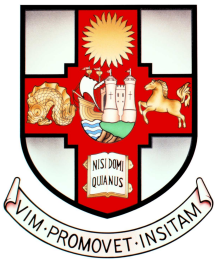
Author:
Hollinghurst, Joe

Title:
Enabling Software Defined Networking in High Criticality Networks

Enabling Software Defined Networking in High Criticality Networks

By

Joseph Hollinghurst



Department of Electrical and Electronic Engineering
University of Bristol

A dissertation submitted to the University of Bristol in accordance with the requirements of the degree of Doctor of Philosophy in the Faculty of Engineering.

September 2017

Word count: forty-five thousand six hundred and ninety-two

Abstract

High-criticality networking solutions are often dedicated, highly specialised, even bespoke in case of hard real-time guarantees. This is required to ensure (quasi) deterministic behaviour of the network services as seen by critical applications. However, dedicated networks incur significant expense, along with the inability to update the system efficiently and effectively. Software-Defined Networking (SDN) uses controllers to allow dynamic, user-controlled, on-demand configuration of the network. This provokes interesting questions on the applicability of SDN concepts and architectures in high-criticality networks.

Although SDN offers flexibility and programmability to the network infrastructure through the introduction of a controller, the controller introduces extra delay into the system. This is due to new flows querying the controller for instructions of how to route traffic. This becomes an increasing problem for large scale and delay sensitive networks such as those found in high-criticality infrastructure. The delay introduced can be minimised by optimal placement of the controller or decreased further by introducing additional controllers. Although the problem of optimal placement for multiple controllers is known to be NP hard, approximations can be used. The analysis of three different methods has been conducted and investigates the scalability, and how the accuracy of the methods varies with the complexity.

In the latter stage of the thesis the use of redundancy and coding is analysed with the aim to reduce latency and increase reliability within the network. The objective is to provide an analysis of the gains achievable through the use of redundant messages and coding. Both redundancy and coding increase the network load and hence the delay of each packet, but can reduce overall delay by exploiting independent randomness across multiple paths. Both the average delay minimisation and probabilistic guarantees on delay exceeding some tolerance threshold are considered.

Dedication

To my Wife, Robyn Hollinghurst née Davies.

Declaration

I declare that the work in this dissertation was carried out in accordance with the requirements of the University's Regulations and Code of Practice for Research Degree Programmes and that it has not been submitted for any other academic award. Except where indicated by specific reference in the text, the work is the candidate's own work. Work done in collaboration with, or with the assistance of, others, is indicated as such. Any views expressed in the dissertation are those of the author.

SIGNED: ..... DATE:.....

Acknowledgements

I would like to thank my long standing supervisors Ayalvadi Ganesh and Tim Baugé for their dedication and support throughout my time at the University of Bristol.

This work was supported by the Engineering and Physical Sciences Research Council [grant numbers EP/I028153/1]; Thales UK; and the University of Bristol. I would like to thank Thales UK Research & Technology for supporting this research through an EPSRC CASE Award

Contents

List of Figures

Chapter 1

Introduction

1.1 Introduction to SDN and NFV

Networking has often lagged behind the pace of innovation within the rest of the computing world. This is largely due to the fact that networking elements are traditionally operated in hardware rather than software, and therefore the time to market is much slower. A conventional networking element consists of both a control plane and a data plane. The control plane can be seen as the 'brain' of the network, which dictates how and where data is sent. The data plane then forwards the traffic according to the decision the control plane has made. As each element has both a control and data plane it is difficult to influence the behaviour of the traffic as each node is preprogrammed, and if a new application is required each element must be individually updated in a distributed manner.

On the other hand, Software Defined Networking (SDN) separates the control and data planes by introducing a controller. The controller can be used as a centralised resource to disseminate instructions to the data plane as shown in figure 1.1. This allows the networking elements to be easily reprogrammable whilst maintaining the necessary line speed for communications.

Another benefit of SDN is the abstraction of the underlying infrastructure. This allows rapid innovation in networking functions as network engineers do not need to be concerned with the hardware involved. Furthermore,

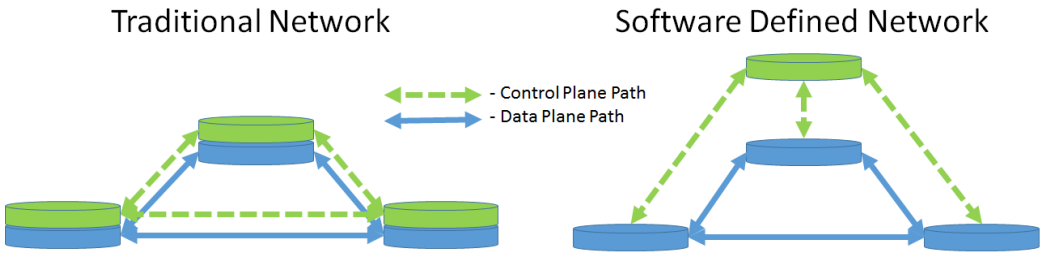


Figure 1.1: Traditional Networking Architecture vs. SDN Architecture

by using abstractions the underlying network can be heterogeneous in nature. This enables the use of COTS (commercial off the shelf) hardware components that can be interchanged dynamically within a network without the need to reconfigure. This also contributes to the reduction in 'vendor lock in', where by companies with existing networking infrastructure are forced to purchase products from a specific manufacturer due to compatibility issues.

