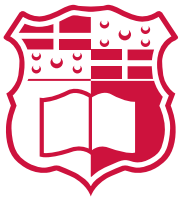
Evolutionary Algorithms for Globally Optimised Multipath Routing

Noel Farrugia



L-Università ta' Malta

Department of Communications and Computer Engineering

University of Malta

January 2020

Supervised by

Prof Ing Victor Buttigieg and Prof Johann A. Briffa

Submitted in partial fulfilment of the requirements for the degree of Doctor of Philosophy

Acknowledgements

This paragraph has been the easiest one I had to write, by far. I was lucky enough to be surrounded by amazing human beings that have made this 4 year journey not only bearable, but fun. I would like to thank my two supervisors; Prof Ing Victor Buttigieg and Prof Johann A. Briffa for their immense support and motivation throughout the entire duration of this PhD. Their guidance, advice and discussions we shared over the years, have changed the way I do research and the way I look at life in general. For this, I will be forever grateful. I must express my heartfelt gratitude to my life partner Marija Magro and two great people I have the honour to call them my friends, Karl Casha and Jasmine Spiteri. I am not going to lie, doing a PhD has been much harder than I thought; however, the knowledge and comfort of having a close knit group of friends that I can truly rely on, made the road much easier than it would have been otherwise. I would also like to thank my sister for proof reading this thesis while she was busy doing her own. Finally, I would like to thank my parents for their financial support, without which, I would not be where I am today.

The research work disclosed in this publication is partially funded by the ENDEAVOUR Scholarships Scheme (Group B).

This research has been carried out using computational facilities procured through the European Regional Development Fund, Project ERDF-076 'Refurbishing the Signal Processing Laboratory within the Department of CCE', University of Malta.

L-Università ta' Malta

FACULTY/~~INSTITUTE/CENTRE/SCHOOL~~ of ICT

DECLARATION OF AUTHENTICITY FOR DOCTORAL STUDENTS

Student's Code 23592M

Student's Name & Surname Noel Farrugia

Course Doctor of Philosophy

Title of Dissertation/Thesis

Evolutionary Algorithms for Globally Optimised Multipath Routing

(a) Authenticity of Thesis/Dissertation

I hereby declare that I am the legitimate author of this Thesis/Dissertation and that it is my original work. No portion of this work has been submitted in support of an application for another degree or qualification of this or any other university or institution of higher education.
I hold the University of Malta harmless against any third party claims with regard to copyright violation, breach of confidentiality, defamation and any other third party right infringement.

(b) Research Code of Practice and Ethics Review Procedure

I declare that I have abided by the University's Code of Practice and Research Ethics Review Procedures.

☒ As a Ph.D. student, as per Regulation 49 of the Doctor of Philosophy Degree Regulations, I accept that my thesis be made publicly available on the University of Malta Institutional Repository.

☐ As a Doctor of Sacred Theology student, as per Regulation 17 of the Doctor of Sacred Theology Regulations, I accept that my thesis be made publicly available on the University of Malta Institutional Repository.

☐ As a Doctor of Music student, as per Regulation 26 of the Doctor of Music Degree Course Regulations, I accept that my dissertation be made publicly available on the University of Malta Institutional Repository.

☐ As a Professional Doctorate student, as per Regulation 55 of the Professional Doctorate Degree Course Regulations, I accept that my dissertation be made publicly available on the University of Malta Institutional Repository.

Signature of Student	NOEL FARRUGIA Name in Full (in Caps)	30/01/2020 Date

12th December, 2019

Abstract

With the ever increasing rise of traffic generated on the Internet, the efficiency with which a network operates has become of great importance. The use of a distributed network architecture and single path routing algorithms limits the level of efficiency a network is able to sustain. To tackle this problem, a set of novel, globally optimal, multipath capable routing algorithms are proposed. The routing algorithms are designed to increase the total network flow routed over a given network, while giving preference to lower delay paths. Two routing algorithm frameworks are proposed in this work; one using Linear Programming (LP) and the other using a Multi-Objective Evolutionary Algorithm (MOEA). Compared to Evolutionary Algorithms (EAs), which are inherently sub-optimal, the LP routing algorithm is guaranteed to find a solution with the maximum load a network is able to handle without exceeding the link's capacity. However, LP solvers are unable to concurrently optimise for more than one objective. On the other hand, EAs are able to handle multiple, possibly non-linear objectives, and generate multiple viable solutions from a single run. Even though EAs are inherently sub-optimal, the EAs designed here manage to satisfy, on average, 98% of the demand found by the optimal LP generated solution.

All routing algorithms designed in this work make use of Per-Packet multipath because of its increased flexibility when compared to its Per-Flow multipath counterpart. It is well known that connection oriented protocols, such as TCP, suffer from severe performance degradation when used in conjunction with a Per-Packet multipath routing solution. This problem is solved by adding a custom scheduler to the Multipath TCP (MPTCP) protocol. Using the modified MPTCP protocol, TCP flows are able to reach a satisfaction rate of 100%, with very high probability even when that flow is transmitted over multiple paths. The combination of the modified MPTCP protocol and the designed routing algorithm(s) led to a network that is able to handle more load without sacrificing delay, when compared to OSPF under all the conditions tested in this work using network simulations.

Table Of Contents

List of Figures

List of Tables

List of Acronyms

AAA Alienated Ant Algorithm.

ACO Ant Colony Optimisation.

CA Considering Acknowledgements.

CBR Constant Bit Rate.

DEAP Distributed Evolutionary Algorithms in Python.

EA Evolutionary Algorithm.

ECMP Equal Cost Multipath Routing.

ED Edge Disjoint.

GLPK GNU Linear Programming Kit.

GMPLS Generalised MPLS.

GOMR Globally Optimal Multipath Routing.

GPU Graphical Processing Unit.

HD High Definition.

ILP Integer Linear Programming.

IP Internet Protocol.

ISP Internet Service Provider.

KSEDP k-Shortest Edge Disjoint Path.

KSP k-Shortest Path.

KSREDP k-Shortest Relaxed Edge Disjoint Path.

LP Linear Programming.

MCFP Multicommodity Flow Problem.

MIP Mixed-Integer Program.

MMAS Max-Min Ant System.

MMFMC Multicommodity Maximum-Flow Minimum-Cost.

MOEA Multi-Objective Evolutionary Algorithm.

MPLS Multi-Protocol Label Switching.

MPTCP Multi-Path TCP.

NCA Not Considering Acknowledgements.

NSGA Nondominated Sorting Genetic Algorithm.

OSPF Open Shortest Path First.

PC-MFMC Path Constrained Maximum-Flow Minimum-Cost.

PPFS Per-Packet Flow Splitting.

PSNR Peak Signal-to-Noise Ratio.

QoS Quality of Service.

RTT Round Trip Time.

SDN Software Defined Network.

SOEA Single-Objective Evolutionary Algorithm.

TCP Transmission Control Protocol.

TE Traffic Engineering.

TS Tabu Search.

UDP User Datagram Protocol.

UHD Ultra High Definition.

VM Virtual Machine.

Glossary of Symbols

C The chromosome.

$d_i \in V$ Flow f_i's destination node.

δ_i Flow f_i's requested data rate.

E The set of links.

e The link $e = (u, v) \in E$, where $u \in V$ and $v \in V$ are the links source and destination nodes, respectively.

ϵ The number of links in a given graph $\mathcal{G}$, i.e. $|E|$.

$\bar{e}$ The reverse link of e, $\bar{e} = (v, u)$.

γ_e The cost of link $e \in E$.

λ_e The capacity of link $e \in E$.

F The set of flows.

$f_i \in F$ The i^{th} flow in set F.

$\mathcal{G}$ Loop-free directed graph representing the network topology.

G_i Sequence of genes related to flow f_i.

$g_{i,j} \in \mathbb{R}_{\geq 0}$ The data rate flow f_i transmits on path $p_{i,j}$, where $\mathbb{R}_{\geq 0}$ represents all the positive real numbers, including zero.

$\alpha(g_{i,j})$ The TCP ACK data rate for $g_{i,j}$.

k_i The number of paths flow f_i is allowed to transmit on, where $k_i \leq k$.

k The maximum number of paths a flow is allowed to take.

μ The fraction of flows modified by a single mutation operation.

$\mathbf{P} = \{P_1, P_2, \ldots, P_n\}$ The set of all paths used by all flows.

$\mathcal{P}$ The population size of the evolutionary algorithm.

P_i The set of paths related to flow f_i.

$\tilde{P}_i \subseteq P_i$ The set of paths flow f_i is allowed to use during mutation.

$p_{i,j} \in P_i$ The j^{th} path for flow f_i.

$\phi(p_{i,j})$ The cost of path $p_{i,j} \in P_i$.

$r^i_{u,v}$ Data rate transmitted by flow f_i over link $(u, v) \in E$.

$s_i \in V$ Flow f_i's source node.

$\mathcal{T}$ The total network flow allocated by the *Maximum Flow* portion of the PC-MFMC problem.

θ Total number of paths given by $\theta = \sum_{i=1}^{n} |P_i|$.

$\mathcal{U}(a, b)$ A uniform random number generator, generating real values between $[a, b)$.

V The set of nodes.

ω The evolutionary algorithm's crossover probability.

ψ The evolutionary algorithm's mutation probability.

χ The number of generations the evolutionary algorithm runs for.

$z \in \mathcal{U}(0, 1)$ The random number generated from a uniform distribution between 0 and 1.

Chapter 1

Introduction

Our daily lives have become dependent on the Internet, putting ever increasing pressure on network operators to keep up with the demand. From the time the first computer networks were being designed and tested in the late 1960s, the reliance and expandability of the largest network on Earth grew rapidly with each passing year. This rapid and unprecedented growth has seen yearly increase in the traffic transported over the Internet, with researchers and the industry focusing on finding methods to increase the network's capacity while keeping costs at bay. Nowadays, the majority (80%) of the network traffic is video data [1]. Video takes such a large proportion of the traffic generated due to the ease by which an end user can consume as well as generate video data. The transmission of video streams, especially live transmissions, presents network architects with two main, but conflicting challenges; how to reduce the size of the transmitted video while retaining or even improving the video quality over previous versions. The arrival of the Internet also broke the bounds defined by country borders as users are now able to consume data that has been generated from every corner of the globe with a single click. Additionally, the concept of the *cloud* has become the norm and users have come to expect the ability to access their data and services from anywhere at any time. To hold such promises, *cloud* providers require two main components. First, reliable data centers with enough capacity to store whatever the users upload to the cloud. Second, data centers installed in strategic locations throughout the globe to make accessing the data as efficient and responsive as possible. Such environments create two types of networks; inter data center networks and the network connecting the data centers together. As the name implies, the inter data center network refers to the network that connects machines located on the same premises together. Inter data center networks are usually owned by a single company; thereby, creating an environment where the network topology can be

designed specifically for data center networks with the aim of increasing network efficiency and capacity. On the contrary, the network used to connect the data centers together usually travels through different countries and continents with the topology being dictated by geography and the financial cost incurred to build and maintain such a network. Due to the imposed restrictions on the network topology design and available link capacity, the latter network's efficiency is highly dependent on the routing algorithm's ability to use all of the available network resources. The design of such a routing algorithm is the main contribution of this thesis.

One of the major inefficiencies in current networks is caused by the use of single path routing algorithms. Open Shortest Path First (OSPF), one of the most commonly used routing algorithm, falls into this category and transmits all the packets from a source-destination pair over the shortest path. The definition of what represents the shortest path varies from network to network and is based on what the network operator seeks to minimise. Hop-count, path delay, financial cost to use a given link or a combination of multiple link properties are examples of such path length metrics. The use of single path routing algorithms, such as OSPF, are sub-optimal in terms of network efficiency due to their inability to make use of all of the available network resources. Using all of the available network resources requires a multipath-capable routing algorithm [2, 3]. Multipath-capable routing algorithms are able to transmit a flow over numerous paths and are categorised into two classes: *Per-Flow* and *Per-Packet*. In this work we define a flow to be the sequence of packets that are travelling between the same source and destination node pair (same Internet Protocol (IP) address pairs), where the packets are transmitted/received by a single application (same port number pairs). Multiple flows can exist between the same source/destination pair. A *Per-Flow* multipath routing algorithm allows flows to be routed over different paths; however, all packets related to the same flow must travel on the same path. On the other hand, a *Per-Packet* multipath routing algorithm allows packets originating from the same flow to take different paths. A *Per-Packet* algorithm is able to divide the network resources down to the packet level; whereas a *Per-Flow* algorithm is only capable of going down at the flow level. For this reason, a *Per-Packet* algorithm will always outperform or match the performance of a *Per-Flow* algorithm in terms of network resource usage. To better explain this, take the simple network topology shown in Figure 1.1 as an example, where all the links connecting the switches are assumed to have a capacity x and the links connecting the terminals to the switches have a capacity $\geq 2x$. Using this setup, one can easily see that using *Per-Flow* multipath a flow can either take the path (A $\rightarrow$ B $\rightarrow$ D) or (A $\rightarrow$ C $\rightarrow$ D), but not both. This means that the flow can use at most

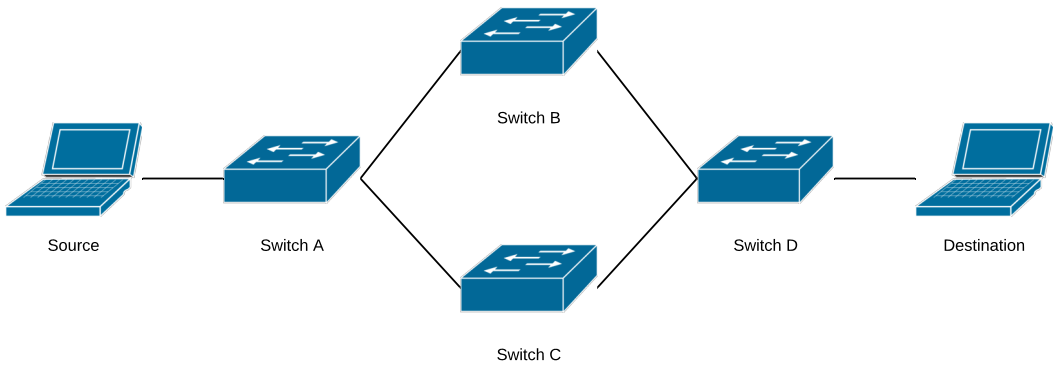


Figure 1.1: Diamond Network used to demonstrate the performance difference between Per-Flow vs Per-Packet multipath.

x capacity units. On the other hand, *Per-Packet* multipath allows the flow to split the packets over the two paths; thereby, giving the flow access to twice the capacity units ($2x$) of a *Per-Flow* multipath solution. Even though the potential benefits offered by *Per-Packet* multipath are clear, to the best of the author's knowledge, only *Per-Flow* multipath is used in practice. The main reason behind this lies in the negative effects a *Per-Packet* multipath solution exhibits when used over connection oriented protocols, such as Transmission Control Protocol (TCP). One of the most commonly used *Per-Flow* multipath solutions is Equal Cost Multipath Routing (ECMP) [4]. ECMP forwards packets related to the same flow over the same path (identical to OSPF); however, different flows are allowed to travel over different paths that have the same cost as the shortest path. ECMP determines which path a flow takes based on a hash value calculated using fields from the packet header. Although ECMP may offer better performance than OSPF [5], it is still limited as it is unable to load balance between links of different cost. In addition, as load balancing is based on the random hash values of a flow's packet headers, flows may still experience congestion if ECMP assigns two heavy flows over the same path. From this summary it is clear that multipath routing is a key component to reach maximal network efficiency. However, careful consideration of the setup where multipath will be deployed needs to be taken into consideration as Liu et al. [6] states that deploying multipath solutions in dense network topologies or lightly loaded networks might have a detrimental effect on the network's performance.

As stated earlier, a multipath-capable routing algorithm is required to make use of all of the available network resources [2, 3]. However, in addition to being multipath capable, a routing algorithm must also be globally optimal in order to reach maximum efficiency [7]. A Globally Optimal Multipath Routing (GOMR) algorithm avoids congestion by ensuring that no link is used beyond capacity for a given flow

set, by taking into account all the flows that are currently using the network. To do this, a GOMR takes into account all the flows that are currently using the network, which is what makes the algorithm globally optimal. The problem of routing commodities (in our case flows) on a network without exceeding link capacity is very similar to the well known Multicommodity Flow Problem (MCFP) [8], which falls under the umbrella of Traffic Engineering (TE). TE relates to any optimisation methods used to improve the performance and/or efficiency of a computer network. Solving the MCFP requires up to date information on both the flows that are currently using the network, and network topology information. The current distributed network architecture is unable to provide such information in an efficient and timely manner. A centralised network architecture is required to allow the deployment of such a routing solution. The Software Defined Network (SDN) [9] architecture moves the distributed control plane to a logically centralised server, referred to as the network controller. This control plane centralisation gives the network controller access to up to date information on both the flows and topology of the network; thereby, allowing the deployment of a routing algorithm that is based on the MCFP.

Linear Programming (LP) is one of the most common optimisation tools used to solve the MCFP. The advantages of using an LP solver is that the optimal solution is always found, assuming that such a solution exists. Obviously, what represents an optimal solution is based on how the MCFP formulation is defined. LP solvers have well known limitations that have to be overcome in order to achieve our goal of designing a multipath routing algorithm. Most relevant to this work are the facts that LP solvers are unable to handle multi-objective problems, output more than one solution, and limit the MCFP formulation to be built from linear components only. LP's inability to handle multiple objectives means that the solution provided will always be biased towards a single objective. The limitation of having access to only linear formulations hampers the ability for the objectives to accurately reflect the system being modelled.

To overcome said limitations, this work presents a routing algorithm that solves the MCFP based on Evolutionary Algorithms (EAs). Compared to LP, EAs are capable of solving problems having multiple objectives, able to work with both linear and non-linear formulations and a single run of the algorithm provides an array of different viable solutions to choose from based on the current situation. EAs are a class of optimisers that emulate nature's evolutionary process to find valid solutions for the given problem. EAs build on the concept of natural selection and work on the premise of survival of the fittest to always improve on the previously found solutions by allowing the good solutions to mate and have offspring, while

killing off the bad solutions. However, EAs are known to be sub-optimal; meaning that the pool of solutions generated by an EA is not guaranteed to include the optimal solution, if such a solution exists. Having said this, the LP provided solution can be used by the EA to improve the solutions generated by the EA and combine the advantages of the two algorithms. However, combining these two algorithms comes at the cost of increased computational complexity as both the LP and EA algorithm have to be run.

1.1 Aims and Objectives

The aim of this research is to develop a globally optimal, multipath capable, routing algorithm using the SDN architecture, intended to increase the traffic load a network is able to sustain. The following is a list of objectives tackled during this thesis:

- Survey the literature to establish the current state of the art research being carried out in the area of computer network routing algorithms and the methods used to find such routing solutions.
- Develop a routing algorithm using EAs to overcome the limitations offered by LP.
- Develop a system whereby TCP applications can benefit from *Per-Packet* multipath algorithms.
- Develop a software environment where the developed routing algorithms can be tested.
- Design and develop a switch on said software that can handle *Per-Packet* multipath at any split ratio without running into scalability issues.
- Analyse the performance results of the developed routing algorithms and designed protocols.
- Compare the network performance of the designed routing algorithms between themselves and OSPF.

1.2 List of Contributions

- The design of a *Per-Packet* flow splitting switch, tested using network simulations, capable of handling any split ratio without facing scalability issues.

- Modifications to the Multi-Path TCP (MPTCP) protocol to allow TCP applications to benefit from *Per-Packet* multipath routing solutions.
- Design of a novel EA framework to generate a multi-objective, globally optimal, multipath capable, routing algorithm.
- Modifications to the Path Constrained Maximum-Flow Minimum-Cost (PC-MFMC) formulation to include TCP acknowledgement flows.

1.3 List of Publications

- N. Farrugia, V. Buttigieg, and J. A. Briffa, "A Globally Optimised Multipath Routing Algorithm Using SDN", in *21st Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN)*, Paris, France, Feb. 2018, pp. 1–8. DOI: 10.1109/ICIN.2018.8401633
- N. Farrugia, J. A. Briffa, and V. Buttigieg, "An Evolutionary Multipath Routing Algorithm using SDN", in *2018 9th International Conference on the Network of the Future (NOF)*, Nov. 2018, pp. 1–8. DOI: 10.1109/NOF.2018.8597865
- N. Farrugia, J. A. Briffa, and V. Buttigieg, "Solving the Multi-Commodity Flow Problem using a Multi-Objective Genetic Algorithm", in *2019 IEEE Congress on Evolutionary Computation (CEC)*, Jun. 2019, pp. 2816–2823. DOI: 10.1109/CEC.2019.8790160
- N. Farrugia, V. Buttigieg, and J. A. Briffa, "Multi-Stream TCP: Leveraging the Performance of a Per-Packet Multipath Routing Algorithm When Using TCP and SDN", in *2019 44th Conference on Local Computer Networks (LCN)*, Oct. 2019

1.4 Structure

This thesis is organised as follows. Chapter 2 outlines the most impactful research in the area of TE using SDN, followed by a review of multi-objective solvers, with focus on nature inspired work. Chapter 3 delves into the workings of the developed, multi-objective evolutionary routing algorithms. Chapter 4 outlines the work required to implement the proposed solutions on a network simulation framework. Chapter 5 presents the results of this work. Finally, Chapter 6 concludes this work by summarising the achievements of this work, drawing conclusions from the results achieved, as well as list additional work to further improve the proposed system.

Chapter 2

Literature Review

This chapter starts by describing the MCFP and its close relationship with the area of TE. A summary of two of the seminal works that deployed such a TE solution using SDN, follows. This summary highlights the challenges and the design of alternate algorithms the researchers had to develop to overcome the limitations imposed by LP. Following this, a review on works that use nature inspired routing algorithms to increase network efficiency is given. As one can easily assume, rarely is the case where a routing algorithm is to be optimised for a single objective. As LP is unable to solve multi-objective problems, alternative solvers have to be found that are capable of handling multiple objectives. An overview of the available multi-objective solvers is given, with a section highlighting the most commonly used multi-objective EA frameworks.

2.1 Multi-Commodity Flow Problem

The MCFP [13] is at the fulcrum of the majority of TE related works due to the close proximity of both problems' objectives. The MCFP deals with the routing of commodities over a network, without exceeding any of the network link capacities. The optimisation objectives and constraints that make up the MCFP formulation varies based on the environment the MCFP will be used in. To have a better understanding of how various works in the literature tackled TE by modifying the MCFP to fit their environment, an explanation of the basic MCFP formulation is given next.

Let $\mathcal{G} = (V, E)$ be a loop-free directed graph representing the network topology, where V and E are the set of nodes and links respectively. Each link is represented by $e = (u, v) \in E$ where $u, v \in V$ are the link's source and destination node, respectively. The capacity and cost of each link $e \in E$ is represented by λ_e and γ_e

respectively. Let $F = \{f_1, f_2, \ldots, f_n\}$ be the set of n flows, with $s_i \in V$ and $d_i \in V$ representing the source and destination nodes for flow $f_i \in F$, respectively. Let δ_i represent the data rate requested by flow $f_i \in F$ and the decision variable $r^i_{u,v}$ represent the data rate flow $f_i \in F$ transmits over link $(u, v) \in E$. Using the above notation, the MCFP is defined as follows:

$$\min_{r^i_{u,v}} \sum_{i=1}^{n} \sum_{(u,v)\in E} \gamma_{u,v}\, r^i_{u,v}, \tag{2.1}$$

such that,

$$\sum_{i=1}^{n} r^i_{u,v} \leq \lambda_{u,v} \quad \forall (u,v) \in E, \tag{2.2}$$

$$r^i_{u,v} \geq 0 \quad \forall (u,v) \in E, \quad i = 1, \ldots, n, \tag{2.3}$$

$$\sum_{v:(u,v)\in E} r^i_{u,v} - \sum_{v:(v,u)\in E} r^i_{v,u} = \begin{cases} \delta_i & u = s_i \\ -\delta_i & u = d_i \\ 0 & \text{otherwise} \end{cases} \quad \forall i, u. \tag{2.4}$$

The capacity constraint (2.2) ensures that no link is used beyond its capacity, while (2.3) ensures that no negative flow is assigned. The flow conservation constraint (2.4) ensures that all the data transmitted from the source node s_i is received in full by the destination node d_i and no data is lost at the relay nodes. The MCFP formulation given above is linear because no restriction is set on the number of paths a flow may take [14]. If a constraint is added to the above formulation, such that a flow may only use a single path, the problem becomes the unsplittable MCFP that is NP-hard [14].

As can be deduced from the definition of the MCFP, working out a routing solution requires up to date knowledge on both the network topology and flows that are currently traversing the network. Gaining access to this information using the traditional network architecture with a distributed control plane is very difficult to do in an efficient and scalable manner. Works such as [14–17] that use the MCFP as a foundation for a routing algorithm design, assume that a central router with all the required information exists. SDNs [9] are a relatively new type of network architecture that centralises the control plane, allowing such routing solutions to be deployed and tested on actual networks.

2.2 Traffic Engineering using Software Defined Networks

Akyildiz et al. [18] presents a survey outlining TE research using SDN, and grouped such solutions into four different categories: Flow Management, Fault Tolerance, Topology Update, and Traffic Analysis. *Flow Management* deals with the handling of how new flows entering the network are routed to avoid network congestion. In other words, *Flow Management* is the routing algorithm installed at the network controller. *Fault Tolerance* is a branch of TE research dealing with the ability of the network to recover from a network device or link failure in the shortest amount of time, ideally without impacting the traffic already travelling over the network. Carrier grade networks have a stringent requirement where network stability must be regained after only 50 ms. In the case of failures in an SDN network, the communication overhead required between the network controller and the SDN switches to restore stability might be too long. Having the controller calculate alternative paths at the time failure occurs is known as the reactive approach. An alternative to the reactive approach is the proactive approach. In a proactive approach, the network controller installs an alternate path on the switches to avoid the communication overhead in case of a network failure. The proactive approach minimises the time required for a network to regain stability; however, this comes at the cost of increasing the number of rules to be installed on the network switches. *Topology Update* is an area of research focused on how to best implement rule changes to ensure rule forwarding consistency at either the packet or flow level. In this context, rule forwarding consistency means that a flow or packet will only follow one pre-determined path. Last but not least, *Traffic Analysis* encompasses all the work related to the gathering and collection of network statistics.

