L. J. Fogel and L. J., *Intelligence through simulated evolution : forty years of evolutionary programming*. Wiley, 1999.

[109] M. Maarouf, A. Sosa, B. Galván, D. Greiner, G. Winter, M. Mendez, and R. Aguasca, "Advances in Evolutionary and Deterministic Methods for Design, Optimization and Control in Engineering and Sciences," in *Advances in Evolutionary and Deterministic Methods forDesign, Optimization and Control in Engineering and Sciences, ComputationalMethods in Applied Sciences 36*, vol. 36, Springer International Publishing Switzerland 2015, 2015, pp. 209–223.

[110] Z. Michalewicz, “Genetic Algorithms + Data Structures = Evolution Programs,” *Springer Berlin Heidelberg New York*, vol. 1, no. 3. p. 387, 1996.

[111] J. Wen, H. Ma, and X. Zhang, “Optimization of the occlusion strategy in visual tracking,” *Tsinghua Sci. Technol.*, vol. 21, no. 2, pp. 221–230, 2016.

[112] M. Dorigo and L. M. Gambardella, “Ant colony system: A cooperative learning approach to the traveling salesman problem,” *IEEE Trans. Evol. Comput.*, vol. 1, no. 1, pp. 53–66, 1997.

[113] B. Sun, L. Li, and H. Chen, “Shortest Path Routing Optimization Algorithms Based on Genetic Algorithms,” *Comput. Eng.*, 2005.

[114] Y. Al-Dunainawi and M. F. Abbod, “Evolutionary based optimisation of multivariable fuzzy control system of a binary distillation column,” in *Proceedings - 2016 UKSim-AMSS 18th International Conference on Computer Modelling and Simulation, UKSim 2016*, 2016, pp. 127–132.

[115] H. Tamaki, H. Kita, and S. Kobayashi, “Multi-objective optimization by genetic algorithms: a review,” *Proc. IEEE Int. Conf. Evol. Comput.*, pp. 517–522, 1996.

[116] M. Srinivas and L. M. Patnaik, “Genetic Algorithms: A Survey,” *Computer (Long. Beach. Calif).*, vol. 27, no. 6, pp. 17–26, 1994.

[117] R. Eberhart and J. Kennedy, “A new optimizer using particle swarm theory,” *MHS’95. Proc. Sixth Int. Symp. Micro Mach. Hum. Sci.*, pp. 39–43, 1995.

[118] J. Kennedy and R. C. Eberhart, *Swarm Intelligence*. 2001.

[119] V. Kothari, J. Anuradha, S. Shah, and P. Mittal, “A Survey on Particle Swarm Optimization in Feature Selection,” in *Global Trends in Information Systems and Software Applications*, Springer, Berlin, Heidelberg, 2012, pp. 192–201.

[120] Y.-M. Jau, K.-L. Su, C.-J. Wu, and J.-T. Jeng, “Modified quantum-behaved particle swarm optimization for parameters estimation of generalized nonlinear multi-regressions model based on Choquet integral with outliers,” *Appl. Math. Comput.*, vol. 221, pp. 282–295, 2013.

[121] J. Kennedy, “Bare bones particle swarms,” in *2003 IEEE Swarm Intelligence Symposium, SIS 2003 - Proceedings*, 2013, pp. 80–87.

[122] L. Y. Chuang, S. W. Tsai, and C. H. Yang, “Chaotic catfish particle swarm

optimization for solving global numerical optimization problems," *Appl. Math. Comput.*, vol. 217, no. 16, pp. 6900–6916, 2011.

[123] Y. Zhang, S. Wang, and G. Ji, "A Comprehensive Survey on Particle Swarm Optimization Algorithm and Its Applications," *Math. Probl. Eng.*, vol. 2015, pp. 1–38, 2015.

[124] B. Fortz, "Optimizing OSPF / IS-IS Weights in a Changing World 1 Introduction," *{IEEE} J. Sel. Areas Commun.*, vol. 20, no. 4, pp. 756–767, 2002.

[125] S. Y. Wang, C. C. Wu, and C. L. Chou, "Constructing an optimal spanning tree over a hybrid network with SDN and legacy switches," in *Proceedings - IEEE Symposium on Computers and Communications*, 2016, vol. 2016–Febru, pp. 502–507.

[126] S. A. Astaneh and S. Shah Heydari, "Optimization of SDN Flow Operations in Multi-Failure Restoration Scenarios," *IEEE Trans. Netw. Serv. Manag.*, vol. 13, no. 3, pp. 421–432, 2016.

[127] T. Cinkler, I. Moldován, A. Kern, C. Lukovszki, and G. Sallai, "Optimizing QoS aware ethernet spanning trees," in *2005 1st International Conference on Multimedia Services Access Networks, MSAN05*, 2005, vol. 2005, pp. 30–34.

[128] M. Caria, A. Jukan, and M. Hoffmann, "SDN Partitioning: A Centralized Control Plane for Distributed Routing Protocols," *IEEE Trans. Netw. Serv. Manag.*, vol. 13, no. 3, pp. 381–393, 2016.

[129] Y. Jararweh, M. Al-Ayyoub, A. Darabseh, E. Benkhelifa, M. Vouk, and A. Rindos, "SDIoT: a software defined based internet of things framework," *J. Ambient Intell. Humaniz. Comput.*, vol. 6, no. 4, pp. 453–461, 2015.

[130] Y. Nakahodo, T. Naito, and E. Oki, "Implementation of smart-OSPF in hybrid software-defined network," in *Proceedings of 2014 4th IEEE International Conference on Network Infrastructure and Digital Content, IEEE IC-NIDC 2014*, 2014, pp. 374–378.

[131] S. Franklin and A. Graesser, "Is It an agent, or just a program?: A taxonomy for autonomous agents," in *International Workshop on Agent Theories, Architectures, and Languages*, 1997, pp. 21–35.

[132] S. Aksoy and R. M. Haralick, "Feature normalization and likelihood-based similarity measures for image retrieval," *Pattern Recognit. Lett.*, vol. 22, no. 5, pp. 563–582,

2001.