Alternatively, the SDN movement aims to be mainly open-source. This is largely due to the ONF [1](Open Networking Foundation), OpenDaylight and importantly the OpenFlow API (application programming interface). Although OpenFlow is not the only API, it is the one most widely used. It is therefore said that within the definition of SDN, OpenFlow is either considered to be equal, or at least a major part of the SDN movement[2].

Network Function Virtualisation (NFV) is another aspect of SDN, NFV is the idea that network functions are written in software (virtualised), rather than requiring dedicated hardware. SDN and NFV are often banded together, as they are normally used in collaboration: SDN provides the underlying infrastructure, whilst NFV provides the networking functions. This has the advantage of being able to use commercially available hardware combined with software for specific purposes such as firewalls and traffic management.

Due to the vendor-neutrality and open source nature of SDN the cost of setting up and running a network can be reduced. First of all, businesses providing networks can purchase hardware at a significantly reduced cost, and the business focus then transfers to the services that the networks provide. Secondly, businesses requiring networks benefit from the reduced cost of the

hardware, and also the programmability of the networks means the networking solution can be tailored to the business needs, resulting in a much better service. Furthermore, the centralised nature of SDN means that performance metrics and statistics can easily be collected, making visibility much clearer. Ultimately, this makes the network management easier, which in turn means the network can be adapted to business needs and consequently the efficiency is improved.

1.2 Industrial Motivation and High Criticality

The industrial motivation behind this thesis is to exploit the programmability of SDN within industry. The aim is to allow industries, other than hardware manufacturers, to provide specialist networking services. This is enabled by combining intellectual property with the heterogeneous, open source, programmable networks that SDN creates.

Throughout this thesis we refer to high criticality networks. High criticality networks are a subset of industrial networks, where the Quality of Service (QoS) requirements differ substantially to those required in commercial/enterprise networks.

A thorough explanation of the difference between enterprise (commercial) networks and high criticality (industrial) networks is provided in [3]. In the paper the authors explain the key differences between the primary functions of the networks, with high criticality networks often being used to control physical equipment. On the other hand, commercial networks are mainly used for data processing and transfer. This change in primary function causes a substantial difference in the QoS requirements. For example if a failure occurs in a high criticality network the severity is much higher than a commercial network as people's safety may be at risk. Therefore high criticality networks often require backup systems.

The reliability of the network is also more important in a high criticality network. In a commercial network if data isn't received it can often be

re-sent without any ramifications. However, for a high criticality network the data may correspond to a warning message from a sensor, which may cause substantial harm if it is not received. Similarly the speed at which the message is received may be critical in this scenario, with the requirement for the round trip time in industrial networks ranging from $250\mu s$–$10ms$ (compared to $50ms+$ in conventional networks). Hence, the reliability and round trip time requirements are substantially stricter in a high criticality network.

The type of traffic also differs between the network types, with high criticality networks often consisting of deterministic periodic and aperiodic small packets. Whereas commercial networks consist of larger aperiodic traffic, that is often random or consists of bursts.

To satisfy the performance demands, high criticality infrastructure traditionally uses bespoke networking solutions which are often expensive and inflexible in nature. This is an interesting contrast to SDN where COTS hardware is used, which is less expensive, and relies on the programmability of SDN to provide the relevant features. Also, the use of COTS hardware adds uncertainty to the network as it is unknown whether the hardware components can maintain the required performance for high criticality. Thus, the motivation raises the research question of *can a Software Defined Network be tailored to satisfy the needs and demands of a high criticality network?*

Although the motivation casts a wide net over the potential research areas, there are two particular areas that are covered in this thesis. The two areas consist of the controller placement problem and data redundancy. The research areas were chosen by identifying potential extensions to research in the current literature. This led to the contributions highlighted in section 1.4 which aim to aid the application of SDN in high-criticality.