Two of the most influential works, that managed to successfully implement a MCFP based TE solution using SDN, are the works done by Google and Microsoft with their B4 [19] and SWAN [7] network architectures, respectively. Since both the B4 and SWAN networks are used in a production environment, they each have components of their solutions' design that fall under each TE category defined in [18]. This literature review will focus on the routing (*Flow Management*) aspect of the two mentioned works as it is the class of TE that the work carried out in this thesis falls under. Particular attention is given to these two contributions as they work on the premise that all network hardware is SDN capable and the network controller has the means to control the applications using the network, which is identical to the setup assumed in this thesis. Both [7] and [19] agree that the status quo of networking is inefficient due to the lack of flow management, where

an application can start transmission at any time and at any data rate. This lack of flow management results in the network to experience network load fluctuations with intermittent peaks of very high network use. To avoid congestion with such an oscillating network load, the network is designed to handle the highest network load which leaves most of the network's resources under utilised for the remainder of the time where the network load is not at its peak. To mitigate the problem described above, SDN is used to control the applications' transmission schedule and rate to maximise the network's efficiency. Applications using the network have inherently different requirements and demands. Due to the large number of application categories that may use the network, it is impossible to design a routing algorithm that handles the requirement of each application independently. This is why [7] and [19] put applications with similar requirements into the same group. Both [7] and [19] sort applications into three distinct groups with similar classification methods. Using the group names given in [7], applications are divided into three groups, listed below in order of decreasing priority:

Interactive Applications directly related with a user's request that are highly sensitive to delay and packet loss.

Elastic Applications that are not as critical as *Interactive* traffic; however, they still have a time limit by which they need to complete their transmission.

Background Applications that have no set or very long deadlines, usually related to maintenance or provisioning services. This application category usually has the largest bandwidth requirements, but, applications in this category are able to handle pauses in transmission and adapt their transmission rate.

Both [7] and [19] use a routing algorithm based on the max-min fair allocation, albeit with some differences in their approach on how to solve such a problem. A max-min fair solution is defined as the solution that maximises utilisation in a way that no flow can be allocated additional resources without penalising the fair share of applications [19].

SWAN [7] routes interactive traffic using standard routing algorithms, such as OSPF, without interference from the network controller due to the application's sensitivity to delay. The two remaining traffic groups are routed using a routing algorithm based on the MCFP with the objective of maximising throughput while preferring shorter paths. The MCFP is solved using LP separately for each application priority class, routing high priority applications first, followed by the next lower priority class of applications. In this manner, high priority applications have a higher probability of being routed over the best paths. Further details on the

implementation and how max-min fairness is achieved can be found in [7]. The routing solution generated from the MCFP, does not take into account the number of rules a switch is capable of handling. This constraint has been omitted as it converts the problem to an Integer Linear Programming (ILP) one, drastically increasing the problem's complexity. A heuristic algorithm is used to modify the solution generated by LP such that the routing solution is able to be deployed on the network without exceeding any of the switches' routing table capacity.

Similarly, Jain et al. [19] replaces the LP based optimal solution routing algorithm with a custom designed, greedy based algorithm, as finding the solution using an LP solver is not fast enough and faces scalability problems. Before explaining how the greedy routing algorithm works, the necessary terminology defined in [19] is given first. A flow group, is the grouping of applications that have the same source, destination and Quality of Service (QoS) requirements. A tunnel represents a path as a sequence of network sites. Finally, a tunnel group represents the amount of traffic to transmit on each tunnel from a given flow group. The designed greedy algorithm works as follows. First, allocate the fair share of data rate to all flow groups on their preferred tunnel. A flow's *fair share* is an arbitrary scale developed in [19] and is a function of the flow's priority. The flow group's preferred tunnel is taken to be the shortest path in terms of aggregate latency. Following this data rate allocation process, any tunnels that use links that are at their maximum capacity are frozen, i.e. cannot be used for further data allocation. This process is repeated until either all flow groups are satisfied or no more viable tunnels exist for the given flow groups. Compared to the LP optimal solution, the described algorithm allocates at least 99% of the bandwidth compared to the optimal solution, achieves similar fairness and generates a routing solution 25 times faster than LP.

Multipath routing is an essential component to use all the available network resources efficiently [2, 3], which is why both routing solutions given in [7, 19] make use of such technology. The greedy algorithm developed in [19], generates split ratios that may not be achievable in practice due to the underlying SDN switch hardware capabilities. Adding a constraint to the problem formulation specifying the split ratios supported by the underlying switches would be equivalent of solving an ILP problem. To resolve this, a second algorithm referred to as *Tunnel Group Quantisation* is developed that modifies these ratios in multiples of 0.5 such that they can be implemented on actual SDN hardware. Details of said algorithm can be found in [19]. The need for the *Tunnel Group Quantisation* algorithm may have been avoided if the authors added the capability to split flows at any given ratio to their already custom designed switches. The routing solution generated in [19] still falls under the category of a *Per-Flow* multipath solution because the routing

algorithm works at the flow group level. Flows from a flow group are divided on different paths based on the result of the *Tunnel Group Quantisation* algorithm.

Combining SDN and multipath technology, the B4 network is able to average 70% link utilisation over long time periods, equivalent to more than double the network efficiency of a standard distributed network. On a similar note, SWAN is able to carry 60% more data than an equivalent network using current distributed network design principles.

The works in [7, 19] serve to show that under certain conditions, a well designed SDN network is able to improve the efficiency of a network by a large margin when compared to the distributed architecture. For further information on TE research using the SDN architecture, including a brief historical background of TE that predates SDN, the reader is referred to the full survey by Akyildiz et al. [18].

2.3 Heuristic algorithms

Heuristic algorithms are a class of algorithms developed to solve problems for which either no method that finds the optimal solution exists, or the time taken to find such a solution is too long for the algorithm to be practical. Both routing solutions used by SWAN [7] and B4 [19] networks resorted to some sort of heuristic algorithms to either overcome LP imposed limitations or generate an approximate solution in a reasonable time frame. Several well known heuristic algorithms such as the Ant Colony Optimisation (ACO) [20], Alienated Ant Algorithm (AAA) [21], and EA [22] have been used in literature to solve TE related problems. ACO algorithms are a class of optimisers primarily designed to find the shortest paths within a graph and take on inspiration from the behaviour of ants [20]. Contrary to the ants' behaviour in the ACO algorithm, ants under the AAA follow the path with the lowest pheromone trail. Also nature inspired, EAs are a class of optimisers that excel at solving multi-objective problems by mimicking nature's evolution process. In addition to solving TE problems, EAs have been successfully used to solve network design [23] and network migration [24] problems. A summary of works using heuristic, nature inspired algorithms to solve MCFP based TE related problems follows.

Masri et al. [14] tries to solve the MCFP with the additional constraint of flows having to use a single path, and multiple sources are able to supply the same information. Finding a solution to such a problem is NP-hard, which led to the authors' use of the ACO algorithm. Due to their setup, the simulated ants start at the destination node and move towards one of the multiple viable source nodes.

The ACO is set to minimise both the time required to satisfy all the requests and the network cost. The performance of the developed ACO routing algorithm is compared with the lower bound for the first objective (minimise request completion time), where the solutions found by the ACO trail very closely to the optimum value when the number of messages to be transmitted are relatively low. As the number of messages starts to increase, the gap between the ACO provided solutions and the lower bound increase. The lack of network simulation results makes it difficult to gauge the performance improvement of such a system over standard routing systems, such as OSPF. Stefano et al. [21] chose the AAA as the basis on which to develop the Adaptive Alienated Ant Algorithm for SDN (A4SDN) architecture. The A4SDN architecture is designed to improve the network performance measured in terms of throughput, delay and packet loss. The AAA is used instead of the ACO algorithm as the ant's behaviour in the former algorithm allows ants to explore unused paths. This leads to the generation of solutions with better load balancing performance as the ants do not converge over a single path. Additionally, AAA has a faster rate of convergence than the ACO algorithm. Compared to the extended Dijkstra algorithm, which calculates the link weights dynamically based on the link's usage, the A4SDN architecture managed to decrease packet loss by 11% and increase the total network throughput by 16%.

Similar to the A4SDN [21] architecture, Yu et al. [25] used the information available at the SDN network controller to shift congested traffic to lightly loaded links. The main objective of the work by Yu et al. is to improve the quality of video streams transmitted over a computer network by shifting them over to less congested links. All video streams entering the network are routed over the shortest path using the Bellman-Ford algorithm. At specific time intervals, the SDN controller queries the switches to gather link usage statistics to determine whether there are links that are currently, or soon to be, congested. Upon sensing link congestion, an EA based routing algorithm is used to find alternative paths for videos that are currently transmitted over congested link(s). The EA is designed to minimise the path cost, where the path cost depends on both the path's aggregate delay and remaining capacity. Each chromosome in the EA represents a path as a list of sequential network switches. The first and last gene in a chromosome remain unchanged during the course of the evolution as those elements represent the source and destination node, respectively. The length of each chromosome, and each gene's value are generated at random. This generation method leads to the creation of a number of unfeasible solutions as a path may contain duplicate switches. It is to be noted that to avoid the generation of unfeasible solutions, the authors could have used path discovery algorithms to find paths with the randomly determined length.

Having said this, the availability of only feasible paths in the initial population may limit the effectiveness of the searching capability of the EA. Therefore, making the proposed change needs to be thoroughly tested to ensure that it does not have a negative impact on the EA's performance. Using a relatively simple EA to move video streams to less congested links reduced the packet loss rate by approximately 20% and improved the Peak Signal-to-Noise Ratio (PSNR) by nearly 100%, when compared to the Bellman-Ford algorithm.

The work by El-Alfy et al. in [15–17] developed an EA routing algorithm aimed at minimising the routing and load balancing costs, designed to work on Multi-Protocol Label Switching (MPLS)/Generalised MPLS (GMPLS) networks using identical constraints as those defined by the MCFP. The routing cost objective is designed to favour transmission over paths with low cost, while the load balancing objective aims to minimise the use of heavily used links. El-Alfy et al. opted for a three dimensional chromosome with each gene being a two dimensional matrix. Each gene is a representation of the traffic generated on each link by a given flow. To contain the problem, paths longer than four hops are excluded from the search space and a flow is only allowed transmission over a maximum of two paths. Two different LP based methods are used to generate the initial population, where both methods vary the link capacity and cost, up to a specified minimum threshold to get multiple solutions from running the same LP formulation. The first method separates the two objectives into two separate LP formulations and fills half of the population with solutions gathered when optimising one objective and the remaining half when optimising for the other objective. The second method combines the two objectives into a single objective by using the weighted sum method. The fact that the initial population is generated with the help of LP, and the lack of results presented where the EA is initialised from a random population, puts into question the ability of the developed EA to search the entire search space and approximate the optimal Pareto Front. Additionally, as admitted to by the authors, the three dimensional chromosome design is not very efficient in terms of space required to represent a routing solution.

2.4 Single vs Multi Objective problems

This section is aimed at clarifying the difference between a true multi-objective solver and the methods that exist which convert a multi-objective problem into a single-objective problem to be able to use single-objective solvers such as LP. A multi-objective problem is a problem that has multiple objectives to be optimised for, with usually conflicting interest. Using car purchasing as an analogy,

an ideal car would be inexpensive, and comfortable. However, these objectives are usually at odds with each other where improving one objective comes at the cost of compromising some other objective. Using the car analogy mentioned earlier, a cheap car is not comfortable, while a comfortable car is not cheap. When dealing with a multi-objective problem, no single optimal solution exists that maximises all of the given objectives. Instead, a multi-objective problem has multiple valid optimal solutions where each solution performs better in one objective, but worse in another. The collection of these optimal solutions is referred to as the *Pareto Front* [22].

Having said this, a single solution to a given problem is required most of the time irrespective of the number of objectives. The difference between a true multi-objective solver or a converted one boils down to the time where the user inflicts his opinion on the importance of the objectives. As will be explained next, solving a multi-objective problem using a single-objective solver requires the user to prioritise the objectives before the solver even starts to try and find a solution. Two methods commonly used to solve a multi-objective problem using single objective tools such as LP are the following. The first method is the *weighted sum* method, where the objectives are combined together with a weight assigned to each objective [22]. The weight to assign each objective needs to be set before running the optimisation algorithm. Choosing the weights to assign to each objective is subjective and non trivial [22], mainly due to the fact that this weight preference is given without any knowledge of the consequences of such values. As a single-objective solver is used, only a single solution will be given. One may argue that in order to have an approximation of the *Pareto Front*, one may achieve this by modifying the weights and re-running the algorithm. However, such method does not guarantee coverage of the entire search space and may leave gaps in the generated *Pareto Front* [26]. Another solution is to solve the objectives in order of priority referred to as the *Lexicographic Approach* [27]. The disadvantage of such a solution is that only one solution is generated, with the objective priority given before running the optimiser. Having said this, the above methods are useful when the user knows a priori the objectives' priority or objective weights. Pulling this off requires a very deep understanding of the problem being solved.

It is important to clarify that the decision of which solution to use is a totally separate and independent problem as to the one that deals with the search and approximation of the *Pareto Front* [27]. The job of a multi-objective solver is to approximate the *Pareto Front* without any preferences given to any of the objectives. It is after this step that the user is then faced with a number of viable solutions of different compromise levels where a decision of which solution to pick is to be

taken. The advantage here compared to the single objective alternative is that an approximation of the optimal *Pareto Front* is available to help you make a more informed decision. Multi-Objective Evolutionary Algorithms (MOEAs) are the ideal algorithms to generate an approximation of the *Pareto Front* in a single run due to their population based approach [28]. Konak et al. [29] states that 90% of the work that required a solution to a multi-objective optimisation problem chose a true multi-objective solver. From those 90%, a staggering 70% of them opted to use an EAs as their algorithm of choice. A major advantage offered by a MOEA is that the algorithm designer does not need to assign any priorities or weights to the objectives [29]. Additionally, compared to EAs, stochastic optimisation techniques such as the ACO may get stuck at a good approximation with no guarantee of finding the optimal trade off [27].

2.4.1 Multi-Objective Evolutionary Algorithms

When choosing to develop an optimisation algorithm based on MOEA, one is faced with a plethora of different MOEA frameworks to choose from [29]. The caveat being that most frameworks have been designed and tweaked to solve one particular problem or one particular class of problems. However, a few MOEA frameworks stand out from the rest as they have been successfully used in literature to solve a number of different problems in different research areas thanks to their versatility and adaptability. One such MOEA framework, that has been cited more than 15 000 times, is the Nondominated Sorting Genetic Algorithm (NSGA) II [30] developed by Kalyanmoy Deb in 2002. NSGA-II is an MOEA framework that evolved from NSGA-I [31]. Compared to NSGA-I, the NSGA-II algorithm reduces the computational complexity required to compute the non-dominated sorting, uses elitism, and removes the need to set a sharing parameter used to ensure a diverse population. Elitist EAs keep the best found solutions in between generations resulting in an EA with better performance and a faster convergence rate than their non-elitist counterparts. Recently, a newer version, the NSGA-III [32, 33] has been published. The NSGA-III algorithm is designed to solve many-objectives problems and requires an estimation of the Pareto Front, which is a problem within itself. Many-objectives problems usually refer to problems with three or more objectives, while multi-objective problems usually refer to problems with two or three objectives [26]. Another well known MOEA framework is the SPEA2 [34] algorithm. The performance of both the SPEA2 and NSGA-II algorithm is very similar; however, the SPEA2 requires an additional parameter, the archive size, when compared to the NSGA-II algorithm [34].

Chapter 3

Globally Optimal Multipath Routing

During the literature review carried out and summarised in Chapter 2, the need for a multi-objective, multipath capable, globally optimal routing algorithm surfaced. The ability for a globally optimal, multipath capable routing algorithm to increase network efficiency has been made abundantly clear in previous chapters. What may not be so clearly defined is the need and advantages offered by a multi-objective routing algorithm. The main benefit of having a multi-objective algorithm is the fact that no weights or priorities need to be given to the objectives a priori, allowing the algorithm to explore the Pareto Front unhindered in all directions. EAs are a class of optimisers that excel in such a scenario mainly thanks to their population approach that allows them to output a number of different viable solutions from a single run of the algorithm. Having a true multi-objective algorithm gives the network operator the full picture of the objective values and the compromise between them before taking a decision on which solution to pick. For example, it may be too expensive to reach the maximum network throughput, pushing a network operator to decide against such a solution. Due to the time constraints imposed on a routing algorithm to generate a valid routing solution, it has to be operated without any human intervention. Therefore, an algorithm has to be developed that selects a solution from all those generated by an EA. Developing such an algorithm depends heavily on the network's use case and is beyond the scope of this work. The alternative of using a multi-objective solver is to combine multiple objectives into a single objective to use a single objective solver such as LP. Using such a method requires the need for preference or priority to be given to the objectives without having the knowledge of the shape of the Pareto Front. In addition, the use of an EA allows the routing algorithm designer to use both integer variables

and non-linear expressions without affecting the algorithm's running time, as is the case with an LP solver.

One of the fundamental objectives for a routing algorithm is to maximise the network load carried over the network. A second, often conflicting, objective being to minimise the delay experienced by flows due to its direct relationship with QoS. Therefore, the objectives of the designed routing algorithms are set to maximise the network flow while preferring paths with low delay values. To contain the problem, all of the routing algorithms mentioned here assume a static flow set. The assumption of a static flow set is one of the main limitations of this research; however, methods of how to update the developed EAs to handle a dynamic flow set are given in Section 6.1. The routing algorithms presented in this work are designed with the assumption that they are going to be used solely by TCP flows. Two reasons are behind this decision. First, TCP is one of most used transport layer protocol along with User Datagram Protocol (UDP). Second, solutions that work with the TCP protocol are guaranteed to work with UDP; however, the opposite is not true.

3.1 Notation

As in Section 2.1, let $\mathcal{G} = (V, E)$ be a loop-free directed graph representing the network topology, where V and E are the set of nodes and links respectively. Each link is represented by $e = (u, v) \in E$ where $u, v \in V$ are the link's source and destination node, respectively. Let $\bar{e} = (v, u) \in E$ represent the reverse of link $e = (u, v) \in E$. The capacity and cost of each link $e \in E$ is represented by λ_e and γ_e, respectively. The link cost value depends on what one seeks to minimise, and can take a myriad of other values, such as the actual financial cost to use a given link. In this work, the cost of a link is set equal to the link's delay value. Such a link cost definition is used as we seek the minimisation of a flow's delay. Let $F = \{f_1, f_2, \ldots, f_n\}$ be the set of n flows, with $s_i \in V$ and $d_i \in V$ representing the source and destination nodes for flow $f_i \in F$, respectively. Let δ_i represent the data rate requested by flow $f_i \in F$. A path is defined as the set of links that connect a sequence of distinct nodes from the flow's source to the destination. Let k represent the maximum number of paths a flow is allowed to take, with the actual number of paths flow f_i is allowed to take equal to k_i, where $k_i \leq k$. We define $P_i = \{p_{i,1}, p_{i,2}, \ldots, p_{i,k_i}\}$ as the set of paths related to flow f_i and $g_{i,j} \in \mathbb{R}_{\geq 0}$ as the data rate flow f_i transmits on path $p_{i,j}$. The aggregate delay value of path $p_{i,j}$,

denoted by $\phi(p_{i,j})$, is calculated using

$$\phi(p_{i,j}) = \sum_{e \in p_{i,j}} \gamma_e. \tag{3.1}$$

Let $p_{i,\min}$ represent the path with the lowest delay from the set P_i, given by

$$p_{i,\min} = \arg\min_{p_{i,j} \in P_i} \phi(p_{i,j}). \tag{3.2}$$

Finally, let $\alpha(g_{i,j})$ represent the TCP acknowledgement flow generated when flow f_i transmits at a data rate of $g_{i,j}$ on path $p_{i,j}$.

3.2 Path Constrained Maximum Flow Minimum Cost

The MCFP is only concerned with minimising the network cost because of the flow conservation constraint that states that all flows must be allocated the data rate they requested in full. This behaviour is not desired because if there is not enough capacity in the network to fully satisfy all the flows, no solution to the MCFP exists. Using LP, one cannot optimise for the two objectives highlighted earlier; network flow maximisation and flow delay minimisation, at the same time. To overcome this limitation, the objectives are separated and solved in succession. The pair of problems that aim to maximise the network flow and minimise the cost are termed as the Multicommodity Maximum-Flow Minimum-Cost (MMFMC) problem [35]. For more information on the MMFMC problem, the reader is referred to [35]. In this work, we make use of the PC-MFMC problem which sets a limit on the paths a flow is allowed to travel on. The reason behind this limitation is to exercise control on the set of paths a flow may use. Having such control is important as this allows us to manage the algorithm's complexity by varying the number of paths each flow is allowed to use accordingly. If the algorithm's complexity is of no concern, the flows may be given access to all the paths that exist between a source and destination.

As will be explained in more detail in the next section, the EA is designed in such a way that a flow is allowed to travel on a given set of paths. Therefore, the solution to the PC-MFMC using LP serves to compare the quality of the solutions generated by the EAs when both problems have the same setup.

The *Maximum Flow* problem is solved first and is given by

$$\max_{g_{i,j}} \sum_{i=1}^{n} \sum_{j=1}^{k_i} g_{i,j}, \tag{3.3}$$

such that

$$g_{i,j} \geq 0 \quad \forall\, i, j, \tag{3.4}$$

$$\sum_{j=1}^{k_i} g_{i,j} \leq \delta_i \quad \forall\, i, \tag{3.5}$$

$$\sum_{i,j:e \in p_{i,j}} g_{i,j} + \sum_{i,j:\bar{e} \in p_{i,j}} \alpha(g_{i,j}) \leq \lambda_e \quad \forall\, e. \tag{3.6}$$

Constraint (3.4) ensures that no negative data rate is assigned. Constraint (3.5) makes sure that no flow is allocated more data rate than what it requested. Constraint (3.6) guarantees that no link is used beyond its capacity, including the acknowledgement flows generated by TCP. Let $\mathcal{T}$ represent the total network flow allocated to the network by the *Maximum Flow* solution in (3.3), given by

$$\mathcal{T} = \sum_{i=1}^{n} \sum_{j=1}^{k_i} g_{i,j}. \tag{3.7}$$

The *minimum cost* solution is then formulated as

$$\min_{g_{i,j}} \sum_{i=1}^{n} \sum_{j=1}^{k_i} g_{i,j} \phi(p_{i,j}), \tag{3.8}$$

such that constraints (3.4), (3.5), (3.6), and

$$\sum_{i=1}^{n} \sum_{j=1}^{k_i} g_{i,j} = \mathcal{T} \tag{3.9}$$

are met. Constraint (3.9) guarantees that the *Minimum Cost* solution allocates the same total network flow to that found by the *Maximum Flow* solution. In [35], the *Minimum Cost* solution is set to allocate the same total network flow to that found by the *Maximum Flow* solution by restricting flows to transmit at the data rate allocated by the *Maximum Flow* solution. In other words, the *Minimum Cost* solution by Szymanski [35] does not have constraint (3.9) and replaces constraint (3.5) with

$$\sum_{j=1}^{k_i} g_{i,j} = \mathcal{D}_i \quad \forall\, i, \tag{3.10}$$

where $\mathcal{D}_i$ represents the data rate allocated to flow f_i by the *Maximum Flow* solution in (3.3). Compared to the formulations used in [35], the ones presented here do not restrict the *Minimum Cost* solution to use the flow assignment used by the *Maximum Flow* solution. This gives the *Minimum Cost* solution the freedom to adjust a flow's allocated data rate in search for a lower cost solution, as long as the same data rate found by the *Maximum Flow* solution is kept. Additionally, the link capacity constraint formulations are updated to take into account the TCP acknowledgement flows generated.

3.3 Evolutionary Based Routing Algorithm Framework

The EA based routing algorithm presented in this work evolved with the addition of new protocols and scenarios where this algorithm is being used. The MOEA-I algorithm described in Section 3.4 is the final iteration of the EA where the use of standard TCP flows is assumed. Subsequent to the design and publication of the MOEA-I algorithm, the MPTCP protocol is modified to enable TCP flows to benefit from a *Per-Packet* multipath routing solution. The modified MPTCP protocol affected which variables the flow's delay value is calculated on, requiring changes to the delay related objectives. Therefore, a second version coined MOEA-II is designed for use with the MPTCP protocol explained in Section 4.1.2. The assumption of using MPTCP allowed the MOEA-II algorithm to meet its requirements with just two objectives, compared to the three objectives required by the MOEA-I algorithm.

Both algorithms (MOEA-I and MOEA-II) share the same EA framework described in this section and differ only in their objectives and mutation operator implementations. The objectives and mutation operators used by the MOEA-I and MOEA-II algorithms are given in Sections 3.4 and 3.5, respectively. Both routing algorithms are presented in this work as they cater for two different scenarios and instil an appreciation on the work involved when designing an EA based routing algorithm. To avoid any bias between objectives, all objectives are normalised to fit within the range $[0, 1]$.

Let $\mathcal{P}$, χ, ω and ψ represent the population size, number of generations, crossover probability and mutation probability, respectively. The pseudocode for a basic EA using NSGA-II is given in Algorithm 1 to better understand how all the operators described in this section are used to make an EA. The NSGA-II algorithm is used to select solutions that make up the current population using the solutions found in

Algorithm 1 Pseudocode for an Evolutionary Algorithm

```
P = Population Size
χ = Number of Generations
ω = Crossover Probability
ψ = Mutation Probability

population = GenerateInitialPopulation(P)
for ∈ 1, 2, ..., χ do
    offspring = TournamentSelection(population, P)
    for c_i ∈ offspring, i = 1, 3, 5, ..., P do
        z = random(0, 1)
        if z < ω then
            Crossover(c_i, c_{i+1})
        end if
    end for
    for c_i ∈ offspring do
        z = random(0, 1)
        if z < ψ then
            Mutate(c_i)
        end if
    end for
    CalculatePopulationFitness(offspring)
    population = NSGA-II([population + offspring], P)
end for
```

the generated offspring and the previous population. By taking into consideration both the generated offspring and the previous population, the NSGA-II algorithm is able to preserve the best solutions found in the previous generation, which is what makes it an elitist EA. The NSGA-II algorithm starts by sorting the solutions into a number of non-dominated fronts. The first non-dominated front is filled by solutions that are not dominated by any solution from the set of solutions found. The second non-dominated front is filled by solutions that are only dominated by the solutions in the first non-dominated front. This procedure is carried out until all solutions are assigned to a non-dominated front. Once all solutions are assigned to a non-dominated front, the new population is generated by fitting as much non-dominated fronts as possible without exceeding $\mathcal{P}$. In instances where adding the solutions that make up the non-dominated front would exceed $\mathcal{P}$, the solutions that are in the least crowded region of that front are chosen. This selection method is used to preserve the diversity in the chosen population.