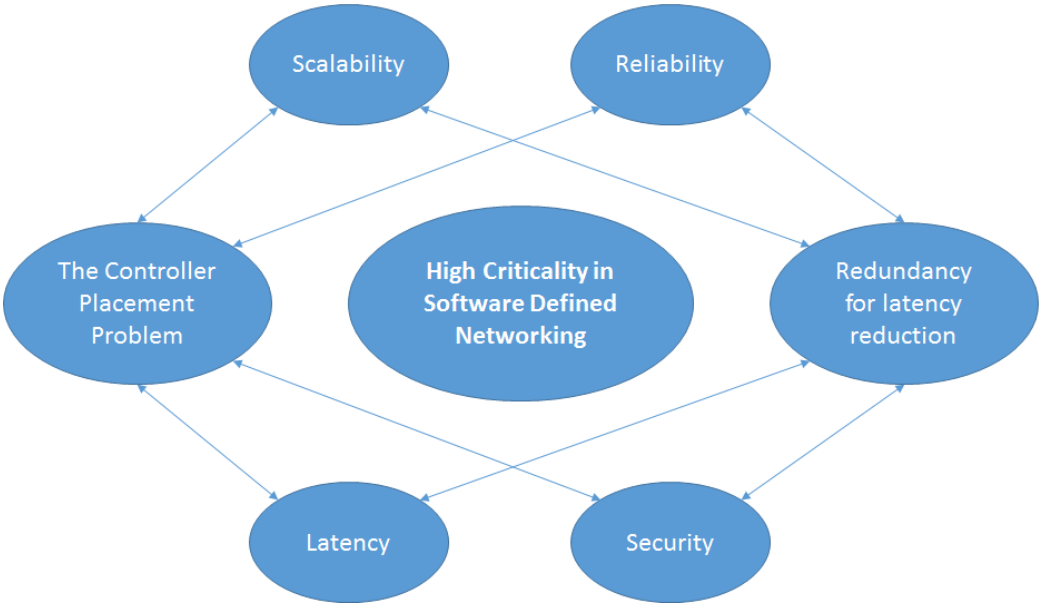


Figure 1.2: The links between aspects of High Criticality and SDN

1.3 Thesis Organisation

Figure 1.2 represents the most important elements involved in high criticality. The figure shows the two main research areas are linked to all the aspects identified. First of all a comprehensive literature review on SDN is presented. This was used to identify possible research questions in the area of high-criticality which led to the research in Chapter 3. The research in these chapters links to a physical aspect of SDN, the controller placement, rather than any virtual aspects. Whilst undertaking the work on the controller placement it was found that there is limited research in the modelling of networks using mathematical techniques, due to the fact that simulation and emulation is often used instead. Therefore another literature review was undertaken to establish potential areas of further work. An area of research that links to high-criticality is the use of redundancy in networks and this is investigated in Chapters 5 and 6. Finally, conclusions and areas of further work are presented in Chapter 7.

1.4 Contributions

The following papers contain many of the results seen in this thesis:

J. Hollinghurst, A. Ganesh, and T. Baugé, 'Controller placement methods analysis', in Information Communication and Management (ICICM), International Conference on. IEEE, 2016, pp. 239-244.

J. Hollinghurst, A. Ganesh, and T. Baugé, 'Latency Reduction in Communications Networks Using Redundant Messages', in the twenty-ninth International Teletraffic Congress (ITC 29), IEEE , 2017.

Chapter 2

Controller placement: Literature

2.1 Introduction to Literature Review

Chapter 3 investigates *The Controller Placement Problem*. The controller placement problem was first formulated in [4], where the authors investigate where an SDN controller should be placed, and how many controllers are needed to support the network. These questions are supported by the work in section 2.2 as the reviewed papers highlight the current challenges in SDN that can be alleviated by better placement of the SDN controller. The challenges include: Latency, Security, Reliability and Scalability, and are investigated in detail in section 2.2.

As we are investigating the controller placement problem the latency in this chapter refers to both controller-to-controller and controller-to-node communication. Since the notion of a controller is introduced in software defined networking these two parameters are SDN specific. The controller-to-controller delay is important to keep network consistency, since an outdated database can cause the network to give false forwarding instructions. The controller-to-node delay is important to maintain an agile, adaptive and dynamic network, which is a key aspect of what SDN wants to deliver. This is because the lower the latency, the quicker the response time can be be-

tween a packet arriving at a node and control instructions being given. A lower latency also has advantages when looking at the security and reliability of the network. The security can be improved with good placement of the controllers as forwarding instructions can be updated quicker, so any threats can be dealt with as soon as possible. More specifically the controller itself is a target for attack since it is a centralised system. This leads to the introduction of middle-boxes, which add complexity to the controller placement as extra nodes are added to the network. With respect to reliability, it is important that the network has a reliable control element for continuous service. To that end, the controller placement problem can be altered to take in to account the connectivity and probability of failure for components in the network. This changes the objective function of the problem formulation and introduces new metrics to quantify reliability in SDN. Finally, the scalability of SDN is considered as, depending on the size of the network, it is unknown if a centralised system can cope with the amount of flows querying the controller. In terms of the controller placement problem this can be alleviated by introducing multiple controllers to balance the number of flows on a per controller basis.

2.2 Problems in SDN

2.2.1 Latency

The latency is considered both physically with the placement of the SDN controllers and virtually for the data transmission within the network. Only the controller delay is discussed in this section as this links to the controller placement in Chapter 3. The literature that links latency and redundancy is presented in Chapter 4 as the research composed is discussed in Chapters 5 and 6. Latency is a key component in the research conducted in this thesis due to the importance latency has in high-criticality. In particular, it is often the case that a specific latency must be adhered to depending on the scenario [5]. One example that highlights this is the use of wireless sensor networks in industry, where an increased latency can cause outdated information, which

may lead to wrong decisions being made [6].

Delay to the controller

In a Software Defined Network extra latency is introduced when communicating with a central controller. This is because any unknown packet, flow or event must first query the controller for forwarding instructions. An acceptable level of latency therefore needs to be adhered to in order to avoid interruption or excessive set-up times. The controller placement can help by increasing the amount of controllers, or placing the controllers in close proximity to the forwarding elements (data plane). This gives rise to the questions of how many controllers are needed and where should they be placed? These questions were first asked in the controller placement problem [4]. In [4] the authors look at both where the controller should be placed, and how many controllers are needed to support the network. The metric used to do this was the node-to-controller latency. The authors consider both the average latency between all the nodes and their corresponding controller and the worst case latency where the maximum latency is found between all the nodes and their controllers. Many topologies were considered in the analysis provided by the internet topology zoo[7]. It was found that optimal placement of the controller was normally in densely populated areas of nodes for the average latency, and in the centre of the network for the worst case delay. Furthermore, with these topologies the authors conclude that one controller often suffices to meet required delay bounds. However, this is a generalisation and may not necessarily be the case in networks with specific delay bounds.

Delay to the controller is also discussed as a metric in [8] where the physical delay is considered due to the distance between the controllers and nodes. The node-to-controller delay is considered negligible in data centres due to close proximity of the controllers to the nodes. Whereas, for service providers, the geographic distribution leads to the metric being much more important as the physical delay is much larger. The controller placement is therefore particularly important in WAN (Wide Area Network) environments due to the physical constraints, this is an area researched in [9], which is

discussed in section 2.3.2. A larger geographic area can also mean a larger network in terms of the number of nodes, and this relates to the scalability of the placement algorithms which is investigated in Chapter 3.

Flow Set-up Time

The flow set-up time is an SDN specific parameter, and refers to the additional overhead required to initially send and receive forwarding instructions between the controller and the forwarding elements. It is claimed that this is exacerbated by the current hardware[10]. The hardware can be a problem for the system due to lower performance CPU's used for control applications in switches [10]. This leads to a poor response time when a large number of initial commands are sent from the controller. However, it is noted that this will be dealt with by the switch vendors when making SDN specific switches. Irrespective of the hardware used, the use of multiple controllers can minimise the risk of the flow set-up becoming an issue by spreading the initial load on each controller. It is therefore important to look at the placement of multiple controllers to make this possible.

Furthermore, it is shown in [10] that a proactive approach offers better performance, particularly at scale. The performance is quantified in [11] by considering the number of flows/sec that a controller can handle as a function of the number of switches in the network. The results are shown in Figure 2.1 which indicates that a proactive controller can handle a larger number of flows/sec for every network size (number of switches). Although the proactive approach is shown to give better performance, this is at the cost of increased complexity. It is also noted that although a proactive approach is more effective, reactive operations would still need to be in place for unpredictable events. In order to have an effective proactive network there needs to be adequate controller to controller communication. This is a consideration in Chapter 3, whereby the optimisation metric can be altered to keep the controllers closer together. The idea of a proactive network is also a consideration when looking at controller placement for resilience. In this case the placement of controllers is computed in such a way that if a failure

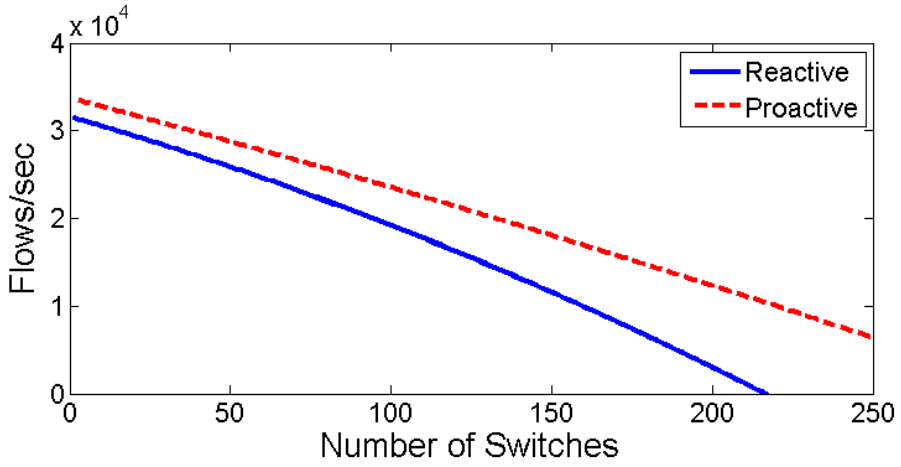


Figure 2.1: The number of achievable flows/sec for Proactive and Reactive controllers

occurs control can be maintained without disruption, and hence the set-up does not need to be re-initiated. This issue closely relates to that of fast failure recovery, as discussed in [12], which compares a proactive approach to failure recovery to that of a reactive approach using NOX [13].