The *TournamentSelection* function is used to select a pair of chromosomes at random to perform crossover. The *TournamentSelection* as used in Algorithm 1 takes a list of individuals and the number of individuals to select as parameters and returns the list of chosen individuals. The *TournamentSelection* function works by randomly choosing two chromosomes from the population and adds the best one to the list of chosen individuals. This process is repeated until the number of individuals is met. The choice of which chromosome to retain is made based on dominance. The chromosome that is not dominated by the other is kept. In the instance where the two chromosomes are on the same non-dominated front, the chromosome that is in a less crowded area is chosen. For further reading on the NSGA-II function, the *TournamentSelection* and MOEAs in general, the reader is referred to [22].

3.3.1 Chromosome Representation

The design of the chromosome is the foundation to any EA as it represents the way a solution is formulated. The chromosome C is defined as the sequence $C = (G_1, G_2, \ldots, G_n)$ where $G_i = (g_{i,1}, g_{i,2}, \ldots, g_{i,k_i})$ is the sequence of genes related to flow f_i. Each element $g_{i,j} \in \mathbb{R}_{\geq 0}$ represents the data rate that flow f_i is to transmit on path $p_{i,j}$. The chromosome has been designed to be able to accurately represent a routing solution while incorporating the flow conservation constraint in such design, and scale independently of the underlying network topology. The flow conservation constraint states that all flow transmitted from a source node must be received in full to the destination nodes, with no loss in the relay nodes. As each gene in the chromosome represents the data rate to transmit on a path, as

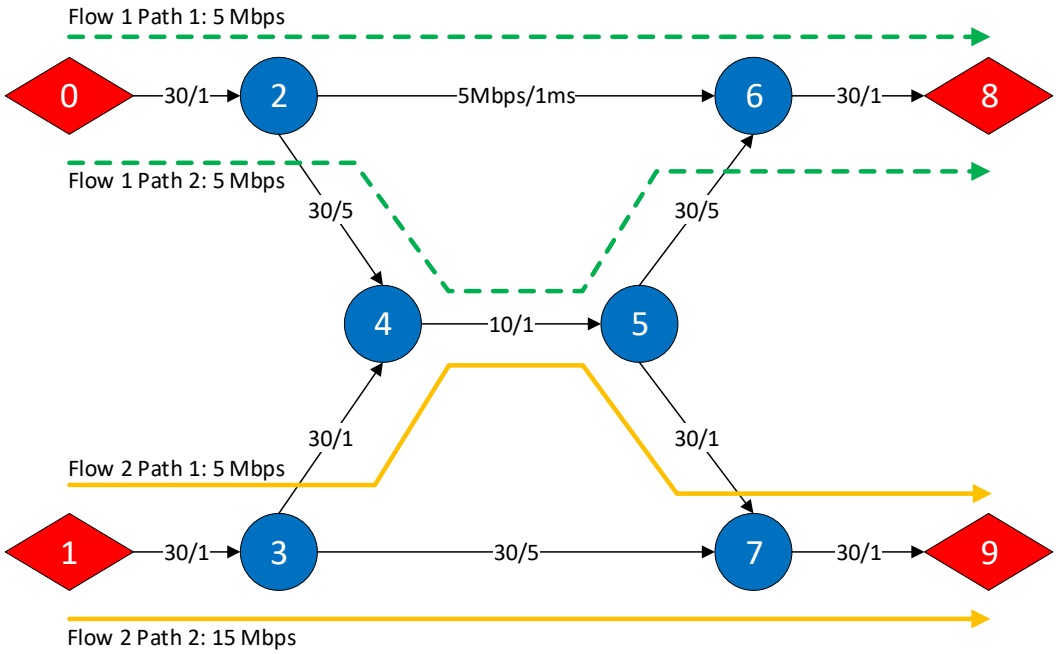


Figure 3.1: Butterfly network used to explain the chromosome representation of a network. Flows 1 and 2 are transmitting at a data rate of 10 Mbps and 20 Mbps respectively. Each flow is allowed to transmit on two paths, as shown by the green and yellow path markers [11].

long as the path starts from the source and ends at the destination node, then the flow conservation constraint will always be satisfied. Compared to the chromosome design proposed by El-Alfy et al. in [15–17], the size of the chromosome developed in this work is independent of the network size. The size of the designed chromosome is dependent only on the number of flows to route and the set paths each flow is able to use.

Using the set up shown in Figure 3.1 as an example, where Flow 1 is transmitting at 10 Mbps, Flow 2 is transmitting at 20 Mbps and both have $k = 2$, the chromosome representation is equal to $C = ((5, 5), (5, 15))$.

3.3.2 Initial Population Generation

The initial population is generated using the following procedure. For each flow f_i, the number of paths the flow is allowed to use, ν, is randomly selected using a uniform integer random number from the range $\nu \in \{0, 1, 2, \ldots, k_i\}$. Subsequently, ν paths are chosen at random from the set P_i. The fraction of data rate the flow is to transmit compared to its requested data rate δ_i is randomly determined using

$$\hat{\delta}_i = \delta_i \times z, \tag{3.11}$$

C_a		C_b
(G_1^a, G_2^a, G_{3^a})		(G_1^b, G_2^b, G_3^b)
$((2,10),(1,5),(2,3))$	⤫	$((1,3),(7,1),(3,2))$
$((2,10),(7,1),(2,3))$		$((1,3),(1,5),(3,2))$

Figure 3.2: Crossover example where the genes related to Flow 2 are swapped to generate a new routing solution.

where $z \in \mathcal{U}(0,1)$. $\mathcal{U}(0,1)$ represents a random source uniformly distributed between 0 and 1. For each of the chosen paths $p_{i,j}$, the smallest link capacity along that path is calculated by

$$\rho(p_{i,j}) = \min_{e \in p_{i,j}} \lambda_e, \tag{3.12}$$

and the corresponding gene is set to $g_{i,j} = \min(\rho(p_{i,j}), \hat{\delta}_i)$. Genes for paths that were not in the chosen subset are set to zero. This population initialisation method may create solutions that break the constraints defined by the MCFP. In such instances, the chromosome is repaired using the methods described in Section 3.3.5.

3.3.3 Crossover

The crossover operator is used to generate new offspring (routing solutions) by mating two chromosomes together, referred to as the parent chromosomes, to generate two new offspring solutions. Two parent chromosomes, C_a and C_b, are selected using dominance based tournament selection [22]. For every crossover operation, a mixing ratio $z \in \mathcal{U}(0,1)$ is chosen. Each gene in the sequence $(G_1^a, G_2^a, \ldots, G_n^a)$ is swapped with its corresponding sequence $(G_1^b, G_2^b, \ldots, G_n^b)$ with probability $z \in \mathcal{U}(0,1)$. A random mixing ratio is used to allow the possibility of an offspring to inherit most of the genes from a single parent. Figure 3.2 shows an example of a crossover operation where the genes related to Flow 2 are swapped to create two new routing solutions.

3.3.4 Mutation

While the crossover operator generates new routing solutions by combining chromosomes together, it does not modify any of the flow's data rate assignments as this task is left to the mutation operator. The mutation operator works on a single chromosome, modifying the gene sequences related to a fraction μ of flows, chosen at random, within that chromosome. For every gene sequence G_i that is selected for mutation, the mutation operation selects a subset $\tilde{P}_i \subseteq P_i$ of paths which flow f_i is allowed to use. Once $\tilde{P}_i$ is chosen, the paths are considered in random order, transmitting as much data as possible until the flow's requested data rate is met

or all paths are used. This data rate assignment does not exceed any link capacity and takes into account all of the other flow data rate assignments. The two MOEAs designed here employ different methods as to how the subset $\tilde{P}_i$ is chosen; therefore, the explanation is given in their respective sections.

3.3.5 Constraint Handling

The chromosome's design already ensures that a number of constraints are met. Two additional constraints that are not implicitly satisfied remain and are:

The flow over-provision constraint Ensures that a flow is not allocated more data rate than requested.

The link capacity constraint Ensures that no link is allocated more data rate than its capacity.

Chromosomes are first checked for over-provisioned flows, followed by a check for over-capacity links. The order is important because it is pointless to fix over-capacity links when over-provisioned flows may still be present in a given solution. After a crossover operation is performed, the two newly generated solutions are checked against the link capacity constraint. The crossover operation does not modify the flows' data rate assignment; therefore, there is no need to validate the flows' over-provision constraint. The mutation operation does not require any validations because the current network usage is taken into account when assigning data rate on paths, and no flow is assigned more data rate than requested. Finally, the method used to initialise the first population requires that each chromosome is checked for both the flow over-provision and link capacity constraints.

Flow Over-Provision Constraint

Flow f_i is said to be over provisioned when its allocated data rate exceeds that requested, i.e.

$$\sum_{j=1}^{k_i} g_{i,j} > \delta_i. \tag{3.13}$$

If excess flow is present, the excess removal algorithm described in Section 3.3.6 is used to remove the excess in an unbiased way from the genes in G_i.

Link Over-Provision Constraint

For the network to perform in accordance to the data rates allocated by the routing algorithm, no link must be allocated more data rate than its capacity as this will

cause network congestion. A link $e \in E$ is said to exceed capacity if

$$\sum_{i,j:e \in p_{i,j}} g_{i,j} + \sum_{i,j:\bar{e} \in p_{i,j}} \alpha(g_{i,j}) > \lambda_e \quad \forall\, e. \tag{3.14}$$

For every link found to be over capacity, the excess removal algorithm described in Section 3.3.6 is used to remove the excess in an unbiased way from the genes in the set $\{g_{i,j} : e \in p_{i,j}\}$. Since the excess removal operation affects a whole path, links other than the one that triggered the operation may be affected. This means that the order in which links are considered will have an effect on the final solution obtained. To reduce bias, links are considered and repaired in a random order. For the same reason, the link usage values calculated by (3.14) have to be recalculated after every excess removal operation. This process is terminated when no over-capacity links remain.

3.3.6 Excess Removal Algorithm

The excess removal algorithm is used to remove a known excess amount from a set of values while being as fair as possible in relation to the amount to remove from each element in the given set. Let $G = \{g_1, g_2, \ldots, g_\kappa\}$ represent the sequence of κ genes, determined by the flow over-provision constraint or link capacity constraint, from which we need to remove an excess value of τ. That is, we want to determine an updated sequence of genes $G' = \{g'_1, g'_2, \ldots, g'_\kappa\}$ such that $0 \leq g'_i \leq g_i$, $i \in \{1, 2, \ldots, \kappa\}$ and $\sum_{i=1}^{\kappa} g_i - g'_i = \tau$. Let ξ_i represent the amount to remove from gene g_i, such that $g'_i = g_i - \xi_i$. We randomly determine each ξ_i with the constraints imposed by G, τ, and previously determined ξ_j, $j < i$:

$$0 \leq \xi_i \leq g_i \tag{3.15}$$

$$\tau - \sum_{j=1}^{i-1} \xi_j - \sum_{j=i+1}^{\kappa} g_j \leq \xi_i \leq \tau - \sum_{j=1}^{i-1} \xi_j \tag{3.16}$$

Each ξ_i is chosen uniformly at random within a range satisfying both constraints. To avoid introducing a bias in the evolutionary algorithm, the genes in G are considered in a random order.

3.4 MOEA-I

3.4.1 Objectives

The fitness of a given routing solution is based on three objectives: the maximisation of the total network flow, the maximisation of the proportion of flows with

minimum delay and the minimisation of the total number of flow splits, represented by $\mathcal{O}_1$, $\mathcal{O}_2$, and $\mathcal{O}_3$, respectively.

Total Network Flow Objective

One of the fundamental requirements of a globally optimal routing algorithm is to maximise the total network data rate, as this has a direct impact on the network efficiency and flow satisfaction rate. The total network flow objective is given by

$$\mathcal{O}_1 = \sum_{i=1}^{n} \sum_{j=1}^{k_i} g_{i,j}. \tag{3.17}$$

We normalise this objective by dividing it with the total requested data rate across all flows, $\sum_{i=1}^{n} \delta_i$.

Proportion of Flows with Minimum Delay

Another key requirement of a routing algorithm is to favour transmission on paths with lower delay values. The PC-MFMC solves this problem by minimising the total network cost, where the total network cost is the summation of the costs of all links. The cost of a link is equal to the multiplication of the data rate transmitted on that link with its delay. This objective works well when used in the PC-MFMC problem, because the *Minimum Cost* problem is solved after the *Maximum Flow*. This is important because the solution to the *Maximum Flow* problem instructs the *Minimum Cost* problem on the total network flow that must be routed when generating a routing solution. Without such constraint on the total network flow transmission, the *Minimum Cost* solution would not transmit anything, as the lowest possible cost value is zero. Because of this undesirable property, the *Minimum Cost* objective as defined by the PC-MFMC problem is unsuited for optimisers where the network flow is not fixed. If such an objective were to be used in an EA, it would steer the EA to favour solutions with no transmission, in direct conflict with the Total Network Flow objective. We overcome this problem with an objective that is independent of the allocated data rate for each flow, and instead reflects the proportion of the data that is transmitted on the path with lowest delay.

The proportion of flows with minimum delay objective is given by

$$\mathcal{O}_2 = \sum_{i=1}^{n} D_i, \tag{3.18}$$

where

$$D_i = \begin{cases} 0 & \eta_i = 0, \\ \frac{1}{\eta_i} \sum_{j=1}^{k_i} \frac{g_{i,j}}{\phi(p_{i,j}) - \phi(p_{i,\min}) + 1} & \text{otherwise}, \end{cases} \tag{3.19}$$

$$\eta_i = \sum_{j=1}^{k_i} g_{i,j}. \tag{3.20}$$

Observe that η_i is the allocated data rate for flow f_i. For flow f_i, a value $D_i = 1$ signifies that all of the allocated data rate is transmitted over the path with the lowest delay value, whereas a value of $D_i = 0$ represents no transmission. This objective is normalised by dividing it with the number of flows n.

Flow Splits

A final requirement of our routing algorithm is to minimise the number of flow splits. A flow transmitted on a single path is said to have no flow splits, a flow transmitted on two paths is said to have one split etc. This is important for two reasons. First, flow splits have a significant effect on the performance of TCP flows [10]. Second, fewer paths require less entries in the routers' network tables, which are a limited and expensive resource. In the first version of this algorithm [10], discouraging the use of multipath was achieved by minimising the total number of paths used. The path usage minimisation objective has been replaced as it does not quantify the number of flows that are split, and similar to the minimise cost objective mentioned previously, its lowest value of zero represents no transmission. Taking all of this into account, the new objective aims to reduce the number of flow splits and is given by

$$\mathcal{O}_3 = \sum_{i=1}^{n} \omega(\psi_i - 1) + \frac{\sum_{i=1}^{n} (\psi_i - \omega(\psi_i))}{1 + \sum_{i=1}^{n} (k_i - 1)}, \tag{3.21}$$

where

$$\psi_i = \sum_{j=1}^{k_i} \omega(g_{i,j}), \tag{3.22}$$

$$\omega(x) = \begin{cases} 1 & x > 0, \\ 0 & \text{otherwise} \end{cases}. \tag{3.23}$$

Observe that ψ_i represents the number of paths used by flow f_i. The first term of (3.21) is an integer value representing the number of flows that are being transmitted on two or more paths. This term is a direct indication of how many flows

are being split, and serves to steer the MOEA towards minimising the number of flows that are split. The second term of (3.21) is the normalised total number of flow splits, with a value in the range $[0, 1)$. This term allows the MOEA to distinguish between solutions that have an identical number of flows that are split, by taking into account the total number of splits for a given solution. This gives the MOEA the ability to favour solutions with lower total flow splits. We normalise this objective by dividing it by $1 + n$.

3.4.2 Mutation

MOEA-I employs three different path selection mechanisms. The method chosen to mutate each gene sequence G_i selected for mutation is selected at random, with equal probability, from the three methods explained next.

Minimise Number of Paths

This method attempts to reduce the number of paths used by a flow, by randomly choosing a number of paths $\nu \in \{0, 1, \ldots, k_i\}$, with probability $\Pr\{\nu\}$ given by

$$\Pr\{\nu\} = \frac{k_i + 1 - \nu}{\sum_{i=1}^{k_i+1} i}, \tag{3.24}$$

that diminishes linearly with increasing ν. These ν paths are chosen uniformly at random from P_i to form $\tilde{P}_i$.

Minimise Delay

This method attempts to minimise the flow's delay by favouring transmissions on paths with lower aggregate delay value. For a given flow f_i, the probability of including path $p_{i,j}$ in $\tilde{P}_i$ is given by

$$\Pr\{p_{i,j}\} = 0.95 \frac{\phi(p_{i,\min})}{\phi(p_{i,j})}. \tag{3.25}$$

In this manner, the path with the smallest cost has a 95% chance of being selected and included in $\tilde{P}_i$. Paths with higher cost have a diminishing probability of being selected, with a linear relationship to the ratio of costs. The probability of choosing the path with the lowest delay is not set at 100% so as to allow this mutation operator to generate a solution where the path with the lowest aggregate delay value is not used.

Maximise Flow

The objective of this method is to transmit as much of the flow's requested data rate as possible over all available paths; thus, $\tilde{P}_i = P_i$.

3.5 MOEA-II

3.5.1 Objectives

The fitness of a given routing solution is based on two objectives: the maximisation of the total network flow, and the minimisation of the application's estimated mean end-to-end delay, represented by $\mathcal{O}_1$, and ($\mathcal{O}_4$), respectively.

Total Network Flow Objective

The total network flow objective used by MOEA-II is identical to the one used by MOEA-I and is given in (3.17).

Estimated Mean End-To-End Delay

When using MPTCP, the mean delay experienced by the application is affected by both the path delay values as well as the data rate transmitted on each path. An application's end-to-end delay is defined as the time taken from when the transmitting application sends a byte of data, to when the receiving application receives that same byte of data. Modelling the interaction between the packets to calculate the mean end-to-end delay is not trivial. Due to the complexity involved in modelling the actual mean end-to-end delay value, an approximation of the application's end-to-end delay measurement is used. In order to determine what is a good approximation of such a metric, a simple custom developed network model is used. The model consists of a simple network with two nodes and a variable number of links connecting the nodes together. To analyse what had the biggest influence on an application's mean end-to-end delay, the number of links, link delay values and fraction of data rate to transmit on each link was varied, taking note of the application's mean end-to-end each time a variable was modified. Using such a simple model it was discovered that the average end-to-end delay tends to be very close to the largest path delay value, from the set of paths used. The development of an accurate model of the mean end-to-end delay is an area worth investing time into as it gives the EA a more accurate model of reality. Having an accurate representation of the objectives gives the EA the ability to differentiate between solutions with higher accuracy, which leads to the better generation of solutions by the EA. The aim of this objective is to minimise the transmission rate

on paths with large delay values from the set of paths associated with each flow, ergo reducing the application's delay and is given by

$$\mathcal{O}_4 = \sum_{i=1}^{n} \mathcal{F}_i, \tag{3.26}$$

where

$$\mathcal{F}_i = \frac{\sum_{j=1}^{k_i} g_{i,j}}{\sum_{i=1}^{n} \sum_{j=1}^{k_i} g_{i,j}} \times \max(\phi(\hat{P}_i)). \tag{3.27}$$

$\mathcal{F}_i$ represents the metric value for flow f_i and the set $\hat{P}_i \subseteq P_i$ includes all the paths $p_{i,j}$, where $g_{i,j} > 0$. $\mathcal{F}_i$ is normalised with respect to the total flow rate that is allocated for that given solution such that the final metric value is independent of the solution's total network flow value. Note that this objective is non-linear because the flow's delay value is conditionally based on which paths the flow is currently using; thus, it cannot be used with an LP solver. This objective is normalised by dividing it with the cost of the path with the largest delay from the set of all paths.

3.5.2 Mutation

MOEA-II employs two methods of selecting the path set $\tilde{P}_i$ on which mutation will be performed, and are described next.

Minimise the Maximum Path Delay

This method attempts to minimise the probability of including paths with high delay values from the set of paths a flow is allowed to transmit on. For a given flow f_i, all paths with $\frac{\phi(p_{i,\min})}{\phi(p_{i,j})} \geq z$ are included in $\tilde{P}_i$, where $z \in \mathcal{U}(0,1)$. Paths with higher delay have a diminishing probability of being selected as this has a direct impact on the flow's end-to-end delay performance.

Maximise Flow

The objective of this method is to transmit as much of the flow's requested data rate as possible over all available paths; thus, $\tilde{P}_i = P_i$.

3.6 The role of SDN in the deployment of a globally optimised solution

Both of the LP and EA algorithms described here require up-to-date network information to be capable of generating a valid routing solution. The information

required includes the network topology, link properties, and all flows currently using the network. Only when all this information is available and accurate can the developed routing algorithms have all the necessary information to generate a routing solution. This information is what gives the routing algorithm the ability to optimise a solution at a global level. All of the routing solutions in [14–17] either have, or assume the existence of such network information. Similar to this thesis, the scope of the work in [14–17] is not in the deployment of such routing solutions, but rather the design of the routing algorithm. Although such works do not directly mention the use of SDN, as is the case here, an assumption is made whereby there exists a central router with access to accurate and up-to-date network information, which is essentially the key philosophy behind the design of SDN networks. The works presented in [7, 19] take it one step further and together with the design of a GOMR algorithm, highlight the steps taken to actually implement and deploy their given solution on an SDN network.

Having said this, there are deployment challenges to overcome when it comes time to deploy the designed routing solutions. The first hurdle to overcome, that is the result of having a centralised controller, is the additional delay used by the terminal or switch to communicate with the network controller before transmission has even started. The works in [7, 19] overcome this limitation by routing delay sensitive traffic using standard routing algorithms, such as OSPF, without interference from the controller. Another issue that arises from the central approach is the problem where if the controller fails, the whole network goes down. Having a single point of failure is a major weakness that has to be overcome. Research is being carried out on how to solve this issue, with solutions already being proposed. A summary of such solutions can be found in [18]. Going over the deployment issues faced by SDN is beyond the scope of this work. The aim of this section has been to give the reader the peace of mind that the network architecture assumed to exist in this work is based on solid research with evidence given that such solutions have been deployed and proven to work on deployed networks.

3.7 Path Selection Algorithms

All of the routing algorithms proposed in this work rely on an external algorithm to supply them with a set of paths that each flow is allowed to use. The only requirement set by the routing algorithms on the path discovery methods is the supply of loop free paths. One of the objectives sought after by all of the routing algorithms used here is the minimisation of delay. Therefore, an obvious choice for a path selection algorithm is the k-Shortest Path (KSP). The KSP algorithm

returns the first k paths with the lowest aggregate delay value. The KSP algorithm used here is a variation on Yen's KSP algorithm [36], where all the paths with an equivalent cost to the k^{th} path are chosen at random such that a flow will always have at most k paths. This method is used so as to have a fixed limit on the maximum number of paths a flow is allowed to use. Having control over the number of paths is important as it has a direct effect on the routing algorithm's complexity performance. One shortcoming of the KSP algorithm is the lack of link diversity when used on a highly inter-connected network topology, as most of the selected paths will share a large number of links between them. A path set sharing a large number of links is limited in the amount of data that can be transmitted over that particular path set. This is because the amount of data transmitted on a single path, reduces the capacity of all the paths that share a common link, limiting the data rate that can be transmitted over the entire path set. A path set sharing a large number of links is also problematic at the global routing solution level, not just at the flow level described previously. The reason being that if a given link, that is used by the majority of the paths in a given flow's path set, is used to near capacity by other flows, the routing algorithm does not have any alternative paths where to route the given flow. In such cases, the routing algorithm can decide to either share the link capacity between flows, or allocate data rate to only a single flow. To increase the link diversity between a given path set and overcome the said problem, the k-Shortest Edge Disjoint Path (KSEDP) [37] algorithm was considered.

Contrary to the KSP algorithm, the paths returned by the KSEDP algorithm do not share any edges. Paths using the same nodes are allowed. The KSEDP algorithm may be too restrictive in situations where a node has a single connection to another node; therefore, we opted to use a relaxed version of the KSEDP termed the k-Shortest Relaxed Edge Disjoint Path (KSREDP) algorithm. The difference between the KSEDP algorithm and the KSREDP algorithm developed here is that the KSREDP algorithm allows initial path segments that are the only means of communication between a source and destination pair to be shared by multiple paths. To determine which set of links to freeze, i.e. can be used by multiple paths, the shortest path is traversed from the source to the destination. If the node under consideration has a single outgoing link, the link is frozen and traversal continues to the next node. If the node under consideration has multiple outgoing links, traversal stops. The same procedure is repeated starting from the destination to the source node but considering incoming links instead of outgoing. Once the set of frozen links has been found, the KSP algorithm is run with $k = 1$ to find the shortest path between a source and destination. All the non-frozen links that make

up the found path are marked as used and will be ignored by the path selection algorithm. The KSP algorithm is run again with $k = 1$ to find the next relaxed edge disjoint path, and the non-frozen links marked as used. The above steps are repeated until either k paths are found, or no other paths that connect the source to the destination exist. Similar to the KSP algorithm, all the paths with an equivalent cost to the k^{th} path are chosen at random such that a flow will always have at most k paths.

The implementation of the KSP and KSREDP path selection mechanisms described in this section are based on the algorithm developed by Szcześniak [38]. All of the path selection algorithms have their metric set equal to the link's delay value. In other words, the shortest path is equal to the path with the lowest aggregate delay value.

3.8 Complexity Analysis

3.8.1 MOEA

This section considers the time complexity of the MOEA-I and II algorithms using big O notation [39]. The complexity analysis of functions that are used by other functions, such as the excess removal, link constraint, and flow constraint functions are analysed first. Let ϵ represent the number of links in a given graph, where $e_i \in E$ represent the i^{th} link. Let $\mathbf{P} = \{P_1, P_2, \ldots, P_n\}$ represent the set of all paths used by all the flows and θ represent the total number of paths as given by

$$\theta = \sum_{i=1}^{n} |P_i|, \tag{3.28}$$

where $|P_i| = k_i$ represents the cardinality of the set P_i. Note that θ is equivalent to the length of the chromosome.

Excess Removal

The complexity of the excess removal function described in Section 3.3.6 is linear with the number of values the algorithm needs to remove the excess from.

Flow Over-Provision Constraint

The flow over-provision constraint function ensures that no flow is allocated more data rate than what it had requested. To verify that no flow in a chromosome breaks this constraint, the algorithm is required to go over and check each flow separately. The chromosome is designed to store that data rate at a path level; therefore,

calculating the allocated data rate for a given flow requires $O(k)$ operations, where k is the maximum number of paths a flow is allowed to use. Removing excess data rate using the excess removal algorithm analysed in Section 3.8.1 requires an additional operation of $O(k)$, bringing the total complexity to $O(2k)$. Assuming the worst case where each flow has excess data rate, the total complexity for the flow over-provision constraint becomes equal to $O(2kn)$, where n is the number of flows. Dropping constants, the complexity is reduced to $O(nk)$.

Link Over-Provision Constraint

The link over-provision constraint verifies that no link is used beyond its capacity. To calculate the load passing on each link the network connectivity matrix is used. The network connectivity matrix is a binary matrix used to represent the network topology in terms of which links are used by each path. The Network Connectivity Matrix ($\mathbf{M}$) is given by (3.29), where cell $m_{i,j}$ is set to 1 if link $e_i \in E$ is used by path $p_j \in \mathbf{P}$, 0 otherwise.

$$\mathbf{M} = \begin{bmatrix} m_{1,1} & m_{2,1} & \cdots & m_{\epsilon,1} \\ m_{1,2} & m_{2,2} & \cdots & m_{\epsilon,2} \\ \vdots & \vdots & \ddots & \vdots \\ m_{1,\theta} & m_{2,\theta} & \cdots & m_{\epsilon,\theta} \end{bmatrix} \tag{3.29}$$

Calculating the load on each link involves multiplying the chromosome by the network connectivity matrix as shown in (3.30). The Hadamard product, represented by the $\circ$ operator, of the two matrices as given by (3.30) updates the network connectivity matrix to contain actual data rate values as given by a chromosome solution, referred to as the Actual Network Matrix ($\mathbf{A}$).

$$\mathbf{A} = \underbrace{\begin{bmatrix} g_{1,1} & g_{1,1} & \cdots & g_{1,1} \\ g_{1,2} & g_{1,2} & \cdots & g_{1,2} \\ \vdots & \vdots & \ddots & \vdots \\ g_{n,k_n} & g_{n,k_n} & \cdots & g_{n,k_n} \end{bmatrix}}_{\epsilon \text{ columns}} \circ \mathbf{M} = \begin{bmatrix} m_{1,1}g_{1,1} & m_{2,1}g_{1,1} & \cdots & m_{\epsilon,1}g_{1,1} \\ m_{1,2}g_{1,2} & m_{2,2}g_{1,2} & \cdots & m_{\epsilon,2}g_{1,2} \\ \vdots & \vdots & \ddots & \vdots \\ m_{1,\theta}g_{n,k_n} & m_{2,\theta}g_{n,k_n} & \cdots & m_{\epsilon,\theta}g_{n,k_n} \end{bmatrix} \tag{3.30}$$

The complexity to carry out said matrix operation is equal to $O(\theta\epsilon)$. Calculating the data rate carried over a given link becomes a matter of adding the link's respective column at a complexity of $O(\theta)$.

The link over-provision constraint function starts by generating the $\mathbf{A}$ matrix at $O(\theta\epsilon)$. Each link is checked until one that exceeds capacity is found. Assuming the worst case, this requires $O(\epsilon)$ iterations to go over each link, $O(\theta)$ iterations

to calculate the data rate passing through that link and $O(\theta)$ to fix it using the algorithm analysed in Section 3.8.1. Note that once a link has been repaired, this process is restarted, with matrix $\mathbf{A}$ having to be recalculated as the repair operation may affect other links. Assuming the worst possible case where all the links have to be repaired, the total complexity of the algorithm becomes equal to $O(\epsilon(\theta\epsilon+\theta\epsilon+\theta))$. If we reduce it to its most dominant function and drop constants, the complexity becomes $O(\theta\epsilon^2)$.

Initial Population Generation

Each chromosome is generated by assigning data on each path for every flow present. This requires a loop over all the paths for all the flows, resulting in a complexity of $O(\theta)$. Once a chromosome has been generated, it needs to be checked to make sure that it does not violate the flow or link capacity constraint. The complexity to generate an entire population at random is $O(\mathcal{P}(\theta+\theta\epsilon^2+nk))$, where $\mathcal{P}$ is the population size. If we reduce it to its most dominant function, we are left with $O(\mathcal{P}\theta\epsilon^2+\mathcal{P}nk)$.

Crossover

The *Crossover* function takes two chromosomes and swaps their flow's entry at random to generate two new offspring chromosomes. In this case, the worst case scenario assumes that all flow entries are swapped with each other. A swap operation is assumed to be of constant time $O(1)$ and is therefore dropped with the constants. This crossover does not alter the flow data rate assignment; therefore, chromosomes generated by this operation only need to be checked for link constraint violation. The complexity of the crossover operator is equal to $O(n+\theta\epsilon^2)$.

Mutation

In contrast with the crossover function, the mutation operator only works on a fraction μ of the flows on a single chromosome. The mutation operation can be further split into two sub-functions: the path filter function and the flow data rate assignment. Various different path filter functions are designed in this work, since each function works by filtering a flow's path set, their complexity is at most $O(k)$. The flow data rate assignment assigns as much data as possible on the path set returned from the path filter function without exceeding any of the link capacities. To calculate the current link capacity usage, the $\mathbf{A}$ matrix is generated at a complexity of $O(\theta\epsilon)$ as explained in Section 3.8.1. Data rate is then assigned to the k paths of the mutated flow in random order. The complexity of the data rate allocation function is equal to $O(\theta\epsilon+k)$. This brings the total complexity of

Table 3.1: Time complexity of each objective

Objective	Complexity
Total Network Flow ($\mathcal{O}_1$)	$O(\theta)$
Proportion of Flows with Minimum Delay ($\mathcal{O}_2$)	$O(nk)$
Flow Splits ($\mathcal{O}_3$)	$O(\theta)$
Estimated Mean End-To-End Delay ($\mathcal{O}_4$)	$O(\theta)$

the mutation operator equal to $O(\mu n(\theta\epsilon + k))$, assuming constants are dropped. Due to the link capacity consideration included in the mutation operator and the fact that at most the flow's requested data rate is assigned, chromosomes that have been mutated do not require any further validation checks.

Chromosome Fitness Evaluation

Before the NSGA-II algorithm can choose which solutions to keep and which solutions to remove, each newly generated chromosome must be evaluated based on the objectives. An explanation of how the complexity of each objective is determined is given next, with the results summarised in Table 3.1.

Total Network Flow is simply the summation of the chromosome values $O(\theta)$.

Proportion of Flows with Minimum Delay needs to find the path with the lowest delay value from the flow's path set and then calculate the metric for that flow. Calculating the metric for the entire chromosome comes at $O(nk)$ with constants dropped.

Flow Splits counts the number of flows that are using multiple paths, as well as the total number of paths used by a flow. This can be accomplished by looping over the chromosome once at the cost of $O(\theta)$.

Estimated Mean End-To-End Delay is rather simple in concept as it chooses the path with the largest delay value from the set of paths being used by a flow. This comes at a complexity of θ as one pass over the entire chromosome is enough. Since this metric value is normalised by the network flow, an additional $O(\theta)$ is required for the total network flow calculation. This bring the complexity equal to $O(2\theta)$, which becomes equal to $O(\theta)$ with constants dropped.

NSGA-II Selection

The NSGA-II selection algorithm has a complexity of $O(M\mathcal{P}^2)$, where M is the number of objectives and $\mathcal{P}$ is the population size, respectively [30].

Complexity of the Entire Algorithm

The complexity for the *TournamentSelection* function is set to be equal to $O(\mathcal{P})$, since the sorting of the population is included in the NSGA-II algorithm described in Section 3.8.1. Recall that χ represents the number of generations, ω the crossover probability, and ψ the mutation probability. In the below expressions, the order in which functions appear is the same as that given in Algorithm 1 in Section 3.3. The full complexity for MOEA-I is given by

$$O(\mathcal{P}(\theta\epsilon^2+nk)+\chi(\mathcal{P}+\omega(n+\theta\epsilon^2)+\psi(\mu n(\theta\epsilon+k))+\mathcal{P}\theta+\mathcal{P}\theta+\mathcal{P}nk+3\mathcal{P}^2)). \quad (3.31)$$

Assuming the worst case scenario where $\omega = \psi = \mu = 1$ and dropping constants brings the complexity equal to

$$O(\mathcal{P}(\theta\epsilon^2 + nk) + \chi(\theta\epsilon^2 + n\theta\epsilon + \mathcal{P}\theta + \mathcal{P}nk + \mathcal{P}^2)). \quad (3.32)$$

Using the same procedure, the full complexity for MOEA-II is equal to

$$O(\mathcal{P}(\theta\epsilon^2 + nk) + \chi(\theta\epsilon^2 + n\theta\epsilon + nk + \mathcal{P}\theta + \mathcal{P}^2)). \quad (3.33)$$

From the above analysis, the developed EA that uses the NSGA-II algorithm scales quadratically with the population size and the number of links in the topology and linearly with the total number of paths. Note that no time has been dedicated to the optimisation of the algorithms used by the EA such as the flow and link constraint functions, meaning that there may exist other, more efficient implementations that carry out the same job.

3.8.2 LP

The GNU Linear Programming Kit (GLPK) library does not provide information on the complexity and scalability of the algorithms used to solve LP formulations. Therefore, empirical evidence is used instead to determine the scalability of the developed LP routing algorithm that solves the PC-MFMC problem. Figure 3.3 shows the time taken by the LP solver to find a solution to the PC-MFMC problem as the number of variables increases, grouped by the network load. Using curve fitting, it is clear that in general, the LP solver used in this work scales quadratically with the number of variables. Note that there are particular instances where the time required to find a solution is more than double the time required by solutions with the same number of variables. The reason behind these outlier values needs to be investigated further. To the best of the author's knowledge, the most recent work on how to solve LP problems efficiently is the one by Cohen et al. [40], which

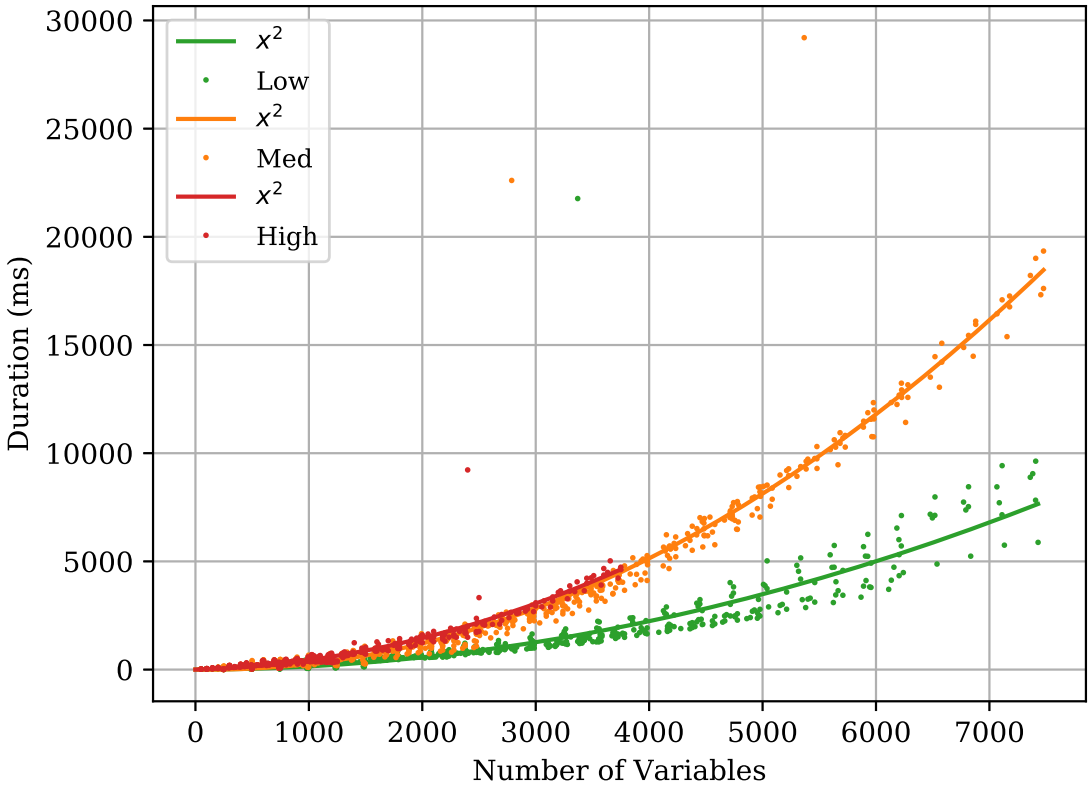


Figure 3.3: Time taken for the LP algorithm to find a solution to the PC-MFMC problem as the number of variables is increased, grouped by the network load. x^2 plot refers to the degree 2 polynomial generated when fitting a curve to the generated data points.

still scales quadratically with the number of variables.

3.8.3 Complexity comparison: MOEA vs LP

The LP's complexity scales quadratically with the number of variables. In the case of the PC-MFMC problem, the number of variables is equivalent to the total number of paths in a given solution. The number of paths in a given solution is equivalent to the chromosome size (θ). On the other hand, the developed EA scales quadratically with the number of links in a topology and the population size chosen. However, assuming the EA is used over a fixed topology and constant population size, the EA scales linearly with the chromosome size which is better than LP. Keep in mind that the MOEA-II algorithm is using non-linear objectives which if used by the LP solver would drastically increase it's complexity.

Chapter 4

Protocol Design and Implementation Issues

This chapter highlights the necessary work required in order to generate and simulate the routing solutions of both the EA and LP routing algorithms described in Chapter 3. First, the two different methods used to implement *Per-Packet* multipath are given. A description of the LP solver and the EA framework used are given next. A summary of the available network simulation/emulation frameworks and the reasons behind the selection of the Ns3 simulation framework is given next. This chapter concludes by providing detailed information on how the network simulations are set up to enable easy replication of the results presented in this thesis.

4.1 Per-Packet Multipath

Splitting a flow over multiple paths can be accomplished either at the source, or by the network switch. The advantage of splitting a flow at the switch level is that no additional protocol headers will be required; however, such method is known to negatively interfere with TCPs performance and requires modification to the underlying SDN hardware. TCP is a transport layer protocol designed for stream oriented reliable communication between two devices over a network. TCP has been designed with the assumption that packets follow the same path. Because of this assumption, splitting a TCP stream over multiple paths negatively affects the stream's performance. The reason being that when using multiple paths for transmission, packets may be received out of order due to the different properties of each path. In such a case, TCP mistakenly treats this out of order reception as a sign of network congestion and reduces the transmission rate. More information

on this topic can be found in [41, 42] with simulation results presented in [12]. Alternatively, splitting the flow at the source, with the help of the modified MPTCP transport layer protocol, does not require any modifications to the SDN hardware and allows TCP applications to benefit from the increased performance *Per-Packet* multipath is able to offer. However, using the MPTCP will have a marginal impact on the transmission efficiency due to the additional MPTCP header required to be appended to each packet before transmission.

Section 4.1.1 explains the novel Per-Packet Flow Splitting (PPFS) method developed in this work that outperforms the current OpenFlow [43] method of flow splitting in terms of splitting accuracy and scalability. Section 4.1.2 explains the required modifications for MPTCP to work seamlessly with the developed routing algorithms.

4.1.1 Split at Switch (PPFS)

OpenFlow [43], one of the most widely used SDN southbound interface protocol, implements flow splitting with the use of groups and hash-based splitting [44]. An SDN southbound protocol is the protocol used for communication between the network controller and the SDN switches. Groups are sufficient for coarse, equal traffic splitting but will quickly run into scalability issues when faced with unequal, fine grained traffic split ratios. OpenFlow switches support unequal flow splitting by means of hash-based splitting [44, 45]. To achieve the desired flow split ratios while using hash-based splitting, entries are added in the routing table until the split ratio is met. For example, transmitting 90% of a flow's packet on one path, and the remaining 10% on another requires a total of ten entries. It is clear that this method does not scale well with respect to the number of entries required as the split ratios become more intricate. Because of these limitations, Tuncer et al. [45] developed a traffic splitting mechanism based on IP addresses. In their work, Tuncer et al. [45] developed a system using IP masks to group a range of IP addresses together in order to enable unequal *Per-Flow* multipath load balancing. However, the split ratio accuracy of their developed system depends on the IP address distribution. The routing algorithms developed in this work have no constraints on the split ratio granularity; therefore, a system that enables the deployment of a *Per-Packet* multipath solution while remaining scalable is required to test out the routing solutions provided by the routing algorithms. The developed, stochastic based flow splitting algorithm residing on the network switch is explained next [5].

Table 4.1: Partial routing table with split flow

Entry Number	Flow	Cumulative Split Ratio	Port Number
0	Flow A	0.3	1
		0.45	2
		1	3
1	Flow B	0.1	1
		1	3

PPFS Algorithm

Additional information is required to be stored with each routing table entry at the SDN switch to add the required information to achieve the desired flow splitting ratio instructed by the routing algorithm. For each flow entry installed on the switch, a list of the output port numbers and their respective cumulative split ratios is appended. Upon packet reception, the switch will match the flow using the packet's headers, generate a uniform random number in the range $[0-1)$, and select which port to forward this packet on based on this number. The forwarding port is selected such that the corresponding cumulative split ratio is the first value greater than the generated random number.

Consider as an example the partial routing table shown in Table 4.1, with two flows, A and B. In this example, the switch is set up to forward 30% of Flow A's packets through port 1, 15% of Flow A's packets through port 2, and the remaining 55% of Flow A's packets through port 3. Similarly, the switch needs to forward 10% of Flow B's packets through port 1 and the residual 90% over port 3. When the switch receives a packet from Flow A it generates a random number, for example 0.4, and forwards the packet based on this value; in this case, port 2.

Using the OpenFlow hash-based method of splitting a flow, the proposed PPFS method requires 80% fewer rules to implement the splitting required by Flow B, shown in Table 4.1. As the split ratios become more intricate, our technique scales much better than hash-based splitting as it depends only on the number of paths a flow is split to rather than the split ratio. For more details about unequal hash-based splitting, the reader is referred to [44].

In addition to the negative effect on TCP traffic when used in conjunction with a *Per-Packet* routing algorithm, the other disadvantage of the proposed PPFS system is the modifications required to both the OpenFlow protocol and switch firmware. An alternative method of deploying a *Per-Packet* multipath routing solution on a network without any modifications to the SDN hardware is by splitting the flow at

the source.

4.1.2 Split at Source (MPTCP)

The major barrier faced by the PPFS flow splitting method described in Section 4.1.1 is the performance penalty suffered by TCP flows. Since TCP is the main transport protocol used to date, a system that negatively effects TCP will not find much use in the real world. TCP is a stream oriented transport layer protocol designed for applications seeking a reliable connection between two devices over a computer network. TCP assumes that all packets travel over the same path and builds the congestion control algorithms based on this assumption. This central assumption is broken when a flow's packets are transmitted over multiple paths. Transmitting packets over different paths, with different properties, may lead to packets being received out of order. TCP mistakenly treats this as a sign of congestion and reduces the transmission rate.

To pave the way for deploying the routing algorithms developed here, the network performance issues faced by TCP must be resolved first. This problem is overcome by using a modified version of the MPTCP protocol. An overview of the MPTCP protocol, and the methods employed in literature to overcome problems similar to the ones faced here are given next, that will serve as background information to the MPTCP protocol modifications proposed in this work.

MPTCP Protocol Description

MPTCP [46] is a transport layer protocol that aggregates multiple TCP sub-flows to improve the flow's data rate and/or reliability. MPTCP being a transport layer protocol, does not have the ability to control the path taken by each created TCP sub-flow. Therefore, MPTCP has been originally targeted, and found its first major practical use case in multi-homed devices. Apple first deployed MPTCP with iOS 7 to increase the reliability of the Siri voice assistant [47]. In this case, MPTCP is used to create two connections, one over WiFi and another over LTE for a seamless handover in the event a user loses WiFi connection. The lack of path selection knowledge requires MPTCP to implement a shared congestion control mechanism between all the TCP sub-flows such that multiple MPTCP sub-flows do not starve a single TCP connection from resources if they happen to share a bottleneck link [46].

The availability of SDN allows the routing algorithm to gain the intelligence required to distinguish between different MPTCP sub-flows and thus avoid routing them over the same path. Zannettou et al. [48] does just this by exploiting SDN to

route MPTCP sub-flows over different paths. However, previous to the work done by Zannettou et al. in [48], the Linux kernel implementation of MPTCP is only able to create one sub-flow for a pair of IP addresses. This limitation has been addressed and fixed by Zannettou et al. [48] and added to version 0.9 of the Linux kernel MPTCP implementation. This change allows MPTCP to open more than one sub-flow for a pair of IP addresses. The scheduler that ships as standard with MPTCP is the *minRtt* scheduler. As the name suggests, the *minRtt* scheduler transmits all the packets on the path with the lowest Round Trip Time (RTT). Such a scheduler does not work well with the routing algorithms developed here, as the routing solution dictates the amount of data rate to transmit on each path. Therefore, a new stochastic scheduler has to be added to MPTCP.

MPTCP Proposed Modifications

All routing algorithms used in this work are globally optimal because they take into consideration all of the flows using the network when generating a solution. Because the routing algorithms are assumed to be globally optimal, multiple MPTCP sub-flows are never routed over the same path, and congestion caused by over using links is also handled by the routing algorithm; therefore, the shared congestion control provided by MPTCP is unnecessary and is ignored. A custom, stochastic scheduler is added to MPTCP to distribute a flow's packets based on the information provided by the routing algorithm. Any congestion that may arise due to the dynamic nature of a computer network and the stochastic nature of the scheduler are handled by the underlying TCP sub-flow congestion control mechanism.

The proposed MPTCP framework model is shown in Figure 4.1, which outlines the steps taken in sequence by a flow before starting data transmission over the network. The MPTCP protocol sits in between the TCP and the application layer, and is assumed to have direct communication with the network controller to exchange information with the routing algorithm. On receiving a flow transmission request from an application, the routing algorithm generates a routing solution and informs the MPTCP layer on the number of connections to open and the data rate to transmit on each one. Next, the network controller and MPTCP protocol negotiate the port numbers to use for each connection. This port number negotiation is required as it allows switches to identify which path a packet must follow. In other words, there is a one-to-one relationship between a path and the port numbers used. The distribution of packets between the different paths is handled using a stochastic scheduler, similar to that used by the PPFS algorithm. The stochastic scheduler distributes packets across the different paths based on the data rates the routing algorithm allocates on each path. A stochastic scheduler is

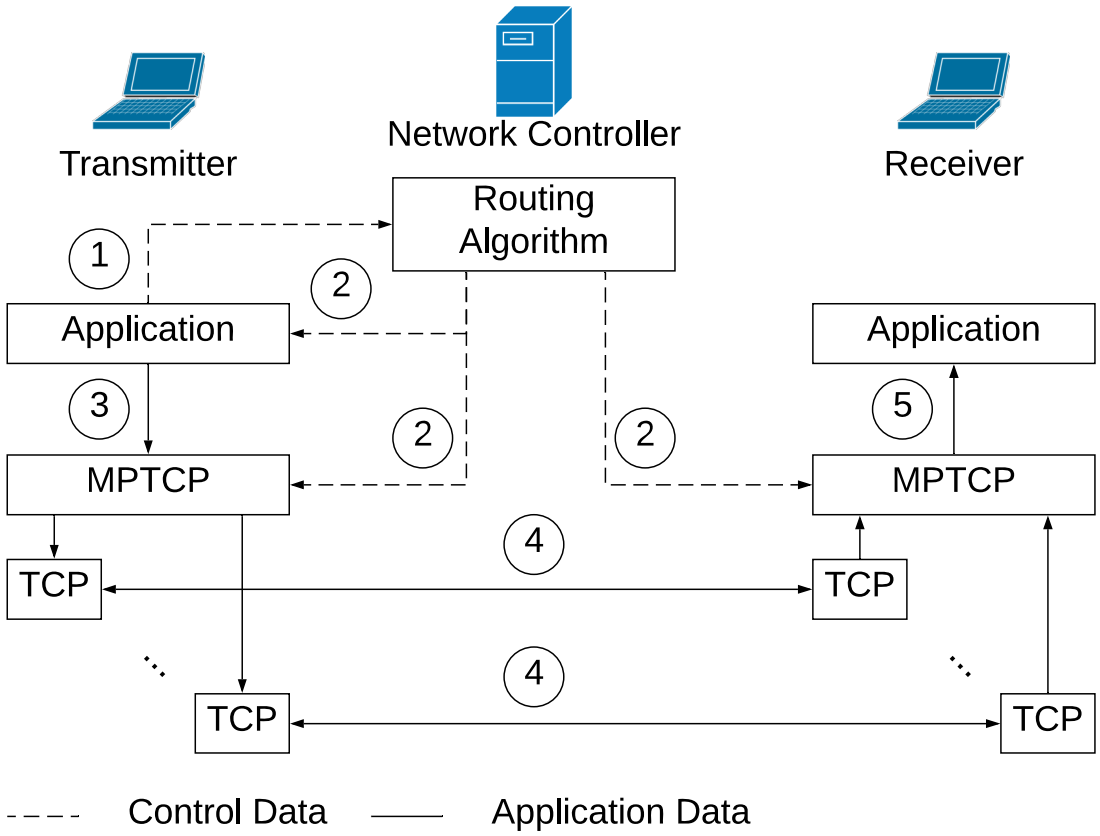


Figure 4.1: The proposed MPTCP framework. The numbers represents the sequence of events, in order, when an application has data to transmit.[12]

used because of its implementation simplicity and ability to handle any arbitrary split ratio without running into scalability issues.

To summarise, the only change done to the MPTCP protocol is the addition of a stochastic scheduler that is capable of distributing packets based on the distribution set by the routing algorithm. The 64-bit packet identifier appended by MPTCP to every transmitted packet is kept as this is required to deliver packets to the application in the same order as that transmitted. The ordered delivery of packets is one of the promises made by TCP to the application, which is a promise that MPTCP needs to honour as well. In the text that follows, MPTCP refers to the MPTCP with the modification proposed in this section.