Within some of the monitoring tools such as OpenTM[14] and when looking at emulation[15], the performance is calculated based upon steady state behaviour and ignores flow set-up. This is because a proactive set-up is assumed or reactive set-up is disregarded. This assumption could significantly impact performance as an initial set-up is likely to require the most resources at any one time. This is because even though a proactive approach can alleviate much of the initial burst (flow set-up) of requests for forwarding instructions, a reactive approach means the flow set-up time can be much larger as the requests would have an element of queuing. One way to get the best of both worlds is to use hybrid systems, such as the ones considered in [11].

2.2.2 Security

Although security is not an immediately obvious aspect of SDN related to the placement of controllers, the use of middle boxes and backup controllers

can effect the way the placements are calculated. The use of middle boxes in SDN is considered in the security survey by Scott et. al [16]. In an SDN environment the controller would be able to control and install new rules on middle boxes. By doing this it would take some computational complexity away from the controller, as flow anomaly detection could be handled by the middlebox. However, the network will become more complex due to the additional nodes in the system which makes the controller placement more difficult. The survey gives a detailed account of the current and potential security threats in SDN and provides both positive and negative viewpoints. In particular the work in [17] highlights the two main security threats: falsifying a controller and Denial-of-Service (DoS) attacks. Within the survey however security is mainly handled through anomaly detection, as when new flows are sent to the controller, the controller can then decide whether or not a flow is malicious. This introduces the problem of DoS attacks through flooding the controller with new flows. A software approach to this problem is FortNOX [18], which is a controller software package that detects threats and conflicts in flows. The metric used to evaluate the performance of the tool is the delay introduced, which has an average 7ms when analysing 1000 rules. This delay may be required due to the secure nature of the network, but careful consideration of how much delay a network can tolerate needs to be taken into account. A hardware approach however is the use of backup controllers, which is also a consideration with reliability, this is because if a controller fails or is attacked, a continuous service can still be maintained. The use of backup controllers adds another element to the controller placement problem, and this is the focus in [19], which is reviewed in section 2.3.1.

2.2.3 Reliability

Reliability is an aspect of networking that needs to be addressed for both centralised and distributed systems. It is argued that a distributed system improves the physical reliability [20] as clusters of off the shelf components can be used, which are easy to replace if a failure occurs, and hence con-

sistency can be maintained. On the other hand, a centralised system will improve reliability at a software level due to: automated management of network devices, uniform policy enforcement, and fewer configuration errors [21]. The fact that a distributed network is possible leads to the need for resilient controller placement, this is a problem addressed through the use of alternative metrics in the problem formulation. Furthermore, the centralised nature of SDN may improve the reliability at a software level, but this needs consistency amongst controllers if more than one is used in the network. This relates to the convergence of controller consistency, and can only be achieved if the latency of controller-to-controller communication is adequate. Reliability can also refer to the resilience of the network, this means that fault tolerance needs to be achieved, and if failures occur a suitable strategy needs to be adopted to maintain connectivity, or at least provide a quick response.

Fault tolerance is a big issue with reliability and convergence must take place within specified time frames to maintain network connectivity. The controller placement is therefore an important factor to ensure the convergence time is possible. In [22] a QoS-aware network operating system is created, of which fault tolerance is a major part. A 50ms time frame needs to be adhered to in order to maintain carrier grade fault tolerance [23]. This is achieved using their operating system, but only for a limited amount of forwarding elements and links. They conclude that in order to maintain QoS for a carrier grade network they can only accommodate 50 forwarding elements (FEs) and 200 links. These figures are arrived at based on how long it takes for a fault notification to reach the controller (1.85ms-2ms) and how long it takes to recalculate a route, which is ultimately dependent on the size of the network and the controller placement.

Ethane [24] introduces backup controllers for centralised systems. Within the paper the authors describe three different methods for replicating the controller. Cold-standby (booted when needed and spanning tree computed when activated), warm-standby (booted when needed but spanning tree preconfigured) and fully replicated (separate spanning tree for each controller but flow-requests can be spread across them). Each method thus has a different recovery time. The paper also looks at how link failures effect the system

and it was shown that a path converges in less than 40ms with cold-standby, although the packets may be delayed by around a second due to the controller handling a flurry of requests. The use of backup controllers leads to better convergence times, but this also creates the problem of how to generate their placements. One method for creating a list of backup controllers is presented in [19], but to save replication this is discussed in section 2.3.1.

In summary, one focus on reliability is fault tolerance, which can be characterised through the connectivity and probability of failure within a network. The SDN approach gives the possibility of using proactive backup controllers which can reduce convergence time in a network (CPRecovery [25] is one such example). Also, due to the programmability and network wide view of the controller, new paths can be created for node failures whilst avoiding loops and black holes, which is an inherent problem in traditional networks. As reliability is a major concern in high criticality applications a proactive approach to back up would be advantageous as this would ensure functionality. Furthermore, the controller placement can be influenced to benefit reliability, and this can easily be made an additional consideration by altering the objective function.