4.2 Linear Programming Solver

The solutions to the LP formulations are found using the GLPK [49] library accessed through the LEMON's [50] library interface. LEMON's GLPK interface has been updated to run the `glp_exact` function after running the `glp_simplex` function to improve numerical stability. The use of the `glp_exact` is required as otherwise, flows may be allocated very small negative data rates, even though a constraint is set where solutions are only allowed to use positive numbers. The `glp_exact` function is similar to `glp_simplex` but uses exact arithmetic. Since computations that use exact arithmetic are very time consuming, the GLPK manual suggests to first run the `glp_simplex` function to find an optimal basis and then call the `glp_exact` function. Following this procedure reduces the number of simplex iterations that have to be carried out using exact arithmetic, improving the running time of the algorithm.

4.3 Evolutionary Algorithm

The MOEA designed here is implemented using the Distributed Evolutionary Algorithms in Python (DEAP) v1.3 [51] library. The DEAP library has been chosen due to its wide use in the EA community and the use of the Python programming language. Preference is made to frameworks using the Python programming language due to its ease of development, which minimises the time required to develop and test out a solution when compared to other languages, such as C++. This comes at the cost of the algorithm's running time, as Python is known to be much slower than its compiled alternatives, such as C++. This is a known limitation of this work, with steps required to overcome such a problem highlighted in Section 6.1.

4.4 Network Simulator

To test out the performance of the developed routing algorithms, the routing solutions generated by such algorithms have to be tested on a network to gauge their actual performance. Ideally, the proposed systems are tested on an actual network; however, financial and time limitations made such an option unfeasible. Therefore, we opted to choose the second best option and verify the validity of the routing solutions using network simulations. When choosing a network simulator the following criteria were set:

- Be under active development. This is required as the simulation environment would be constantly updated with new protocols and bug fixes to maintain stability and accuracy in the provided results.
- Have good, detailed documentation.
- Flexible enough to allow the development of custom protocols or devices.
- Ideally available free of charge and open source. Open source is important as it removes any restriction set by the vendor and gives the user the freedom to modify any part of the simulation framework as he sees fit.
- Run on the Linux Operating System, as it is the operating system used by the servers where simulations will be run on.

Three network simulators met all of the requirements set out above: Mininet [52], OMNeT++ [53], and Ns3 [54]. Mininet [52] is a network emulator that is the tool most often used to produce results with works dealing with SDN. Owing to the fact that Mininet is an emulator, the size and scale of the network that it can handle is limited by the underlying hardware. Additionally, the timing results provided by an emulator are not as accurate as those provided by a simulator as they are affected by the other processes running on the same machine. The main advantage of an emulator is the accurate model of the hardware being emulated which makes it easier to transition from a virtual to an actual network. A more thorough description of Mininet's performance limitation can be found in [55]. Both OMNeT++ and Ns3 are discrete event simulators, meaning that they rely on a simulation generated clock leading to much more accurate timing results. Since both OMNeT++ and Ns3 are simulators, Ns3 is used in this thesis due to its popularity with the networks community and the author's familiarity with the framework. Network simulations are carried out using the Network Simulator version 3.29 (Ns3).

Custom devices are developed to replicate the required functionality of an SDN

switch and the PPFS switch described in Section 4.1.1. All switches are assumed to have unlimited buffers to eliminate the effect of packet loss caused by buffer overflow. Although this is unrealistic, this assumption simplifies the analysis of the network performance results. The random numbers required by the MPTCP module and the PPFS switches are generated using Ns3's own Uniform Random Number generator. All flows are assumed to transmit at a Constant Bit Rate (CBR) with a data packet size of 590 bytes including all the necessary headers with each TCP acknowledgement packet being 54 bytes long. Methods of how to shape bursty traffic to have a profile similar to a CBR exist, with Szymanski [35] presenting one such method. Using the above packet sizes, and the assumption that TCP transmits an acknowledgement packet for every two data packets received [56], TCP's acknowledgement rate, $\alpha(g_{i,j})$, is given by

$$\alpha(g_{i,j}) = 0.0458 \times g_{i,j}. \tag{4.1}$$

The *NewReno* TCP congestion control mechanism is used [57]. With the exception of OSPF, applications transmit at the rate assigned by the routing algorithm, not that requested. Data rate transmission modification is possible as SDN allows for the bi-directional communication between the routing algorithm hosted on the network controller and the application. OSPF lacks such functionality; thus, OSPF results are generated by setting the flows to transmit at their requested data rate. Two examples of rate adapting applications are file transfer and video streaming applications. For each TCP connection/sub-flow created by the MPTCP protocol, the TCP transmit and receive buffer size is automatically adjusted such that it is large enough to support transmitting at the data rate assigned by the routing algorithm on that given path. The buffer size in bytes is calculated using the bandwidth delay product [56] as given by

$$\text{Buffer Size} = \min\left(\frac{g_{i,j} \times \text{RTT}}{8}, 4096\right), \tag{4.2}$$

where RTT is given in seconds and $g_{i,j}$ is in bits per second. A minimum buffer size of 4096 bytes is set to match the value used by the TCP implementation in the Linux kernel. Under all scenarios presented here, the routing tables are populated before packet transmission starts, eliminating the routing protocol overhead.

Due to the lack of a proper, Ns3 native, MPTCP protocol implementation, the TCP sub-flow generator, and scheduler were developed as these two blocks are enough to test the performance of the updated MPTCP protocol. The TCP sub-flow generator is the module that creates a number of TCP sessions, where the number of sessions to open, and which port numbers to use on each session is specified by the

routing algorithm. The developed MPTCP stochastic scheduler distributes packets between the different TCP sessions (each TCP session is equivalent to a path) using a method similar to that used by the PPFS algorithm, where the split ratios are given by the routing algorithm. The MPTCP shared congestion control mechanism is not implemented because the use of a globally optimal routing algorithm makes this congestion control mechanism redundant. We do not see any reason why the shared congestion control used by MPTCP should negatively impact performance, with time being the only reason it was not implemented. Ns3 has the ability to use protocol algorithms found on a machine's kernel; however, modifying the MPTCP kernel implementation has been deemed too time consuming for this project, since simpler alternatives were available. To reduce development time further, MPTCP receiver applications are assumed to have infinite receiver buffers to avoid the need to implement MPTCP's acknowledgement mechanism to recover from packet losses caused by receiver buffer overflow. Note that packets lost during transmission are handled by the underlying TCP sub-flows and are handled by our developed MPTCP model.

Chapter 5

Results

This chapter presents the results of the algorithms designed in this work and draws conclusions based on the presented results. The routing algorithms developed here are compared with each other, highlighting the advantages and disadvantages of each. The generated routing solutions are validated using network simulations lasting 120 simulation time seconds. Choosing EA parameters, such as crossover probability, population size and number of generations is highly subjective and depends on the problem at hand. Therefore, an in-depth explanation of the decisions taken that led to the chosen EA parameters is given. For ease of replication, detailed information on the setup used in this work is given at the beginning of the chapter. Additionally, all of the source code and results will be made publicly accessible at the time of publishing.

5.1 Setup

5.1.1 Network Topology

A model based on the 2017 GÉANT network topology shown in Figure 5.1 is used to test out the performance of the developed routing algorithms. The 2017 GÉANT network topology is chosen as it models an actual network topology and has multiple paths available between a given source and destination node pair, especially in the core of the network. Such a feature is important to this work as it allows us to test the multipath capability of our routing algorithms. The actual link capacities have been scaled down from Gbps to Mbps to allow us to simulate conditions that require the use of multipath routing with a reasonable number of flows. Instead of reducing the link capacities to Mbps, we could have used flows transmitting at Gpbs levels; however, the standard TCP that ships with

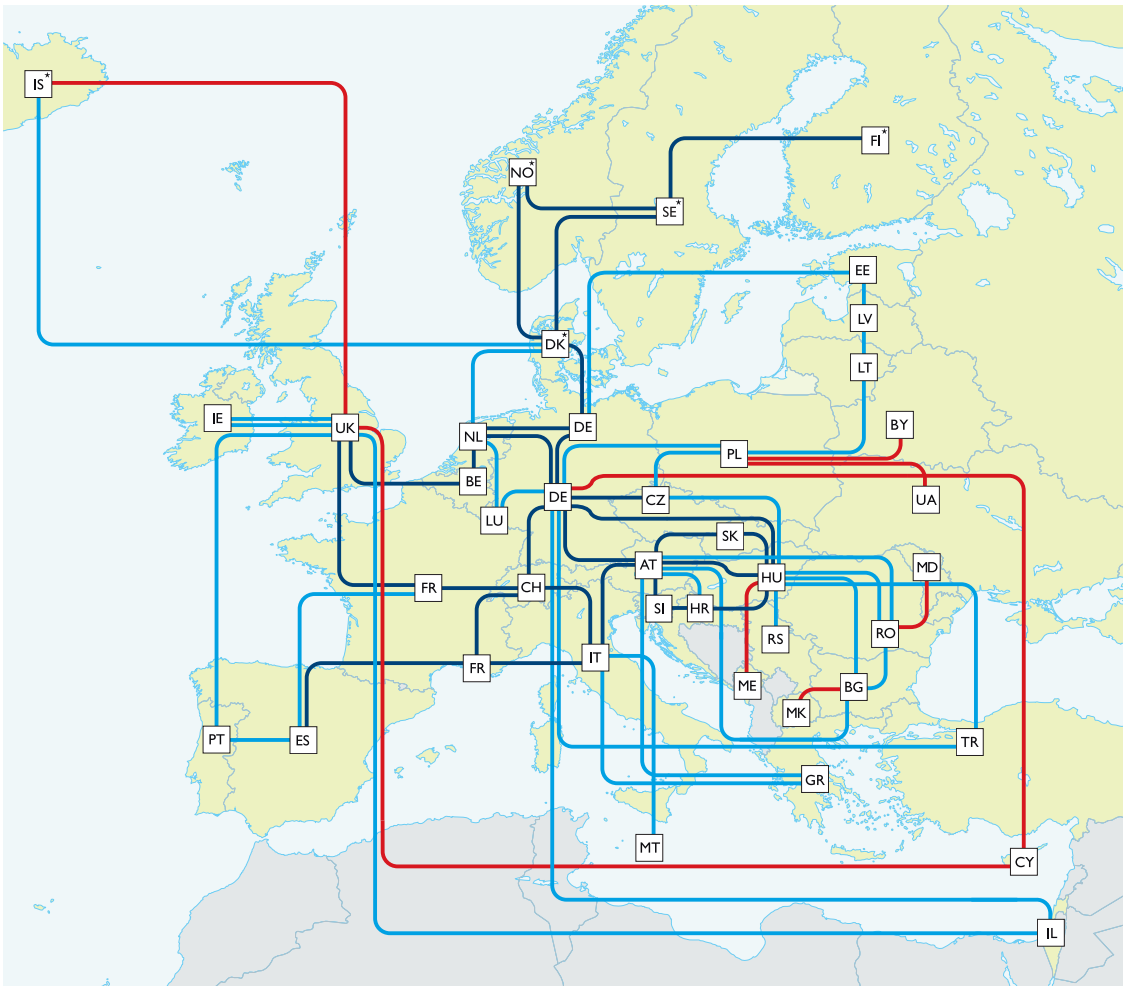


Figure 5.1: 2017 GÉANT network topology, adapted from [58]. Copyright 2017 GÉANT, all rights reserved. Used with permission. The modified capacity of the red, light blue, and dark blue links are equal to 30 Mbps, 60 Mbps, and 120 Mbps, respectively. The actual capacity of the red, light blue, and dark blue links are equal to 1–9 Gbps, multiples of 10 Gbps, and multiples of 100 Gbps, respectively [58].

Table 5.1: Network Load Configuration

Network Load	Data Rate (Mbps)	
	Mean	Std. Deviation
Low	5	0.25
High	25	2.5

Ns3 is unable to reach gigabit speeds without modification. Therefore; instead of modifying TCP to reach gigabit speeds, we took the decision to scale down the network link capacities and scale the flow data rates accordingly. The modified, scaled down, link capacities are equal to 30 Mbps, 60 Mbps, and 120 Mbps for the red, light blue, and dark blue links, respectively. The actual delay values of the GÉANT network topology were not readily available at the time of development; therefore, the delay attribute of each link is set in proportion to the geographical distance between the cities where the GÉANT Points of Presence corresponding to the two nodes are located [59]. The link with the shortest physical distance is set to have an arbitrary delay value of 1 ms. This value is used as a reference when calculating the delay values for the remaining links. To ensure repeatability, the link delay values used in this work are given in Appendix A. Symmetrical, bi-directional links are assumed in this work. Figure 5.1 only shows the network switches and their connections. When setting up network simulations, terminals are created and attached to switches based on whether that switch is a flow's source or destination node. In the case where multiple flows start or end at the same switch, only one terminal is created. The links connecting the switches with the terminals are set to have a capacity of 10 000 Mbps and a delay of 1 ms. An absurdly high capacity is given to such links to ensure that they are never the cause of network congestion as this would skew the network simulation results.

5.1.2 Flow Setup

Three network loads are used in this work: Low, Medium and High. With the exception of the high network load, that goes up to 150 flows, the number of flows ranges from 50 to 300 in steps of 50. Five flow sets are generated for each network load. The flow data rate is generated using a normal distribution with the mean and standard deviation for the low and high network load setup given in Table 5.1. The flow data rate values for the low and high network load setup are made to represent High Definition (HD) and Ultra High Definition (UHD) video transmission, respectively. The medium load setup has an equal number of flows having a low and high network load profile. Under all scenarios considered here, the flow's source and destination nodes are selected randomly with the selection

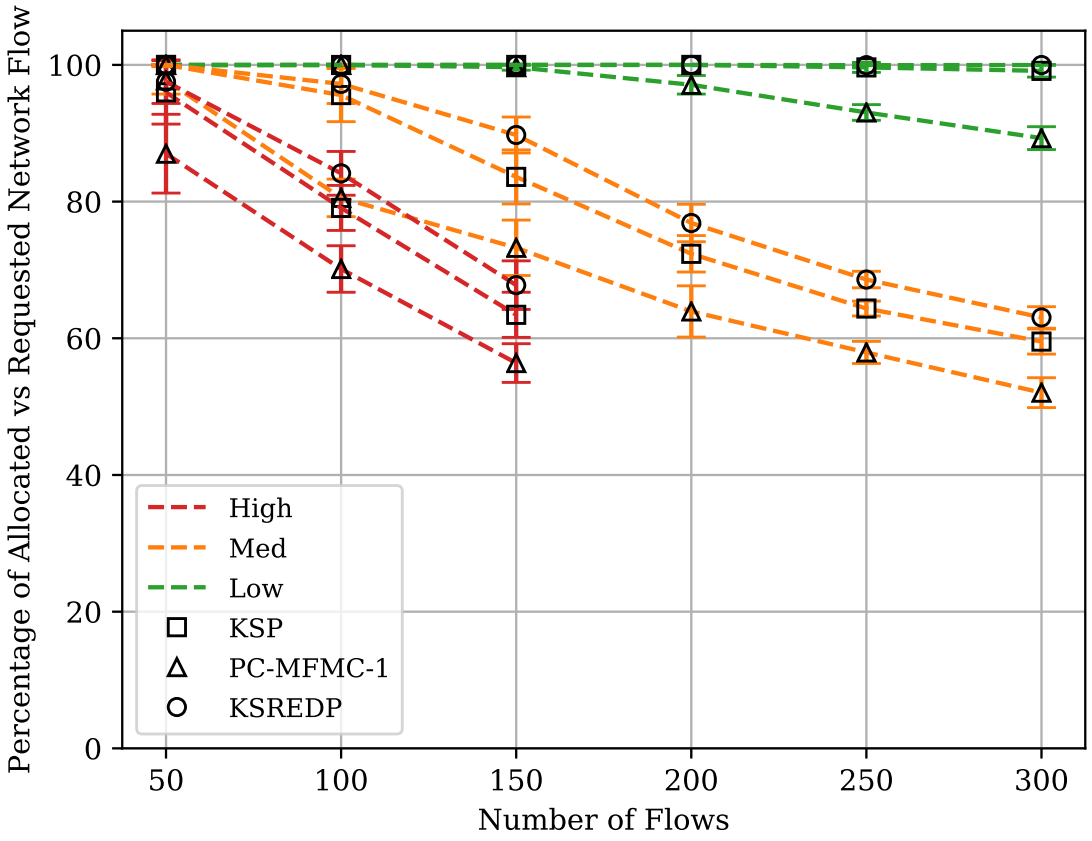


Figure 5.2: Plot illustrating the percentage of the total allocated network flow as a fraction of the total requested network flow for all the 5 flow sets. The total allocated network flow value is found by solving the PC-MFMC using LP and $k = 5$. PC-MFMC-1 represents the solution found by LP when only the shortest path is available and is used to represent the maximum attainable network flow by OSPF.

probability directly proportional to the node's total outgoing or incoming capacity, respectively. A flow is not allowed to have identical source and destination nodes.

The high network load scenario only goes up to 150 flows, because even at this stage, network capacity is already exceeded with only 70% of the total requested flow rate being allocated, as shown by Figure 5.2. Figure 5.2 shows the percentage of the total allocated network flow as a fraction of the total requested network flow. As shown by the results in Figure 5.2, increasing the number of flows to values higher than what is used here is unnecessary, as the network's capacity has already been exceeded. Increasing the network load any further would result in the LP solution to the PC-MFMC problem to contain a large number of unassigned flows. The PC-MFMC-1 results are generated by solving the PC-MFMC problem with $k = 1$. The PC-MFMC-1 results are included as they represent the highest attainable network flow that can be achieved if a single path routing algorithm, such as OSPF is used. The results presented in Figure 5.2 are generated with $k = 5$. The reason for choosing $k = 5$ is explained in Section 5.1.3. Only solutions to the

PC-MFMC problem using LP are shown in Figure 5.2 because of LP's optimality guarantee. Solving the PC-MFMC problem using LP guarantees that there exists no solution with higher allocated network flow than the solution returned by LP. Note that the above statement does not imply that the routing solution returned by the LP solver is the only solution capable of reaching the given network flow, but rather, that no solution exists with a higher allocated network flow.

In this work we simplify the problem by using a static flow set, meaning that no flows enter or exit the network for the duration of the simulation. This is a known limitation of this work, with the steps required to overcome such a limitation highlighted in Section 6.1. In the results that follow, *Goodput* is defined as the rate at which an application is able to generate or consume data. *Delay* is defined as the time taken from when the transmitting application sends a byte of data, to when the receiving application receives that same byte of data. Any time used waiting to deliver a block of data to the application in its correct order is included in the delay measurements. This setup is used to accurately represent the performance of an application using the protocols under test.

5.1.3 Path Setup

The EA chromosome is designed at the path level granularity, which requires a path selection algorithm to hand over the paths to the EA. Two different path selection algorithms are proposed here; the KSP and KSREDP, explained in Section 3.7. Both algorithms take the k value as a parameter, which represents the maximum number of paths a flow is allowed to take. The selection of the k value needs to strike a balance between the flow's path selection variety and the algorithm's complexity. Obviously, the more paths a flow is allowed to take, the more different network resources can be used which may lead to solutions with a higher network performance. On the other hand, increasing k has an effect on the algorithm's running time, as explained in the Complexity Analysis Section 3.8.1. Therefore, when selecting the k value, a compromise needs to be reached between the algorithm's running time and path variety. To take an informed decision on the k value, the unconstrained Maximum Flow problem is used with the optimal solution compared with the PC-MFMC problem with various k values. Let $S \in E$ and $T \in E$ represent all the incoming links to node s_i and all the outgoing links to node d_i where s_i and d_i are flow f_i's source and destination nodes, respectively. The formulation for the unconstrained Maximum Flow is defined as follows:

$$\max_{r^i_{u,v}} \sum_{i=1}^{n} \sum_{(u,v)\in E} r^i_{u,v}, \tag{5.1}$$

such that

$$\sum_{i=1}^{n} r_{u,v}^{i} \leq \lambda_{u,v} \quad \forall (u,v) \in E, \tag{5.2}$$

$$r_{u,v}^{i} \geq 0 \quad \forall (u,v) \in E, \quad i, \tag{5.3}$$

$$\sum_{v:(u,v)\in E} r_{u,v}^{i} - \sum_{v:(v,u)\in E} r_{v,u}^{i} = \begin{cases} \hat{\delta}_i & u = s_i \\ -\hat{\delta}_i & u = d_i \\ 0 & \text{otherwise} \end{cases} \quad \forall i, \quad u, \tag{5.4}$$

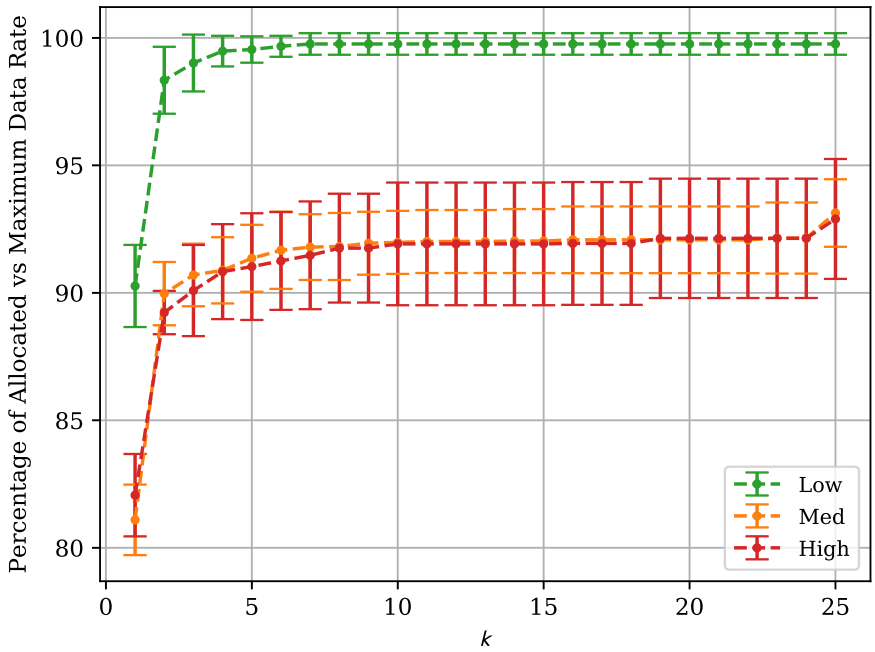
$$\sum_{(u,v)\in S} r_{u,v}^{i} = 0, \tag{5.5}$$

$$\sum_{(u,v)\in T} r_{u,v}^{i} = 0, \tag{5.6}$$

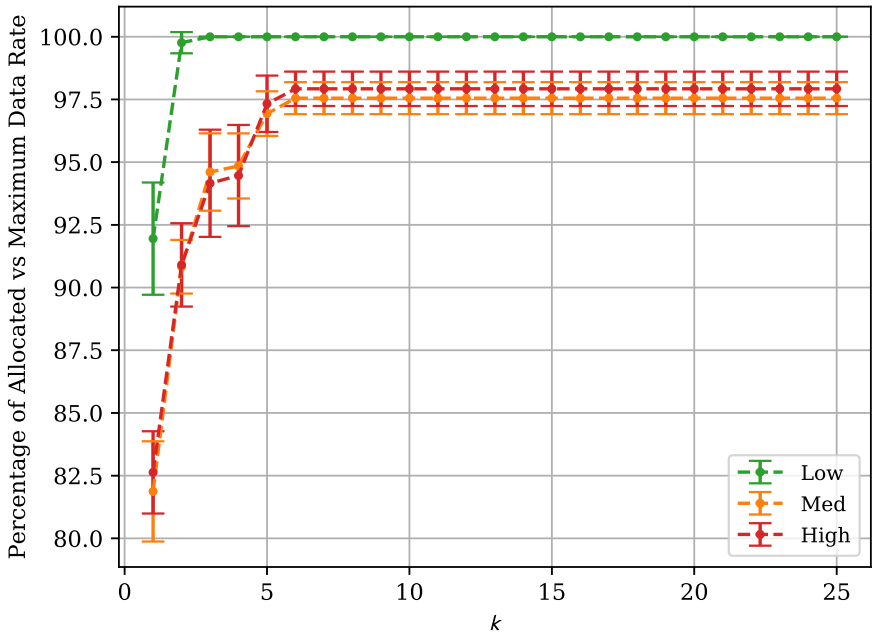
$$0 \leq \hat{\delta}_i \leq \delta_i. \tag{5.7}$$

The constraints (5.5) and (5.6) are set to eliminate the possibility of loops. All the remaining constraints are similar to those used by the MCFP described in Section 2.1. The main difference between the MCFP and the unconstrained Maximum Flow being that a flow can be assigned a lower data rate value than what it requested as set by constraint (5.7).

The k values are incrementally increased from 1 up to 25. Figure 5.3 shows the total network flow allocated by the PC-MFMC problem when varying k for both of the path selection algorithms proposed here as a fraction compared to the unconstrained Maximum Flow problem. Figure 5.3 is only showing the solutions for the largest number of flows available for a given network load. This means 300 flows for the low and medium network load and 150 flows for the high network load. Again, only solutions to the PC-MFMC problem solved using LP are given in this figure because of LP's optimality guarantee. Note that the allocated data rate percentage does not reach 100% even when $k = 25$ because the unconstrained Maximum Flow solution uses links that are not present in the set of 25 paths available to a given flow. From the results in Figure 5.3, it is evident that the KSREDP algorithm finds solutions that are closer to the network's capacity with a lower k value when compared to the KSP algorithm. Under both path selection algorithms chosen here, marginal improvement is reported when k is larger than 5. Seeing such results, we opt to use a k value of 5 as it has the capability to allocate more than 90% of the maximum network capacity under all network load conditions.



(a) Path Selection Algorithm: Ksp



(b) Path Selection Algorithm: KSREDP

Figure 5.3: Percentage of the Allocated Data Rate when using the PC-MFMC algorithm when compared to the unconstrained Maximum Flow problem, with varying k. All five flow sets for a given network load and the highest number of flows for a given network load are used; 300 Flows for the Low and Medium network flow, and 150 flows for the High network load.

5.2 MOEA Parameter Choice

It is important to appreciate that finding the optimal MOEA parameter values is itself a multi-objective optimisation problem, and is heavily dependant on the algorithm's use case. Although several tests were carried out to ensure that the values chosen work well for a wide number of cases, as is demonstrated by the results presented in this chapter, we do not state that these are the optimal values.

Choosing EA parameter values is a long and arduous process due to the multi-objective nature of such a problem. The EA parameter values must be chosen so as to strike a balance between the rate of convergence and the even coverage of the optimal Pareto Front. On the one hand, the improvement between generations should not be very small as this would require a very large number of generations before reaching a good enough approximation of the true Pareto Front. And on the other, there should not be a huge jump between the current and previous population as this may go over solutions that are on the true Pareto Front. In addition, it is also important that the solutions found by the EA, are evenly spread over the entire Pareto Front. Having a number of solutions either grouped in a relatively small area, or heavily biased to one objective, signifies that the EA is not exploring all areas equally. This results in the generated results to be biased towards an area/objective, which is undesirable when using a true multi-objective solver. The ideal set of EA parameters needs to offer a steady and gradual improvement with each generation until a satisfactory Pareto Front is generated. The EA parameter values are highly dependent on the design and function of the designed EA operators, such as crossover and mutation. Due to the large variety of such operators, only recommendations on what values to use exist, with a high crossover probability and a low mutation probability being the norm. In our case, the chosen crossover probability of 0.9 is a typical value used for the NSGA-II algorithm [30] that has been found to work well in our setup. The mutation probability and mutation fraction μ values have been chosen empirically by observing the evolution from a number of different generated solutions. As explained earlier, the values that show the best balance between the aggressiveness of the algorithm in searching for a new solution and the number of generations required by the EA to converge have been chosen. The MOEA parameters used to generate the results presented here are given in Table 5.2.

When choosing the parameter values for an EA, a holistic approach must be taken and no decision should be based solely on the information given by a graph that represents the progress of a single parameter value. The figures that will be shown in the following sections, serve only as an indicator on the correctness of a particular

Table 5.2: MOEA parameters

Parameter	Value
Population Size ($\mathcal{P}$)	800
Number of Generations (χ)	400
Crossover Probability (ω)	0.9
Mutation Probability (ψ)	0.2
Fraction of flows to be mutated (μ)	0.2

parameter value. During the EA parameter selection process, the effect on the EA's evolution progress when changing a single parameter value has been thoroughly analysed and tested before taking the final decision.

5.2.1 Number of Generations

A good metric to use to determine whether the EA has converged to a set of solutions is to observe the progress between successive generations. An EA is said to converge when there is little to no improvement in the quality of the solutions being generated by the EA. Such a metric will highlight whether the EA has been stopped pre-maturely or whether the number of generations can be reduced without heavily impacting the final output of the algorithm. The metric used in this work to quantify the progress between successive Pareto Fronts is the mean Euclidean distance. The mean Euclidean distance $M_{A,B}$ between the set of solutions on the Pareto Front in generations A and B, respectively S_A and S_B, is calculated as follows

$$M_{A,B} = \frac{1}{|S_A|} \sum_{a \in S_A} \min_{b \in S_B} d(a, b), \tag{5.8}$$

where $d(a, b)$ is the Euclidean distance between a and b. a and b are tuples of the normalised objectives for the algorithm used. Figure 5.4 shows the mean Euclidean distance between successive Pareto Fronts for a randomly chosen flow set. Similar conclusions can be made from all of the other flow sets used here. From Figure 5.4 it can be observed that both EAs presented here tend to stabilise after approximately 250 generations. Seeing such results, we have decided to set the number of generations to 400 to allow for some fluctuation in the number of generations the EA requires to stabilise.

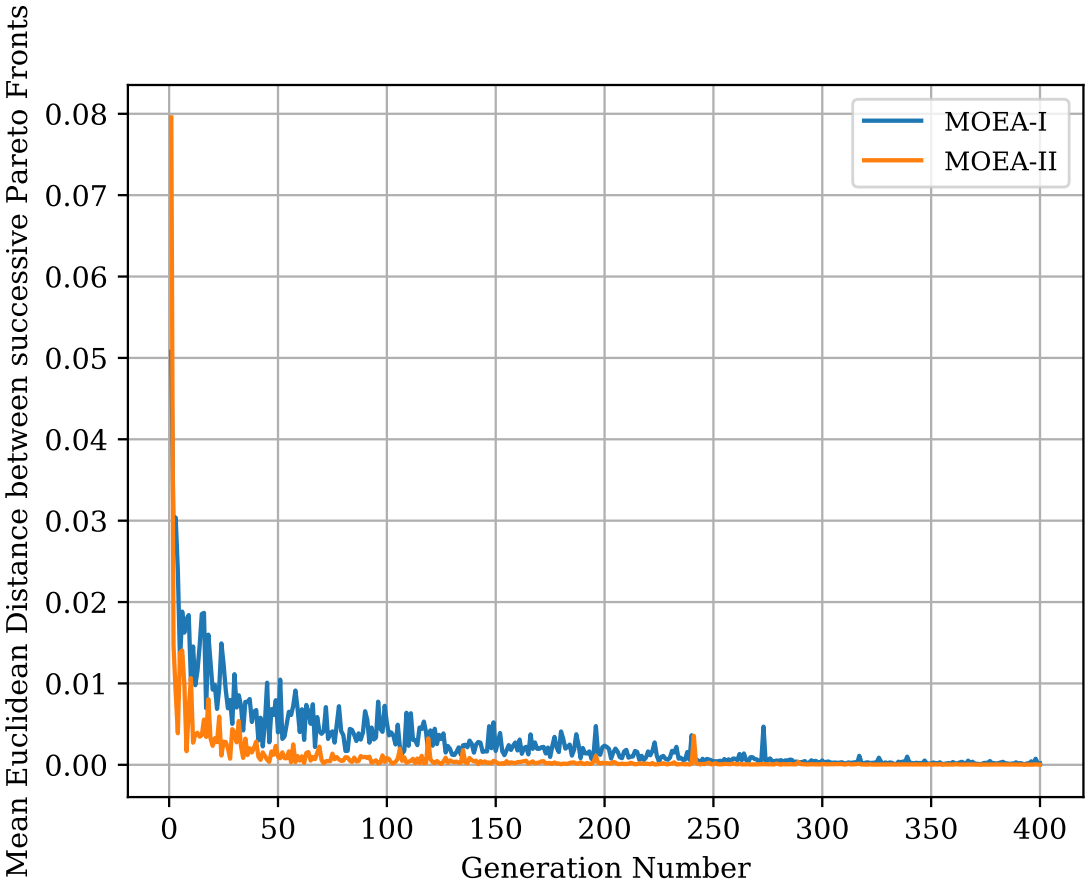


Figure 5.4: Mean Euclidean distance between successive Pareto Fronts

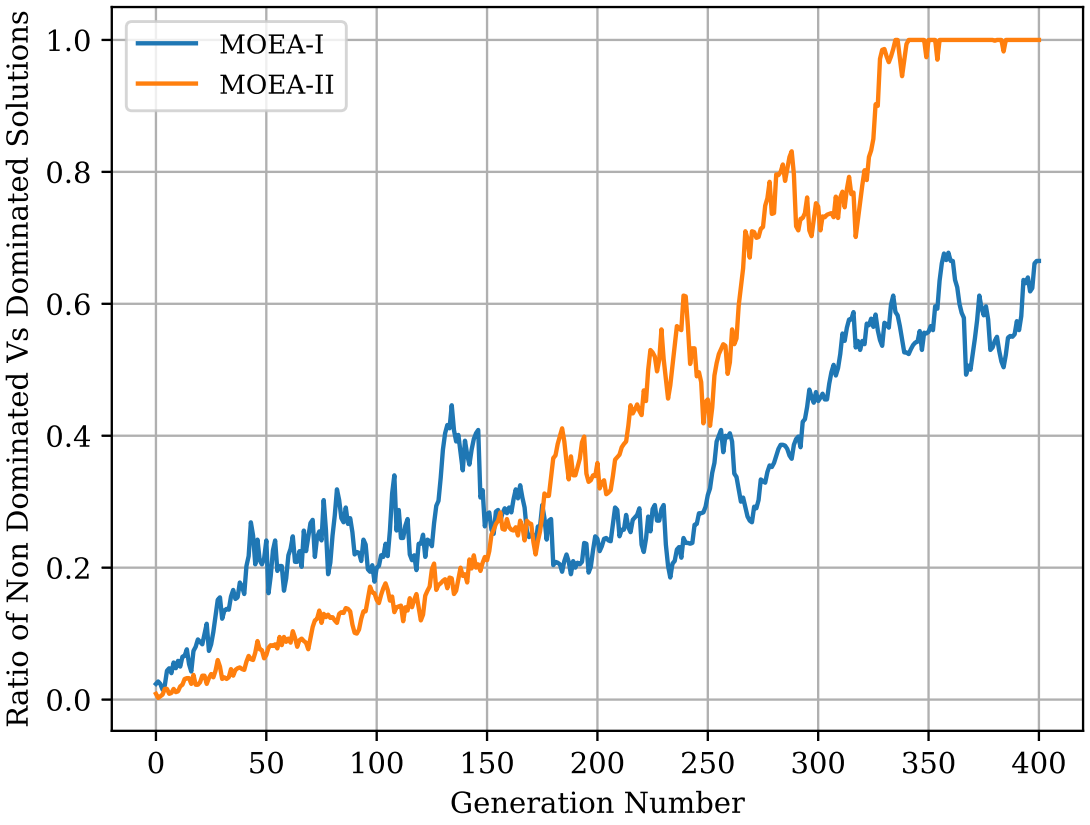


Figure 5.5: The ratio between non-dominated and dominated solutions for every generation.

5.2.2 Population size

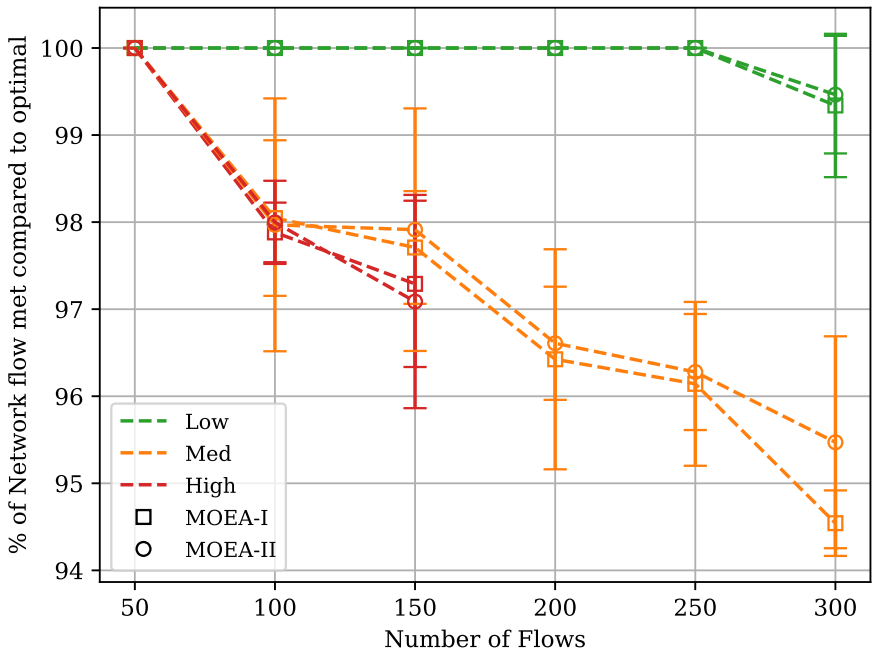
Selecting an EA's population size is very dependent on the problem at hand, and as such, there are no rules on how big or small an EA's population should be. Deb [22] presents a graph suggesting the minimum population size based on the number of objectives and the proportion of non-dominated solutions in the initial population. However, such a graph must only be used as a reference and one should adjust the population size accordingly as deemed fit. The number of solutions in an EA's population should be large enough to allow the EA to explore different regions of the Pareto Front, but not unnecessarily large that the running time of the algorithm is compromised. One measure that helps to determine whether the population size is large enough or not is the ratio between the number of non-dominated and dominated solutions. Ideally, the population should not be made entirely of non-dominated solutions, especially at the early stages of the evolution as dominated solutions provide the necessary diversity that allows the algorithm to better explore solutions in all directions. Figure 5.5 shows the ratio between the number of dominated and non-dominated solutions for every generation for a given flow set, chosen at random. Observe that as the evolution progresses, the number of non-dominated solutions also increases, but only reaches a probability

of one at the final stages of the evolution, if it ever reaches such a value. Having a graph with a profile similar to the one shown in Figure 5.5, is a good indicator that the chosen population size is sufficient as it still has enough space to store some dominated solutions. Note that in the case of the MOEA-II, for the particular flow set seen here, the ratio of Non-Dominated vs Dominated solutions reaches one at the very last stages of the evolution. Such behaviour should not be the sole reason that the population size is increased, as the shown behaviour may be due to the convergence of the EA. In such cases, increasing the population size further would not improve the EA's performance any further, but rather increase the algorithm's running time.

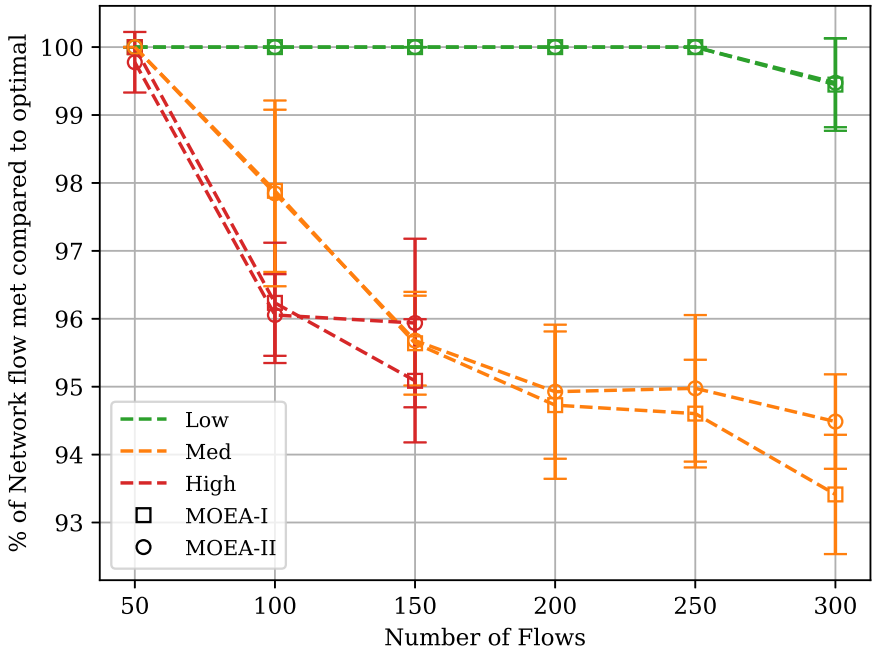
5.3 Performance Analysis

5.3.1 MOEA vs LP

One of the key questions to answer when designing an EA, is to determine how close the generated solutions are to the optimal, under various conditions. The solution to the PC-MFMC problem using LP returns the maximum demand the network is able to handle for a particular flow set. To compare the quality of the solutions found by the algorithms used here, the solution with the highest network flow found by the EA is compared with that found by LP, for all the flow sets and network loads used here. Figure 5.6 shows the percentage of demand the EA managed to satisfy when compared to the LP solution. From the results, we can observe that both EA routing algorithms match the solutions found by LP very closely, for a lightly loaded network. As the network load and number of flows increase, the difference between the EAs satisfied demand and the optimal, deviate slightly. The EA found solutions deviate by at most 7% and 8% when using the KSP and KSREDP path selection algorithms, respectively. These results show that even though EAs are inherently sub-optimal, the EAs designed here manage to satisfy, on average 98% of the demand found by the optimal LP generated solution.



(a) Path Selection Algorithm: KSP



(b) Path Selection Algorithm: KSREDP

Figure 5.6: Plot illustrating the Percentage of demand achieved by the EA algorithms when compared to the optimal solution to the PC-MFMC problem solved using LP. The solution with the largest network flow found by the EAs is used.

5.3.2 Promised vs Actual Network Performance

The main hypothesis this research is based on revolves around the assumption that using a globally optimal routing solution leads to an increase in network efficiency. A globally optimal routing solution should be devoid of congestion; therefore, one comes to expect that when deploying such a routing solution on a network, the network performance should closely match the performance guaranteed by the routing algorithm. To measure how close the actual network performance is compared to the routing solution, the flow satisfaction metric is used. A flow's satisfaction rate represents the fraction of goodput received when compared to the goodput allocated by the routing algorithm. For example, a flow that is allocated 10 Mbps by the routing algorithm, and receives 10 Mbps, is said to have 100% satisfaction rate. On the other hand, if the flow receives only 5 Mbps, the flow satisfaction rate drops to 50%. Figure 5.7 is a boxplot showing the distribution of the flow's satisfaction rate for all the flows in a given flow set. The tighter the distribution and the closer it is to 100%, the closer the actual network performance is to the one given by the routing solution.

It is well known that TCP and a *Per-Packet* multipath routing solution should not be used together. It comes as no surprise that when using standard TCP and PPFS, the network performance is nowhere near that promised by the routing algorithm. This can be verified by looking at the columns in Figure 5.7 that use the PPFS method to deploy *Per-Packet* multipath. What may not be so obvious is the impact the inclusion of acknowledgement flows have on the flow satisfaction rate. Comparing the flow satisfaction rate distribution between the setup when considering acknowledgement rates and when not, the negative effects of ignoring acknowledgements on the flow's satisfaction rates becomes clear. Ignoring the acknowledgement flows leaves the network open to congestion as soon as the acknowledgement flows start. Since acknowledgement flows were not taken into consideration when generating the routing solution, not enough capacity may have been left in the network links to route the additional traffic. Therefore, as soon as the acknowledgement flows start transmitting, some links may be used beyond their capacity, causing network congestion. Figure 5.7 also shows that the modifications done to the MPTCP protocol allow TCP consumer applications to benefit from the advantages offered by a *Per-Packet* multipath routing algorithm with the vast majority of flows having 100% flow satisfaction rate. Even though only one particular flow set is shown in Figure 5.7, similar conclusions and distributions were seen under a different flow set. Therefore, we have no reason to believe that the results and conclusions drawn in this section fail to apply under different scenarios than the ones considered here.

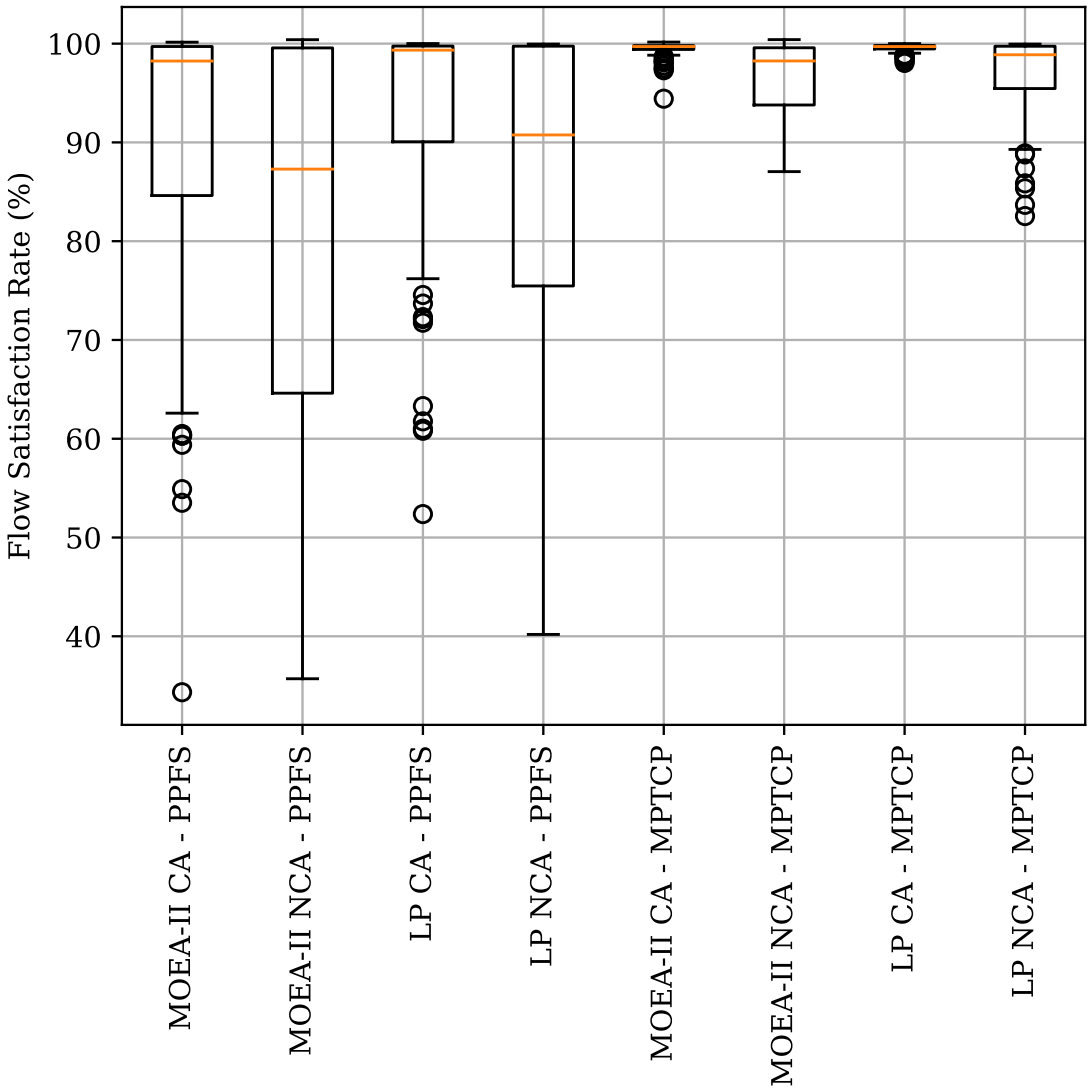
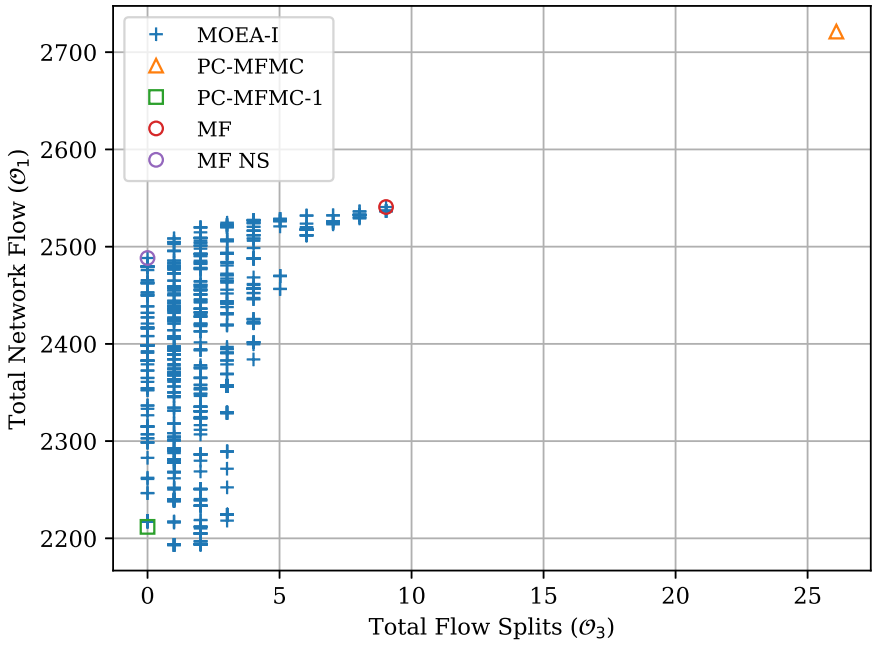


Figure 5.7: Boxplot showing the distribution of the flow's satisfaction rate. EA: Evolutionary Algorithm, LP: Linear Programming, CA: Considering Acknowledgements, NCA: Not Considering Acknowledgements.

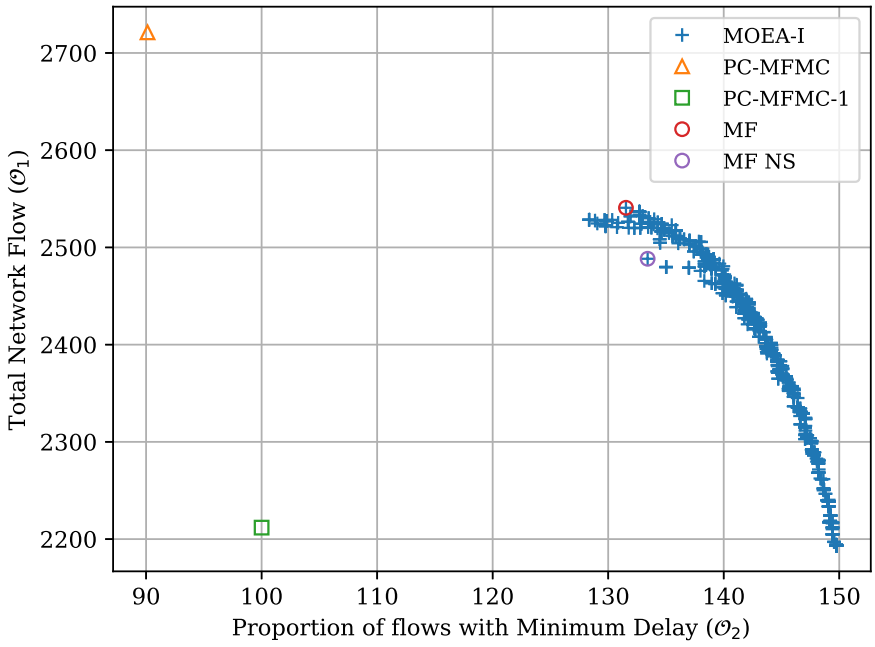
5.3.3 The Advantage of Having Multiple Solutions

As we have seen from Figure 5.7, using TCP and a *Per-Packet* multipath routing algorithm leads to an overall lower flow satisfaction rate when compared to its MPTCP counterpart. In this section, we assume that only the standard TCP protocol is available. This setup is specifically chosen to demonstrate the advantage of having multiple valid solutions to choose from when using an EA, compared to having only a single solution, as is the case when using an LP solver. Because only TCP is assumed to be available, solutions that only use *Per-Flow* multipath, i.e. solutions where no flow is split over multiple paths, are preferred. Modifying the PC-MFMC problem to restrict a flow to only use a single path from the set of paths available to said flow, drastically increases the problem's complexity as it now becomes NP-hard [14]. In this section, the MOEA-I algorithm is used because of its flow splits minimisation objective, that steers the EA to prefer *Per-Flow* over *Per-Packet* multipath solutions. From the multiple solutions generated by the algorithm, the solution offering the highest network flow, without resorting to *Per-Packet* multipath is chosen. An EA generated solution with zero flow splits is classified as a *Per-Flow* multipath solution because even though all the flows are travelling over a single path, different flows from the same source-destination pair are allowed to take different paths.