2.2.4 Scalability

Many software applications have been developed to try and combat scalability in SDN. Publications look at how to scale a network with little overhead, or create a distributed system, which may require multiple controllers. SDN also enables the possibility of developing new metrics for scalability. As well as traditional metrics such as delay and throughput, parallel aspects can be considered such as manageability(how easy it is to add nodes) and functional scalability (how easy it is to add new functions) to the network[8]. The SDN specific metrics can then be used in the controller placement objective function as required. Scalability also has to be considered when looking at the controller placement algorithms, this is because many of the available algorithms are extremely computationally heavy, and can only be used on networks of a limited size. The problem of scalability for the algorithms is

expanded in Chapter 3, where it is shown that approximations must be used when considering larger networks.

One way to address scalability in SDN is to reduce the number of controllers required. DevoFlow [26] aims to do this by limiting the data sent to the controller using various techniques. These include: rule cloning, local actions and efficient statistics collection. The performance is evaluated according to the aggregate throughput of all the flows in the network. The techniques used generally improved throughput, however, increased throughput is not the main function of DevoFlow, reduced flow to the controller is. When looking at the number of packets sent to the controller and the amount of flow entries needed it is shown that DevoFlow reduces both metrics significantly compared to the original OpenFlow procedures.

DIFANE [27] attempts a hierarchical system to dilute the amount of flows going directly to the controller. It does this in the following way (numbers match operations in Figure 2.2):

1. The controller installs the flow rules on to the authority switches.
2. A packet arrives but has no matching rule in the switch.
3. The packet is forwarded to the authority switch.
4. The rules are updated on the ingress switch and the packet is forwarded directly to the egress switch.
5. Subsequent packets arrive, matching the flow rule.
6. The packet is forwarded according to the rule previously installed by the authority switch.

DIFANE distributes the authority switches whilst having a single controller, hence the system maintains a global view. The delay to obtain a flow rule is therefore reduced as the authority switches are physically closer to the ingress/egress switches. The amount of flow going to the controller is also reduced, as new rule requests only go to the authority switch, which

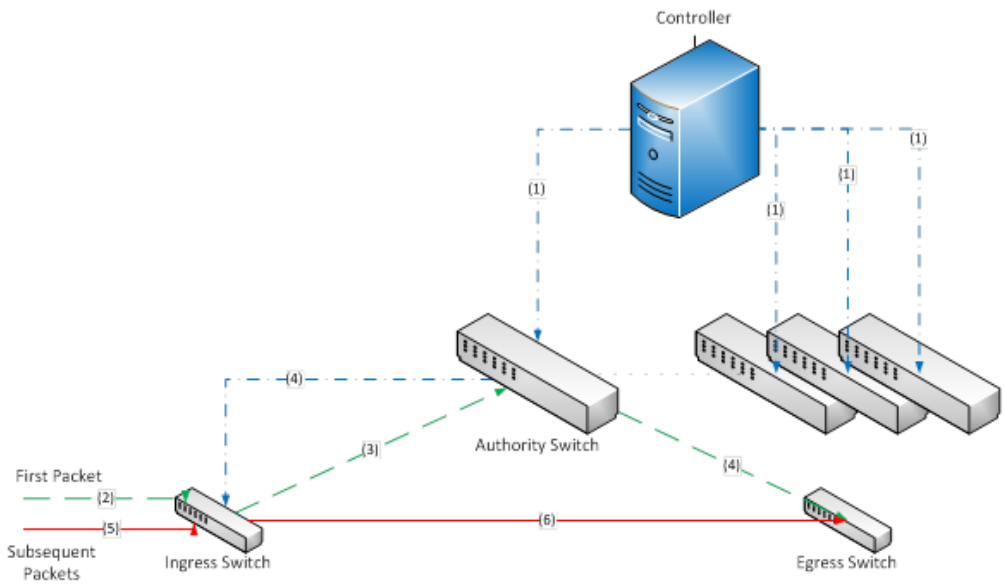


Figure 2.2: Basic Operation of DIFANE

leads to enhanced scalability. A fundamental distinction between the controller and switch is maintained as the controller is the only entity which can create rules and the authority switches only distribute them. This creates a sub-problem for the controller placement problem since now the authority switches also need to be positioned. This could be solved by allowing the controller placement problem to also have a hierarchical solution, by first finding the placements of authority switches with respect to the forwarding nodes, and then finding the controller placements with respect to the authority switches. However, this would also mean that more than one problem would need to be solved, which may cause excessive computation depending on the algorithm used.

In [28] & [29] the authors take an approach to the scalability by looking at how a controller can be logically centralised. They look at the trade-offs in both optimality versus staleness and robustness to inconsistency versus application complexity. When analysing the trade-offs both a strongly consistent network view, and an eventually consistent view are considered. The strongly consistent view gives more accurate data but with an increase in overhead, leading to increased delay, which can cause suboptimal perfor-

mance. Whereas, the eventually consistent view uses data as it becomes available, thus reacting quicker, but it may lead to incorrect behaviour. The authors reach the conclusions that staleness significantly impacts optimality and robustness to inconsistency increases when the application logic is aware of the flow duration. The question of staleness also links to whether or not a controller placement should be dynamic or fixed. A fixed controller placement means that a more complex algorithm can be used, which would generally give a more optimal solution at the time of computation. On the other hand, if a solution needs to be computed more than once the complexity is a limiting factor, and faster algorithms, which are not necessarily as good in terms of optimality, must be used.

HyperFlow [30] looks to combat the problem of scalability by using multiple physically distributed controllers and then synchronising them. The paper looks at a way to efficiently communicate network state information between controllers. It does this by making the following assumptions: any change in the network wide view stems from the occurrence of a network event, the evidence in [13] that only a fraction of network events change the network-wide view (tens of events per second for networks of thousands of hosts),ordering of events does not matter and applications only need to be minimally modified. Using these assumptions they are then able to use a publish/subscribe system called WheelFS[31] in order to efficiently synchronise controllers. The synchronisity of controllers can also be a consideration in the placement, since controllers can be forced to be physically placed closer together as in Chapter 3. The authors also make an important claim that HyperFlow enables OpenFlow deployment in mission critical networks, as the network can increase or decrease the number of controllers to satisfy the demands of the network.

Another way to solve the controller placement problem and maintain scalability is to virtually slice the network. A controller can then be assigned to each slice of the network as with FlowVisor [32]. A slice is configured by determining and enforcing which slice each packet belongs to. FlowVisor is also able to isolate bandwidth within each slice and the forwarding decisions for each slice are independent for each controller. This benefits the network by

allowing it to determine how much bandwidth a particular flow can receive, and is an important enabling feature when using redundancy as in Chapter 5. The authors also show that with message throttling isolation FlowVisor is able to prevent saturation of the CPU in the controller. However, the benefits come at the cost of increased delay when a new flow is determined.

An important metric to consider when deciding how many controllers are needed is the amount of flows in a network. A method to try and balance flows amongst controllers is BalanceFlow[33], which has distributed controllers with one 'super controller'. Each controller contains an N x N matrix, N being the number of switches it services, where the i, j^{th} (i^{th} row, j^{th} column) entry represents the weighted average number of flow requests from node i to node j. The super controller aggregates all the controller matrices and if flow imbalance is detected, according to a threshold value dependent upon the network, reallocates flows according to a cost function. The cost function takes into account: number of flow requests already handled by the controller, the flows about to be handled and the propagation delay. The propagation delay is important as if flows were allocated to controllers with a large propagation delay this could cause adverse effects for fault tolerance and reaction delay. The propagation delay is therefore weighted in the cost function with respect to the total number of flow requests in the network divided by the minimum average node-to-controller latency and a scalar parameter which was found empirically. The only disadvantage to BalanceFlow is an increased overhead, however the amount of overhead is not discussed in the paper. Overall, BalanceFlow provides insight in to fundamental SDN metrics, such as propagation delay, which can be used to create a mathematical formulation for the controller placement problem.

Kandoo [34] limits the amount of flow going to the root controller by physically placing local controllers near to the switches as depicted in Figure 2.3. The local controllers then handle all the flows which only require local information, which are frequent. Whereas, the root controller only handles flows that require a network wide view such as rerouting. By doing this the authors show that the messages per second received by the controller is reduced by an order of magnitude. Another advantage is that the Kandoo

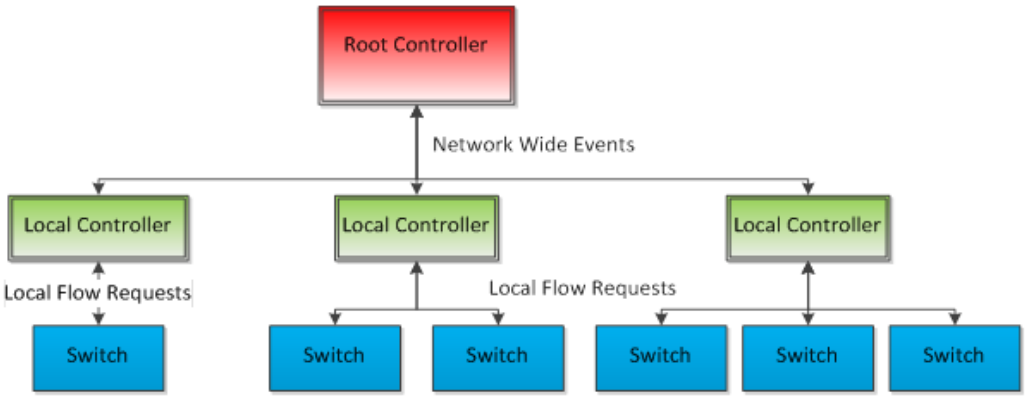


Figure 2.3: Kandoo Hierarchy

hierarchy can be combined with FlowVisor to further improve scalability. Although the approach of Kandoo is similar to that of DIFANE and DevoFlow the authors claim it is different as it does not extend switch capabilities, but it brings the control plane closer to the switches.