Figure 5.8 shows orthogonal projections of the Pareto Front generated by the MOEA-I algorithm with the LP found solutions overlaid on the same plot for comparison purposes. Figure 5.8 is generated using the third flow set of the high, 150 flows network load when using the KSREDP path selection algorithm. This particular flow set is chosen because the LP solution has the largest number of flow splits compared to all the others. Such a gap makes for easier network performance comparison between the two algorithms. The PC-MFMC-1 results are generated by solving the PC-MFMC problem with $k = 1$. If multiple paths with the same cost exists, one is chosen at random. The PC-MFMC-1 results are included as they represent the highest attainable network flow that can be achieved if a single path routing algorithm, such as OSPF is used. Note that the total network flow allocated by PC-MFMC-1 is an upper bound and does not mean that a network running OSPF will reach the same level of network performance as given by the PC-MFMC-1 solution, assuming the same flow set. The main reason behind the performance difference between PC-MFMC-1 and an actual network running OSPF is that PC-MFMC-1 is globally optimal while OSPF is not. In other words, when generating a routing solution using PC-MFMC-1, all flows using the network are taken into consideration to avoid any link capacity overload. Such a constraint may result in flows being assigned no data rate at all, which is why the *Proportion*



(a)



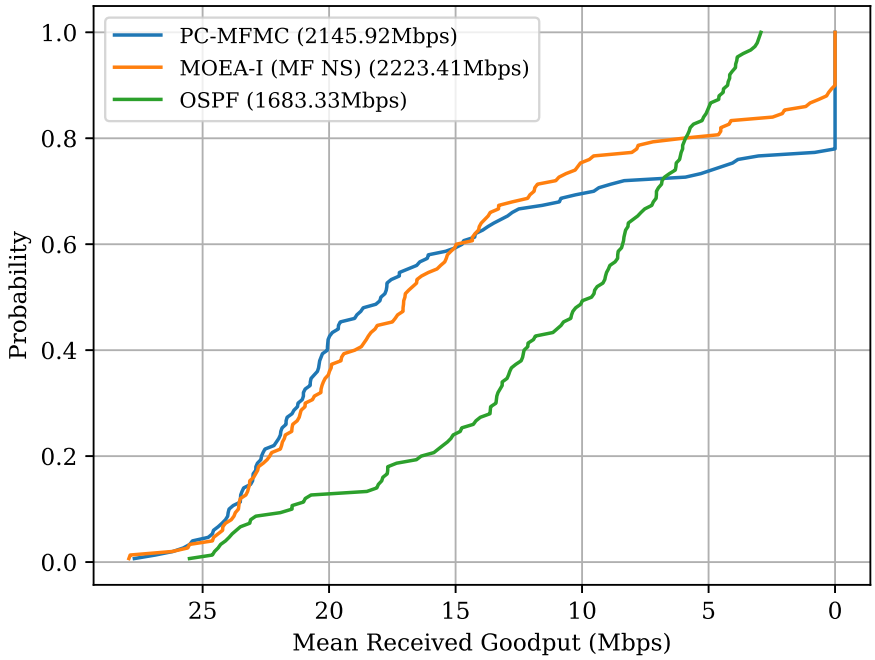
(b)

Figure 5.8: Orthogonal projections of the Pareto front for the MOEA-I algorithm together with the LP solutions. The red (MF) and the purple (MF NS) circles mark the solutions with maximum network flow, and maximum network flow with no flow splits, respectively. PC-MFMC-1 represents the solution found by LP when only the shortest path is available and is used to represent the maximum attainable network flow by OSPF.

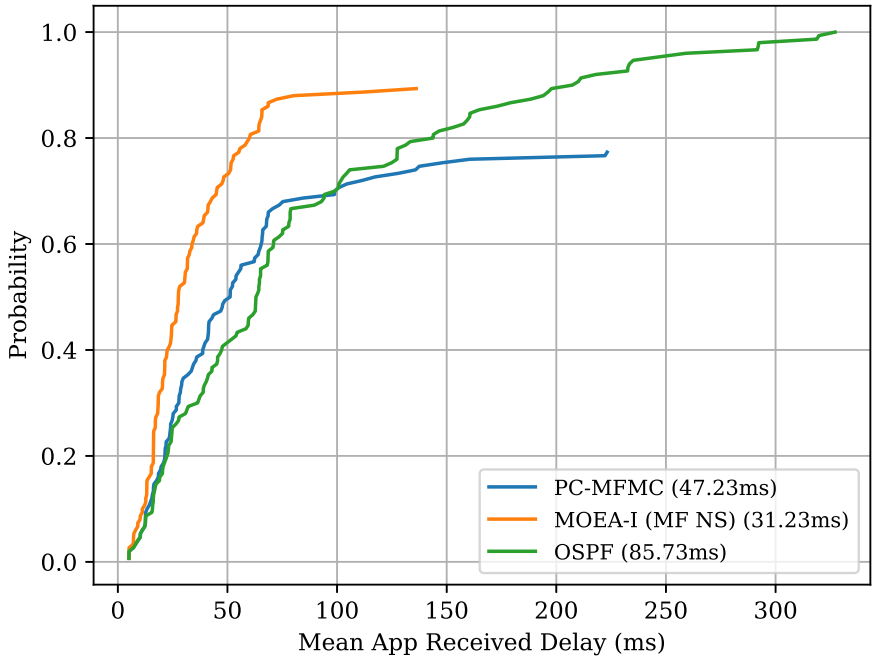
of Flows with Minimum Delay objective value is not equal to its maximum value under the PC-MFMC-1 scenario. On the contrary, a network using the OSPF routing algorithm does not have the means to control the traffic entering the network, meaning that all flows transmit at their requested data rate. The avoidance of network congestion when using an OSPF like routing algorithm is the responsibility of transport layer protocols such as TCP.

Looking at the *Total Flow Splits* objective value for the PC-MFMC found solution in Figure 5.8(a), it is evident that LP resorted to the use of *Per-Packet* multipath to reach the maximum network flow for the given path set. Again, using Figure 5.8(a) for reference, it is abundantly clear that the solution with no flow splits has a much lower *Total Network Flow* than its PC-MFMC counterpart. Therefore, one expects that the PC-MFMC solution would have the best network performance. However, this is not the case as can be observed by looking at Figure 5.9. Figure 5.9(a) shows the distribution of the flow's received Goodput. Comparing the EA and PC-MFMC solutions, in terms of goodput, both of their profiles are very similar with the EA having a higher number of flows that are allocated some data rate, as well as having a higher probability that a flow receives a goodput rate of at least 10 Mbps. The high number of flows allocated some data rate given by the EA solution is thanks to the the inherent design property of the *Proportion of Flows with Minimum Delay* objective with solutions that have unassigned flows scoring lower than their counterparts. On the other hand, the LP solution has a higher number of flows receiving a goodput larger than 15 Mbps. Both solutions have a similar number of flows that are receiving a goodput of at least 23 Mbps. Analysing this figure we can see that OSPF is the only solution where all of the flows are receiving something. However, this comes at the cost of the overall network performance as evidenced by the steep decline in the number of flows receiving at least at a rate of 10 Mbps, when compared to the two other presented solutions. Such poor performance with respect to OSPF is caused by its inability to use multipath and the network congestion caused when a link is used beyond its capacity.

Clear signs of congestion when using OSPF are evident in Figure 5.9(b). Figure 5.9(b) shows the distribution of the mean flow's delay at the application level. The advantage of the EA solution over the PC-MFMC is even more pronounced in Figure 5.9(b), where the EA algorithm's solution has a better delay profile and a lower maximum delay value when compared to both the PC-MFMC and OSPF solutions. Such a result bears more weight when considering the fact that the EA's routing solution has a larger number of flows that are using the network at any one time.



(a) Probability that a flow achieves at least a given average Received Goodput in Mbps, in simulation. The actual total network flow achieved by each algorithm is shown in parentheses.



(b) Probability that an application experiences at most a given average delay, in simulation. The mean application delay achieved by each algorithm is shown in parentheses.

Figure 5.9: Network Performance of the MOEA-I and PC-MFMC generated solutions when using TCP.

The setup used here, together with the presented results, demonstrate the advantage of having multiple solutions generated by a true multi-objective solver, such as an EA, even though one loses the optimality guarantee.

5.3.4 Hybrid Routing Algorithm

So far we have compared the EA and LP routing solutions against each other, highlighting the strength and weaknesses of both. In this section we merge the two algorithms (EA and LP) to create a single Hybrid routing algorithm. The MOEA-II will be used as the foundation for the Hybrid routing algorithm; however, the techniques used here can be easily transferred to any population based EA algorithm. The Hybrid algorithm is a minor modification over the standard MOEA-II algorithm. The only modification to the MOEA-II algorithm is the addition of the LP optimal solution to the initial population. Note that the population size remains equal to 800 including the LP optimal solution. The advantages of such a Hybrid algorithm are twofold. First, the final population generated by the EA routing algorithm is guaranteed to have a solution with the highest data rate the network is able to handle, thanks to the elitist nature of the NSGA-II algorithm. Second, the number of generations required by the EA to converge is reduced, as can be seen from Figure 5.10. In Figure 5.10, the Hybrid algorithm reports a mean euclidean distance of zero for the first few generations because the PC-MFMC solution dominates all of the generated solutions. Therefore, the Pareto Front is only made up of a single point (the PC-MFMC solution), which results in a zero mean euclidean distance. The obvious downside of such a Hybrid algorithm is that both the LP and EA algorithms have to be run. Figure 5.11 shows the generated Pareto Front of both the standard MOEA-II and Hybrid algorithms. Figure 5.11 is using the third flow set, for the 150 flow, high network load when using the KSP path selection mechanism. This particular flow set has been chosen because of its gap in the network flow allocated between the PC-MFMC optimal solution and the solutions found by the MOEA-II algorithm. From Figure 5.11 it can be observed that the Hybrid algorithm found a number of solutions that dominate those of the standard MOEA-II algorithm. However, the final Pareto Front of the Hybrid algorithm is much tighter than the one found by the MOEA-II. This is not ideal, as it shows that the Hybrid algorithm is not looking into all directions equally, but rather leaning towards more network flow oriented solutions. The Hybrid algorithm was developed and tested during the last stages of this research work; therefore, there was not enough time left to identify the cause of such constricted Pareto Front. To try and identify the cause of such a problem, one may start by looking at the entire evolution of the algorithm to determine at which stages

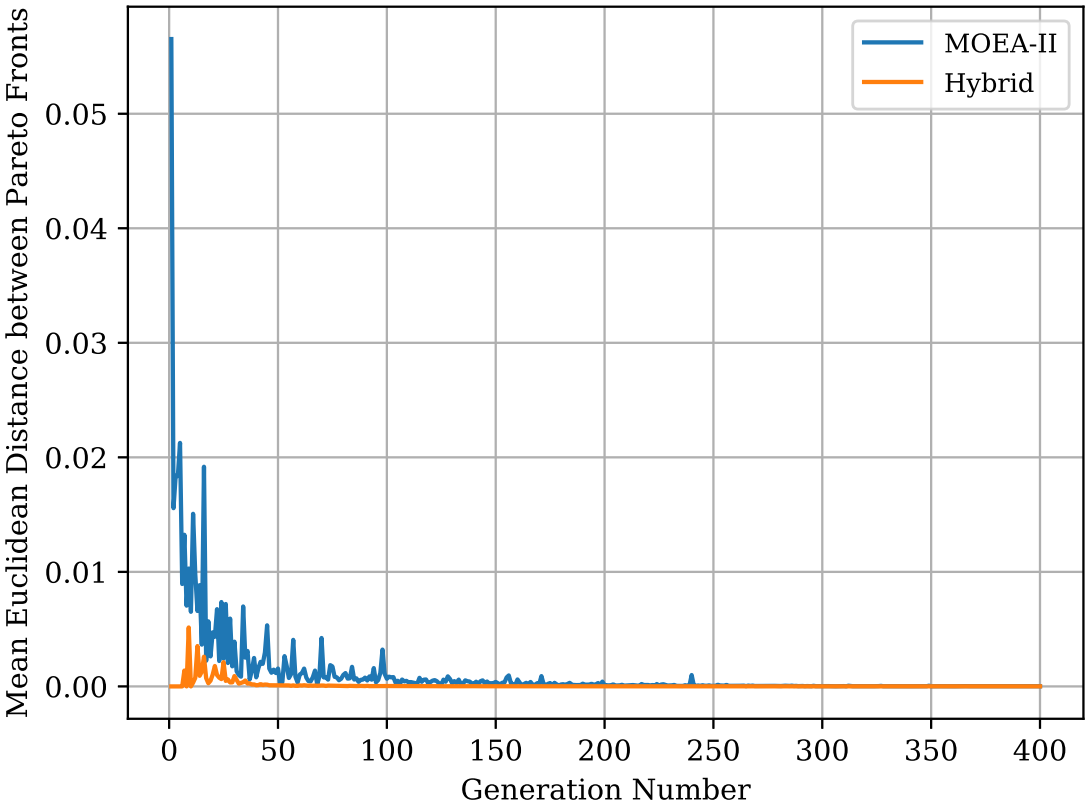


Figure 5.10: Mean Euclidean distance between successive Pareto Fronts comparing the hybrid and non-hybrid versions of the EA.

the Pareto Front starts to close up. If this behaviour is noticed even at the early generations, the insertion of the optimal in the first generation might be too early. Having an optimal solution in the initial population might require tweaks to the EA parameters to either increase or decrease the aggressiveness of the mutation operator. Having said this, the Hybrid algorithm still managed to achieve what it was originally set out to do.

The network performance of the solutions marked in Figure 5.11 are given in Figure 5.12. As expected, the Hybrid MF solution has the overall better network goodput performance when compared to all of the other points chosen here. OSPF has the worst overall performance both in terms of goodput and mean application delay, even though all the flows are receiving some flow. Note that OSPF's delay performance is kept in check thanks to the TCP congestion control mechanism that tries to avoid network congestion. Observe that the MOEA-II (MF) solution has an overall better delay performance when compared to the Hybrid (MF) solution. This means that even though MOEA-II's *Estimated Mean End-to-End Delay* objective is an approximation, it correlates well with the actual delay performance.

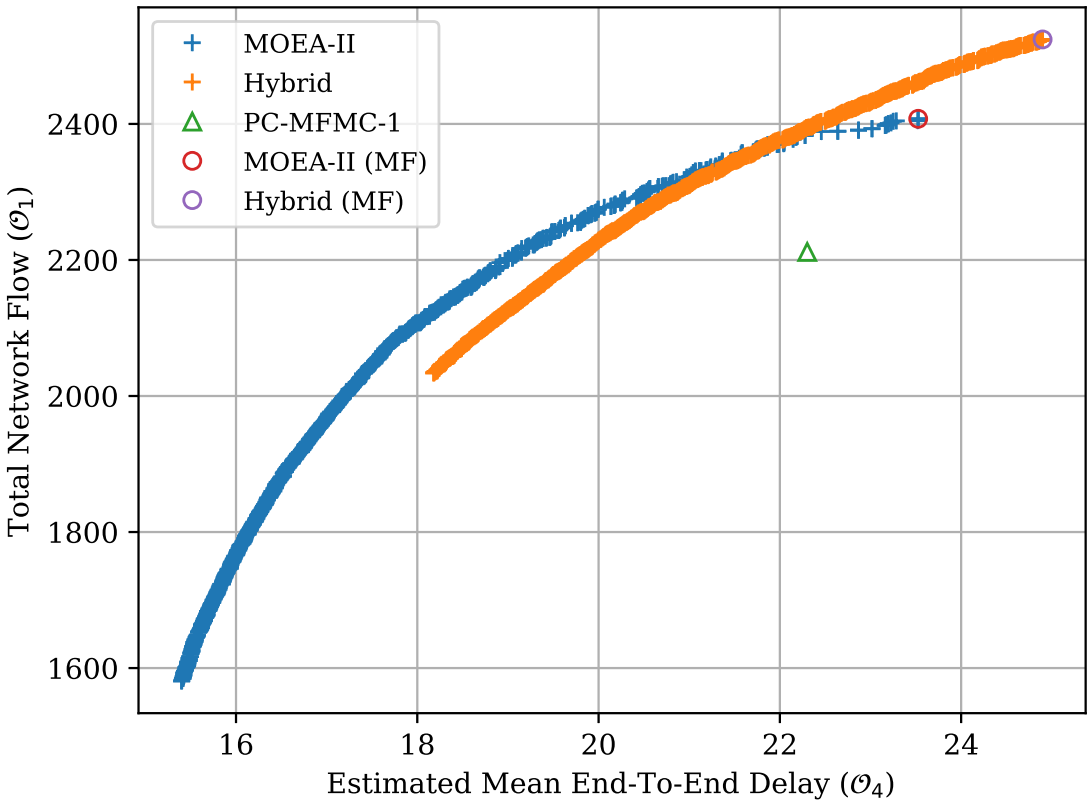
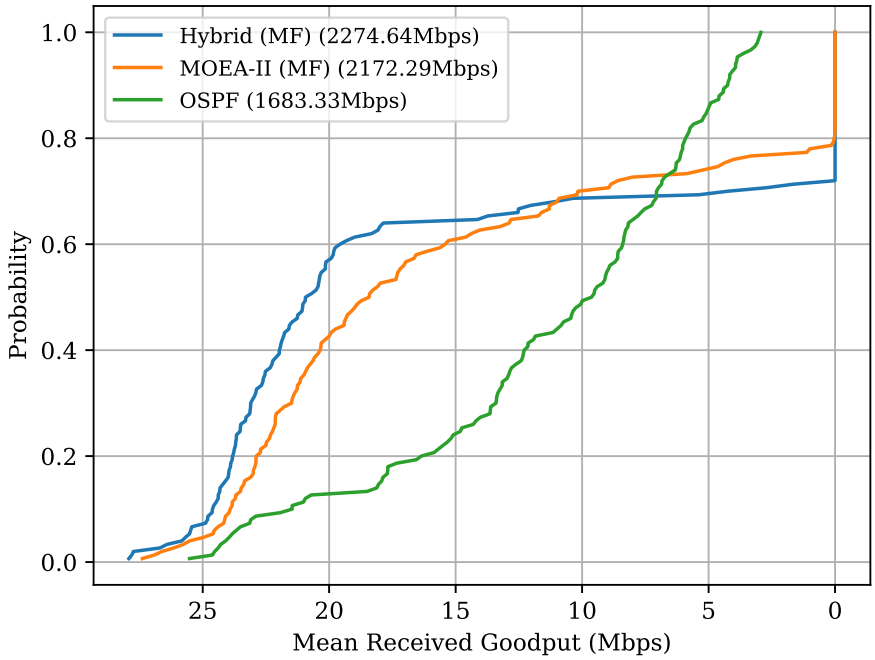
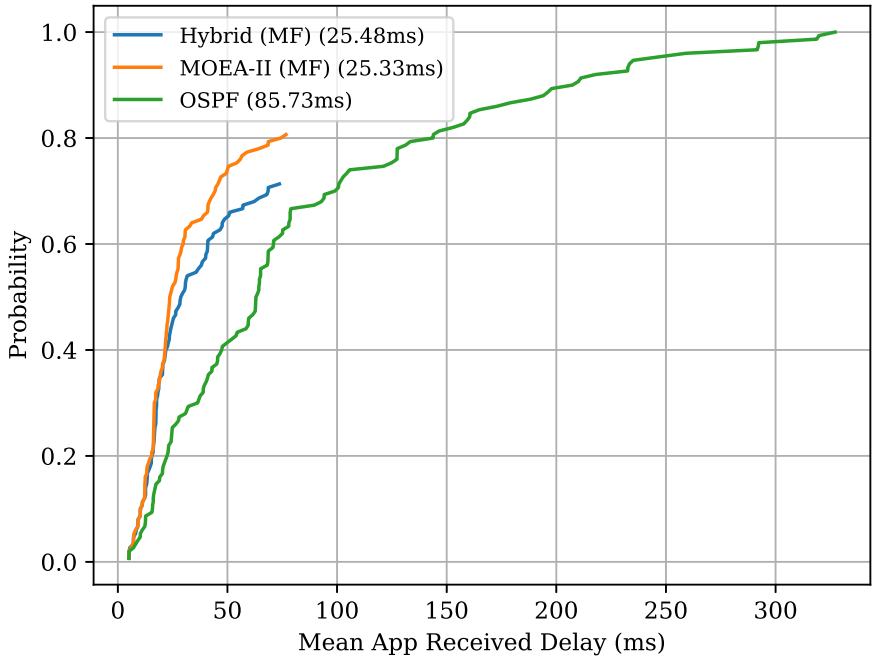


Figure 5.11: MOEA-II represents the Pareto Front generated by MOEA-II when the population is generated entirely at random. Hybrid represents the Pareto Front generated by MOEA-II when the LP solution is added in the initial population. PC-MFMC-1 represents the solution found by LP when only the shortest path is available and is used to represent the maximum attainable network flow by OSPF. MF marks the Maximum network Flow point for a given population.



(a) Probability that a flow achieves at least a given average Received Goodput in Mbps, in simulation. The total received Goodput achieved by each algorithm is shown in parentheses.



(b) Probability that a flow experiences at most a given average end-to-end delay at the application layer, in simulation. The mean application delay achieved by each algorithm is shown in parentheses.

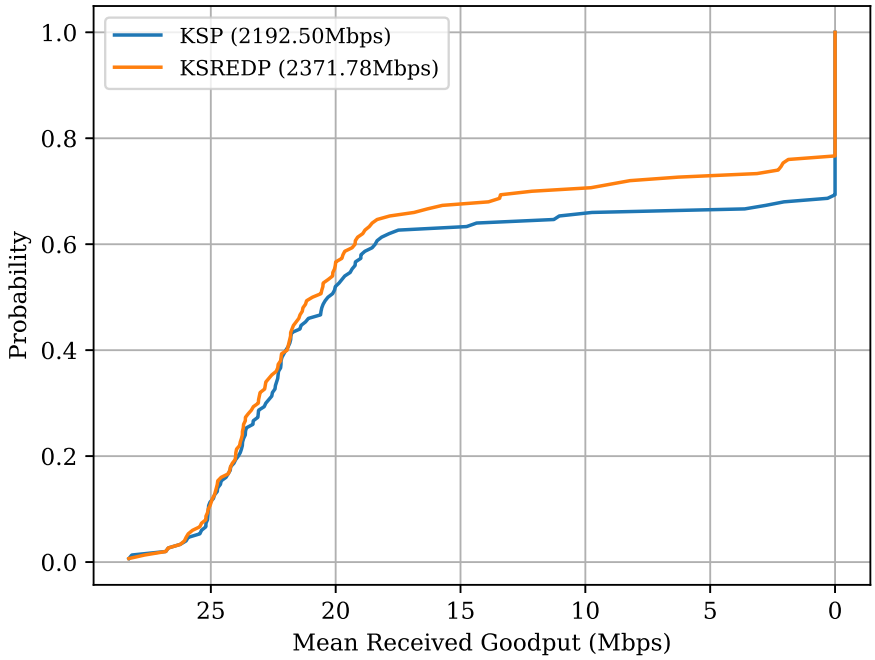
Figure 5.12: Network Performance comparison between the MOEA-II and Hybrid algorithm. MF refers to the solution with the highest network flow value.

5.3.5 Effect of Path Selection Algorithm on Network Performance

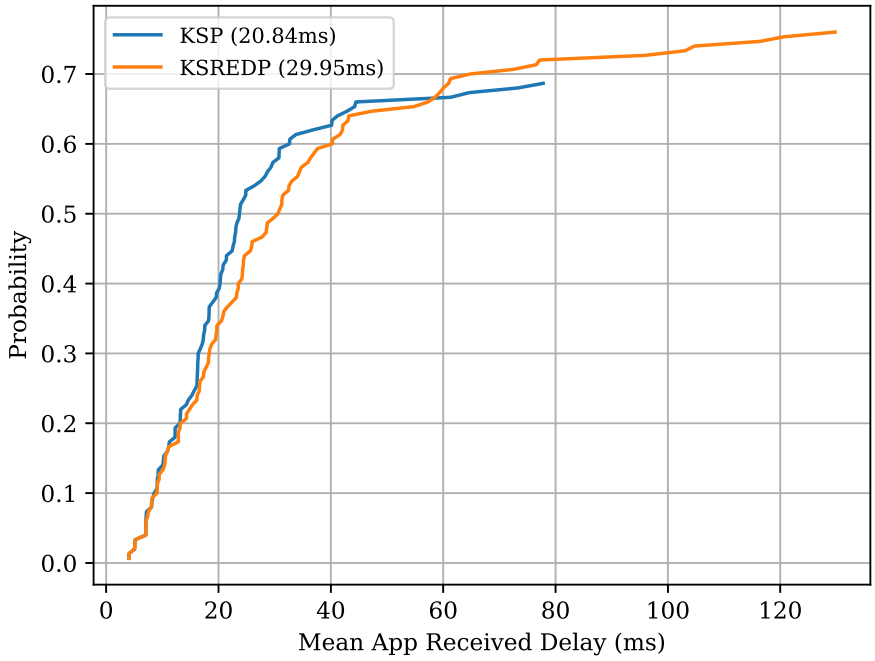
The method used to retrieve the set of paths a flow is allowed to use affects the final routing solution and therefore the network performance. As we have seen in the results presented in Section 5.1.3, both the k value and the path selection algorithm have an effect on the amount of data rate a routing solution is able to carry over a given, fixed, network topology. In this section we back up the statements made in Section 3.7 by providing network simulation results. As explained in Section 3.7, the KSP algorithm is suited for applications that prioritise delay over throughput, while the opposite is true for the KSREDP algorithm.

The KSREDP algorithm is preferred by throughput oriented flows as it has been designed to increase link diversity. As the number of different links available to a flow increases, so does the probability of assigning higher capacity due to the fact that the paths do not have overlapping links. Having said this, using the KSREDP algorithm for a flow is not guaranteed to return better performance than the KSP as this is highly dependent on the network topology and flow set. On the other hand, when using the KSP algorithm, it is guaranteed that the shortest k paths are made available to a flow. This means that paths chosen by the KSP algorithm will always outperform or at least match the paths returned by the KSREDP algorithm, in terms of delay. However, the paths returned by the KSREDP algorithm are not guaranteed to always outperform the paths returned by the KSP algorithm, in terms of network capacity.

The network topology used in this work is highly inter-connected; therefore, there is a good chance that the paths found by the KSREDP path selection mechanism do not share a large number of links. This link variation brought forward by the KSREDP algorithm allows the routing algorithm to find solutions with a higher amount of total allocated data rate when compared to their KSP alternative. However, the higher network capacity offered by the KSREDP algorithm comes at the cost of worse delay performance when compared to the KSP algorithm. The network simulation results shown in Figure 5.13, confirm such statements. The network performance results shown in Figure 5.13 only use routing solutions to the PC-MFMC using an LP solver. Only LP solutions are used here due to LP's optimality guarantee, such that any performance difference will be solely attributed to the different path selection methods used. The probabilities in Figure 5.13(b) do not add up to one as unassigned flows are set with a delay value of infinity. Setting unassigned flows with a delay value of 0 ms would result in such flows to be perceived as having a very good delay performance, which is not the case; there-



(a) Probability that a flow achieves at least a given average Received Goodput in Mbps, in simulation. The total received Goodput achieved by each algorithm is shown in parentheses.



(b) Probability that a flow experiences at most a given average end-to-end delay at the application layer, in simulation. The mean application delay achieved by each algorithm is shown in parentheses.

Figure 5.13: Network performance comparison between the KSP and KSREDP path selection algorithms. Figure generated using the High network load flow set 5 with 150 flows.

fore, we have opted to set unassigned flows with a delay value equal to infinity. Based on the results shown in Figure 5.13, flows that prefer throughput over delay will be better off using the KSREDP path selection algorithm as it increases their likelihood of being allocated more data rate. On the other hand, flows that require the best delay performance should opt to use the KSP algorithm. The scale of the performance difference between the two path selection algorithms relies heavily on the network topology. Developing and testing a solution where each flow gets to choose its own path selection mechanism based on its priorities is an area worth looking into further.

Chapter 6

Conclusion

The aim of this research has been to increase the efficiency of already deployed computer networks by developing a globally optimal, multipath capable routing algorithm for SDN networks. After conducting a thorough literature review on the subject, it became apparent that although multipath routing algorithms already exist, there is a lack of a routing algorithm capable of finding a routing solution for multiple objectives at once. Therefore, a novel routing algorithm framework using EAs is designed. The designed routing algorithm is both globally optimal, and *Per-Packet* multipath capable, to ensure the best use of the available network resources. A globally optimal routing algorithm is an algorithm that takes into account all the flows currently using the network when generating a solution. Such a routing algorithm is used in this work as it avoids network congestion caused by over-used network links thanks to its global network knowledge. Gaining such global knowledge when using a distributed architecture is nearly impossible to do efficiently, which is why we opted to use the SDN network architecture instead. The SDN architecture is a relatively new architecture that centralises the network's control plane into a single entity. This control plane centralisation gives the network controller all the information required for the deployment of a globally optimal routing algorithm and is the main reason behind our decision to use SDNs.

Current SDN switches are unable to handle any set of flow split ratios without running into scalability problems. Therefore, modifications are proposed that enable an SDN switch to handle any flow split ratio in a scalable fashion and allow the testing of the developed routing algorithm on a simulated network. While splitting a flow at the packet level is a solution that works well with connection-less protocols, such as UDP, connection oriented protocols, such as TCP are negatively impacted by the use of a *Per-Packet* multipath solution. An alternate solution of deploying *Per-Packet* has been proposed based on the MPTCP protocol. The

MPTCP protocol is modified to include a stochastic scheduler capable of splitting a flow in the ratios instructed by the routing algorithm. The use of a globally optimal routing algorithm voided the need for MPTCP's shared congestion control algorithm, and is not implemented in this work. The combined use of the updated MPTCP protocol, and the inclusion of the TCP acknowledgement flows when generating a routing solution, guarantee, with a very high probability, that a flow reaches the data rate assigned to it by the routing algorithm. Such a claim is backed up with results generated using network simulations.

EAs are known to be sub-optimal; however, the EAs proposed here find solutions that are just 2% off the optimal flow solution found using LP for all the scenarios considered here. The main advantage of a multi-objective solver is the provision of a number of different viable solutions with different compromises on each of the objectives. The advantages of such a feature have been used in a setup where only standard TCP is assumed to be available. The ability to choose a solution from a Pareto Front resulted in the chosen solution to have better overall goodput and delay performance when compared to the optimal solution found by LP. The LP formulations could not be updated to cater for such a scenario (use *Per-Flow* multipath exclusively), without making the problem an NP-hard one. Finally, a Hybrid algorithm is proposed that combines the multi-objective nature of EAs with the optimality guarantee of LP. The Hybrid algorithm works by inserting the LP optimal solution in the EA's initial population. Since an elitist EA is used here, the LP optimal solution is guaranteed to be present in the final population. Additionally, the presence of the LP solution in the EA's population increases its convergence rate reducing the number of generations required. Under all conditions considered in this work, all of the developed routing algorithms outperform OSPF under network simulation conditions. This performance advantage is thanks to the routing algorithm's ability to use *Per-Packet* multipath, and the ability of the modified MPTCP protocol that allows TCP applications to reach the data rate assigned to them by the routing algorithm.

To summarise, this work presents a MOEA framework for generating globally optimal, *Per-Packet* multipath, routing solutions. The EA framework developed here can be used by future researchers that have to solve a similar problem to the one tackled here. Obviously, the objectives may differ than the ones used here and will therefore have to be modified based on the problem that needs to be solved. Although positive results have been demonstrated, limitations on the presented work exists and will be tackled in the next section.

6.1 Limitations and Future Work

The major limitation surrounding this work is the lack of testing on an actual network. In order to overcome such a limitation, three main issues have to be tackled and overcome. First, the MOEA algorithm has been written in Python and runs on a single thread. Python was chosen because of it's reduced development time which fits well with this project's priorities which were focused more towards the algorithm's network performance rather than its deployability. However, using Python resulted in an unacceptably long running time for the MOEA-II algorithm to finish. This makes the deployment of the MOEA-II algorithm on an actual network unfeasible. Therefore, implementing the algorithm in a more efficient language, such as C++, and parallelising the EA are the first steps that must be taken if such an algorithm is to be tested on an actual network. Developing the EA to work on Graphical Processing Unit (GPU) processing is an avenue worth investigating due to their high number of processor count. Second, a third objective needs to be added to the MOEA-II that minimises the number of routing table changes required to deploy the new routing solution when given the current solution that the network is currently running. This objective is important as it reduces the number of rules that have to be installed on the network, increasing the network's stability. Careful consideration has to be taken when designing such an objective to ensure that the algorithm is still exploring solutions that require a large number of rule changes. This property is important as there might be instances where the network efficiency would be heavily impacted if only a small number of rule changes are allowed. To help avoid such scenarios, the solution to the PC-MFMC may be inserted in the initial population, as is done in the Hybrid algorithm. Third, the updates done to the MPTCP protocol in Ns3 have to be added to the Linux Kernel's implementation. Once these tasks have been completed, the MOEA-II routing algorithm will be able to handle a dynamic flow set and make it ready for testing on a real SDN network.

To date, all the network simulations assume switches are equipped with an infinite buffer size. Network simulations with finite buffers will provide a closer to reality depiction of the performance gap between the devised MOEA algorithm and the OSPF solution. We conjecture that with the use of finite buffers, the gap between OSPF and the developed system will increase due to the packet drops caused by buffer overflow in times of network congestion. The reason being that TCP treats a packet drop as a sign of congestion and immediately reduces the transmission rate. In the current setup, due to infinite buffers, TCP is allowed to adjust to the ever increasing RTT.

Although no design decision has been made on the basis of the network topology being used here, the fact remains that the systems developed here have been thoroughly tested on a single network topology. Every effort has been made to ensure that the routing algorithm's design is independent of the topology it is deployed on, tests on other network topologies are required to confirm this statement. Having said this, the chosen topology used in this work has enough complexity in terms of number of nodes, links and paths between source-destination pairs that gives us enough confidence that the solutions proposed here will work on other topologies without any modifications.

As can be observed from the results presented in this work, the path selection algorithm has an effect on the network performance. A path selection algorithm tailored for each flow's requirements will reduce the EA's running time as paths that do not meet the flow's criteria are pruned, resulting in a smaller chromosome size and consequently, a reduced search space.

During the design stages of new objectives, minor changes to the mutation operator have been carried out to reflect the changes in the objectives. This was done in an effort to increase the EA's convergence rate and improve the quality of the generated solutions. Designing a mutation operator that is independent of the objectives being used increases the versatility of the proposed EA framework as it decouples the objectives from the mutation operators. However, the design of such a mutation operator is not trivial, as the number of generations required by the algorithm to converge must not be heavily impacted because of the addition of a more generic mutation operator.

References

[1] "Cisco Visual Networking Index: Forecast and Methodology, 2016–2021", *Cisco Public White Paper*, Jun. 2017.

[2] S. K. Singh, T. Das, and A. Jukan, "A Survey on Internet Multipath Routing and Provisioning", *IEEE Communications Surveys Tutorials*, vol. 17, no. 4, pp. 2157–2175, 2015, ISSN: 1553-877X. DOI: 10.1109/COMST.2015.2460222.

[3] S. Habib, J. Qadir, A. Ali, D. Habib, M. Li, and A. Sathiaseelan, "The Past, Present, and Future of Transport-Layer Multipath", *arXiv:1601.06043 [cs]*, Jan. 2016, arXiv: 1601.06043. (visited on 12/10/2016).

[4] C. E. Hopps, "Analysis of an equal-cost multi-path algorithm", RFC 2992, Nov. 2000.

[5] N. Farrugia, V. Buttigieg, and J. A. Briffa, "A Globally Optimised Multi-path Routing Algorithm Using SDN", in *21st Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN)*, Paris, France, Feb. 2018, pp. 1–8. DOI: 10.1109/ICIN.2018.8401633.

[6] X. Liu, S. Mohanraj, M. Pioro, and D. Medhi, "Multipath Routing from a Traffic Engineering Perspective: How Beneficial Is It?", in *2014 IEEE 22nd International Conference on Network Protocols*, IEEE, Oct. 2014, pp. 143–154, ISBN: 978-1-4799-6204-4. DOI: 10.1109/ICNP.2014.34.

[7] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer, "Achieving High Utilization with Software-Driven WAN", *SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 15–26, Aug. 2013, ISSN: 0146-4833. DOI: 10.1145/2534169.2486012.

[8] W. Dai, J. Zhang, and X. Sun, "On solving multi-commodity flow problems: An experimental evaluation", *Chinese Journal of Aeronautics*, vol. 30, no. 4, pp. 1481–1492, Aug. 2017, ISSN: 10009361. DOI: 10.1016/j.cja.2017.05.012.

[9] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-Defined Networking: A Comprehensive Survey", *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, Jan. 2015, ISSN: 0018-9219. DOI: 10.1109/JPROC.2014.2371999.

[10] N. Farrugia, J. A. Briffa, and V. Buttigieg, "An Evolutionary Multipath Routing Algorithm using SDN", in *2018 9th International Conference on the Network of the Future (NOF)*, Nov. 2018, pp. 1–8. DOI: 10.1109/NOF.2018.8597865.

[11] ——, "Solving the Multi-Commodity Flow Problem using a Multi-Objective Genetic Algorithm", in *2019 IEEE Congress on Evolutionary Computation (CEC)*, Jun. 2019, pp. 2816–2823. DOI: 10.1109/CEC.2019.8790160.

[12] N. Farrugia, V. Buttigieg, and J. A. Briffa, "Multi-Stream TCP: Leveraging the Performance of a Per-Packet Multipath Routing Algorithm When Using TCP and SDN", in *2019 44th Conference on Local Computer Networks (LCN)*, Oct. 2019.

[13] W. Dai, X. Sun, and S. Wandelt, "Finding Feasible Solutions for Multi-Commodity Flow Problems", in *2016 35th Chinese Control Conference (CCC)*, IEEE, Jul. 2016, pp. 2878–2883, ISBN: 978-988-15639-1-0. DOI: 10.1109/ChiCC.2016.7553801.

[14] H. Masri, S. Krichen, and A. Guitouni, "An ant colony optimization meta-heuristic for solving bi-objective multi-sources multicommodity communication flow problem", in *2011 4th Joint IFIP Wireless and Mobile Networking Conference (WMNC 2011)*, Oct. 2011, pp. 1–8. DOI: 10.1109/WMNC.2011.6097256.

[15] E. M. El-Alfy, S. Z. Selim, and S. N. Mujahid, "Solving the minimum-cost constrained multipath routing with load balancing in MPLS networks using an evolutionary method", in *2007 IEEE Congress on Evolutionary Computation*, Sep. 2007, pp. 4433–4438. DOI: 10.1109/CEC.2007.4425051.

[16] E. M. El-Alfy, "Flow-based path selection for Internet traffic engineering with NSGA-II", in *2010 17th International Conference on Telecommunications*, Apr. 2010, pp. 621–627. DOI: 10.1109/ICTEL.2010.5478839.

[17] E.-S. M. El-Alfy, S. N. Mujahid, and S. Z. Selim, "A Pareto-based hybrid multiobjective evolutionary approach for constrained multipath traffic engineering optimization in MPLS/GMPLS networks", *Journal of Network and Computer Applications*, vol. 36, no. 4, pp. 1196–1207, 2013, ISSN: 1084-8045. DOI: https://doi.org/10.1016/j.jnca.2013.02.008.

[18] I. F. Akyildiz, A. Lee, P. Wang, M. Luo, and W. Chou, "A roadmap for traffic engineering in SDN-OpenFlow networks", *Computer Networks*, vol. 71, pp. 1–30, Oct. 2014, ISSN: 13891286. DOI: 10.1016/j.comnet.2014.06.002.

[19] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, and A. Vahdat, "B4: Experience with a Globally-Deployed Software Defined WAN", in *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, ser. SIGCOMM '13,

New York, NY, USA: ACM, 2013, pp. 3–14, ISBN: 978-1-4503-2056-6. DOI: 10.1145/2486001.2486019.

[20] M. Dorigo and T. Stützle, *Ant Colony Optimization*. Scituate, MA, USA: Bradford Company, 2004, ISBN: 0262042193.

[21] A. D. Stefano, G. Cammarata, G. Morana, and D. Zito, "A4SDN - Adaptive Alienated Ant Algorithm for Software-Defined Networking", in *2015 10th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC)*, Nov. 2015, pp. 344–350, ISBN: 978-1-4673-9473-4. DOI: 10.1109/3PGCIC.2015.120.

[22] K. Deb, *Multi-Objective Optimization Using Evolutionary Algorithms*. New York, NY, USA: John Wiley & Sons, Inc., 2001, ISBN: 9780470743614.

[23] M. P. Kleeman, B. A. Seibert, G. B. Lamont, K. M. Hopkinson, and S. R. Graham, "Solving Multicommodity Capacitated Network Design Problems Using Multiobjective Evolutionary Algorithms", *IEEE Transactions on Evolutionary Computation*, vol. 16, no. 4, pp. 449–471, Aug. 2012, ISSN: 1941-0026. DOI: 10.1109/TEVC.2011.2125968.

[24] Y. Guo, Z. Wang, X. Yin, X. Shi, J. Wu, and H. Zhang, "Incremental Deployment for Traffic Engineering in Hybrid SDN Network", in *2015 IEEE 34th International Performance Computing and Communications Conference (IPCCC)*, Dec. 2015, pp. 1–8. DOI: 10.1109/PCCC.2015.7410320.

[25] Y.-S. Yu and C.-H. Ke, "Genetic algorithm-based routing method for enhanced video delivery over software defined networks", *International Journal of Communication Systems*, vol. 31, no. 1, e3391, Jan. 2018, ISSN: 10745351. DOI: 10.1002/dac.3391.

[26] M. T. M. Emmerich and A. H. Deutz, "A tutorial on multiobjective optimization: fundamentals and evolutionary methods", *Natural Computing*, vol. 17, no. 3, pp. 585–609, Sep. 2018, ISSN: 1572-9796. DOI: 10.1007/s11047-018-9685-y.

[27] E. A. K. Mishra, E. Y. Mohapatra, and E. A. K. Mishra, "Multi-Objective Genetic Algorithm: A Comprehensive Survey", *International Journal of Emerging Technology and Advanced Engineering*, vol. 3, no. 2, pp. 81–90, Feb. 2013.

[28] A. Zhou, B.-Y. Qu, H. Li, S.-Z. Zhao, P. N. Suganthan, and Q. Zhang, "Multiobjective evolutionary algorithms: A survey of the state of the art", *Swarm and Evolutionary Computation*, vol. 1, no. 1, pp. 32–49, 2011, ISSN: 2210-6502. DOI: https://doi.org/10.1016/j.swevo.2011.03.001.

[29] A. Konak, D. W. Coit, and A. E. Smith, "Multi-objective optimization using genetic algorithms: A tutorial", *Reliability Engineering & System Safety*, vol. 91, no. 9, pp. 992–1007, 2006, Special Issue - Genetic Algorithms and

Reliability, ISSN: 0951-8320. DOI: https://doi.org/10.1016/j.ress.2005.11.018.

[30] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II", *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, Apr. 2002, ISSN: 1089-778X. DOI: 10.1109/4235.996017.

[31] N. Srinivas and K. Deb, "Muiltiobjective Optimization Using Nondominated Sorting in Genetic Algorithms", *Evol. Comput.*, vol. 2, no. 3, pp. 221–248, Sep. 1994, ISSN: 1063-6560. DOI: 10.1162/evco.1994.2.3.221.

[32] K. Deb and H. Jain, "An Evolutionary Many-Objective Optimization Algorithm Using Reference-Point-Based Nondominated Sorting Approach, Part I: Solving Problems With Box Constraints", *IEEE Transactions on Evolutionary Computation*, vol. 18, no. 4, pp. 577–601, Aug. 2014, ISSN: 1941-0026. DOI: 10.1109/TEVC.2013.2281535.

[33] H. Jain and K. Deb, "An Evolutionary Many-Objective Optimization Algorithm Using Reference-Point Based Nondominated Sorting Approach, Part II: Handling Constraints and Extending to an Adaptive Approach", *IEEE Transactions on Evolutionary Computation*, vol. 18, no. 4, pp. 602–622, Aug. 2014, ISSN: 1941-0026. DOI: 10.1109/TEVC.2013.2281534.

[34] E. Zitzler, M. Laumanns, and L. Thiele, "SPEA2: Improving the strength Pareto evolutionary algorithm", *TIK-report*, vol. 103, 2001.

[35] T. H. Szymanski, "Max-Flow Min-Cost Routing in a Future-Internet with Improved QoS Guarantees", *IEEE Transactions on Communications*, vol. 61, no. 4, pp. 1485–1497, Apr. 2013, ISSN: 0090-6778. DOI: 10.1109/TCOMM.2013.020713.110882.

[36] J. Y. Yen, "Finding the K Shortest Loopless Paths in a Network", *Management Science*, vol. 17, no. 11, pp. 712–716, 1971. DOI: 10.1287/mnsc.17.11.712.

[37] T. Eilam-Tzoreff, "The disjoint shortest paths problem", *Discrete Applied Mathematics*, vol. 85, no. 2, pp. 113–138, 1998, ISSN: 0166-218X. DOI: https://doi.org/10.1016/S0166-218X(97)00121-2.

[38] Szcześniak, Irek, *Yen k-shortest paths*, (2019, Sep 26). [Online]. Available: https://github.com/iszczesniak/yen (visited on 09/26/2019).

[39] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. MIT press, 2009.

[40] M. B. Cohen, Y. T. Lee, and Z. Song, "Solving linear programs in the current matrix multiplication time", *CoRR*, vol. abs/1810.07896, 2018. arXiv: 1810.07896. [Online]. Available: http://arxiv.org/abs/1810.07896.

[41] M. Li, A. Lukyanenko, Z. Ou, A. Ylä-Jääski, S. Tarkoma, M. Coudron, and S. Secci, "Multipath Transmission for the Internet: A Survey", *IEEE Communications Surveys Tutorials*, vol. 18, no. 4, pp. 2887–2925, 2016, ISSN: 1553-877X. DOI: 10.1109/COMST.2016.2586112.

[42] M. Laor and L. Gendel, "The Effect of Packet Reordering in a Backbone Link on Application Throughput", *IEEE Network*, vol. 16, no. 5, pp. 28–36, Sep. 2002, ISSN: 0890-8044. DOI: 10.1109/MNET.2002.1035115.

[43] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling Innovation in Campus Networks", *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008, ISSN: 0146-4833. DOI: 10.1145/1355734.1355746.

[44] P. Medagliani, J. Leguay, M. Abdullah, M. Leconte, and S. Paris, "Global Optimization for Hash-Based Splitting", in *2016 IEEE Global Communications Conference (GLOBECOM)*, Dec. 2016, pp. 1–6. DOI: 10.1109/GLOCOM.2016.7841861.

[45] D. Tuncer, M. Charalambides, S. Clayman, and G. Pavlou, "Flexible Traffic Splitting in OpenFlow Networks", *IEEE Transactions on Network and Service Management*, vol. 13, no. 3, pp. 407–420, Sep. 2016, ISSN: 1932-4537. DOI: 10.1109/TNSM.2016.2580666.

[46] A. Ford, C. Raiciu, M. Handley, S. Barre, and J. Iyengar, "Architectural guidelines for multipath TCP development", Tech. Rep. 6182, Mar. 2011, (2017, Sep 13). [Online]. Available: https://tools.ietf.org/html/rfc6182.

[47] O. Bonaventure and S. Seo, *Multipath TCP Deployments*, (2019, Dec 11), 2016. [Online]. Available: https://www.ietfjournal.org/multipath-tcp-deployments/ (visited on 12/11/2019).

[48] S. Zannettou, M. Sirivianos, and F. Papadopoulos, "Exploiting Path Diversity in Datacenters Using MPTCP-aware SDN", in *2016 IEEE Symposium on Computers and Communication (ISCC)*, Jun. 2016, pp. 539–546. DOI: 10.1109/ISCC.2016.7543794.

[49] *GLPK - (GNU Linear Programming Kit)*, (2017, Mar 04). [Online]. Available: https://www.gnu.org/software/glpk/ (visited on 03/04/2017).

[50] *LEMON Graph Library*, (2017, Mar 04). [Online]. Available: http://lemon.cs.elte.hu/trac/lemon (visited on 03/04/2017).

[51] F.-A. Fortin, F.-M. De Rainville, M.-A. Gardner, M. Parizeau, and C. Gagné, "DEAP: Evolutionary Algorithms Made Easy", *Journal of Machine Learning Research*, vol. 13, pp. 2171–2175, Jul. 2012.

[52] *Mininet*, (2019, Nov 11). [Online]. Available: http://mininet.org/ (visited on 11/11/2019).

[53] *Omnet++ discrete event simulator*, (2020, Jan 05). [Online]. Available: https://omnetpp.org (visited on 01/05/2020).

[54] *Ns-3*, (2017, Mar 04). [Online]. Available: https://www.nsnam.org/ (visited on 03/04/2017).

[55] E. Jo, D. Pan, J. Liu, and L. Butler, "A Simulation and Emulation Study of SDN-based Multipath Routing for Fat-Tree Data Center Networks", in *Proceedings of the Winter Simulation Conference 2014*, Dec. 2014, pp. 3072–3083. DOI: 10.1109/WSC.2014.7020145.

[56] B. A. Forouzan, *Data Communications and Networking*, 5th ed. McGraw-Hill, 2013, ISBN: 978-0-07-337622-6.

[57] T. Henderson, S. Floyd, A. Gurtov, and Y. Nishida, "The NewReno Modification to TCP's Fast Recovery Algorithm", Tech. Rep. 6582, Apr. 2012, (2020, Jan 23). [Online]. Available: https://tools.ietf.org/html/rfc6582.

[58] *GÉANT Topology Map*, (2017, Mar 08). [Online]. Available: https://www.geant.org/Resources/Documents/GEANT_topology_map_jan2017.pdf (visited on 03/08/2017).

[59] *GÉANT Connectivity Map*, (2018, Jul 07). [Online]. Available: http://map.geant.org (visited on 07/07/2018).

Appendix A

GÉANT Network Link Delays

Table A.1: 2017 GÉANT Network Link Delays

Node 1	Node 2	Delay (ms)	Node 1	Node 2	Delay (ms)
IS	UK	34.4	IT	AT	11.4
IS	DK	38.3	IT	MT	21.0
IE	UK	8.50	IT	GR	26.7
IS	UK	34.4	CZ	PL	5.70
UK	PT	28.9	CZ	HU	8.10
UK	IL	64.9	AT	GR	23.4
UK	CY	58.7	AT	SK	1.00
UK	BE	5.80	AT	RO	15.6
UK	FR (upper)	6.30	AT	HU	3.90
PT	ES	9.20	AT	BG	14.9
ES	FR (upper)	19.2	AT	HR	4.90
ES	FR (lower)	14.9	AT	SI	5.10
FR (upper)	CH	7.50	SI	HR	2.10
NL	DK	11.3	HR	HU	5.50
NL	DE (upper)	6.70	SK	HU	2.90
NL	LU	5.40	ME	HU	10.3
NL	BE	3.20	HU	RO	11.7
LU	DE (lower)	3.40	HU	BG	11.5
FR (lower)	CH	6.00	HU	TR	19.5
FR (lower)	IT	7.10	HU	RS	5.80
NO	DK	8.80	PL	LT	9.80
NO	SE	7.60	PL	BY	13.3
DK	SE	9.50	PL	UA	17.6
DK	DE (upper)	5.30	EE	LV	5.10
SE	FI	7.20	LV	LT	4.20
CH	DE (lower)	8.60	MK	BG	3.20
CH	IT	4.60	BG	RO	5.40
DE (lower)	DE (upper)	7.20	RO	MD	6.50
DE (lower)	PL	11.4	DE (lower)	AT	10.9
DE (lower)	CY	47.3	DE (lower)	TR	34.1
DE (lower)	CZ	7.50	DE (lower)	IL	53.6
DE (lower)	HU	14.8	DE (upper)	EE	20.3
DE (lower)	NL	6.60			