Scalability is combated with two main types of solution, by reducing the amount of flows going to the controller and second, by the introduction of more controllers. Each idea is fundamentally the same, to reduce the amount of flow on a per controller basis. A hybrid approach could therefore potentially be realised by combining techniques such as the ones in DevoFlow with a hierarchical controller system such as Kandoo. Whether a network has a single or multiple controllers a control platform is required as an enabler, one example is Onix [35], which allows a distributed system to appear centralised by maintaining a Network Information Base (NIB). It should also be noted that in a hierarchical system it is possible to add more controllers to satisfy specific requirements, such as delay bounds and reliability measures. A disadvantage to this approach is the increased cost of additional controllers but in order to adhere to specific requirements this may be necessary in an SDN system.

2.2.5 Metrics in Software Defined Networks

This section briefly discusses the metrics that are currently being used for monitoring purposes in SDN. The metrics are important, as it shows what characteristics of networking can be measured, and therefore optimised or improved. In relation to Chapter 3 the metrics are a useful consideration in to what factors can be included in an objective function for the controller placement. Table 2.1 lists the metrics identified in [36], the table also shows which metrics each monitoring software tool uses. A common problem amongst monitoring software is identified in the paper, in that they only use a limited amount of metrics when comparing networks. This is also shown by the fact that although there are a large number of metrics identified only six are ever evaluated.

2.2.6 Summary of the SDN literature

The chapter started by taking an in depth view of the problems SDN faces and analysing the current methods used to tackle certain problems. By undertaking the literature review it became clear (to the extent of the knowledge gained) that no methods currently exist to answer the question of whether or not SDN is an applicable technology for high-criticality networks. However, by combining the ideas and concepts of the methods within the chapter it may be possible to create a system with specific guarantees allowing SDN to be exploited for the use in high-criticality situations.

The monitoring tools and metrics for SDN are in a primitive stage. This causes the problem of having no standard metrics for comparison. However, data collection is fundamental in traffic engineering, irrespective of the metric used. The lack of standardisation is therefore not a big problem as long as the metrics are useful for their specific tasks.

The delays introduced by querying the controller and the flow setup time are SDN specific metrics and are therefore important in this thesis. This is why the controller placement is a critical application to be considered, as the aim for optimal placement of the controllers is to minimise the latency between controllers and their connected networking nodes, hence minimising

Table 2.1: Considerations and Metrics of OpenFlow Monitoring Software

Considerations and Metrics			**Tools**			
	Cacti[37]	Cbench[38]	MSActivator[39]	OFPeck[40]	OpenSeer[41]	OpenView[42]
Multi-tenancy	X	X	X	X	X	X
Packet loss rate	X		X	X	X	X
Latency/latency variation	X		X	X	X	X
Outage downtime				X		X
OpenFlow protocol response times		X		X	X	
OpenFlow protocol activation times				X	X	
OpenFlow controller availability metrics						
OpenFlow controller reliability metrics						
OpenFlow controller capacity metrics						
OpenFlow controller accuracy metrics						
OpenFlow controller security metrics						
OpenFlow switch availability metrics						
OpenFlow switch reliability metrics						
OpenFlow switch capacity metrics						
OpenFlow switch accuracy metrics						
OpenFlow switch security metrics						

the time required for a networking node to query a controller, and consequently flow setup times. By optimally placing the controllers it can then be seen to what extent SDN can be used in applications, as optimal placement will provide the best available physical structure for a given network. This information can then be used to determine whether a particular solution is viable or not.

The amount of flows through a controller or switch posed concerns in the early stages of SDN but as hardware developed with SDN in mind this no longer looks to be an issue. Many companies are building 'SDN ready' switches with OpenFlow support. OpenFlow looks to be the standard API for SDN, this is largely due to the support from industry. A more pertinent

question as discussed in [2] is the lack of a standard northbound API and subsequently the Open Daylight project was created.

Reliability and security are big issues for SDN due to the nature of a central control point. Reliability is often addressed through the use of replication and redundancy and this also seems to be the case for SDN. The main concern is how efficiently normal operation can resume and whether or not it can be resumed under the strict deadlines required for the application of the system. Papers have shown proactive approaches offer good performance and as long as the synchronisation is consistent, should offer similar performance to that of existing networks, and is therefore applicable to high-criticality.

The security issues show SDN will give both advantages and disadvantages. The advantages being that anomaly detection will become easier due to the unknown packets querying the controller. However, this also becomes a disadvantage due to the possibility of DoS attacks. Overall the programmability and flexibility of SDN allows for the security to be easily upgraded for specific purposes, whether it be by introducing middle boxes or otherwise. Another concern is insider attacks, however this would be a concern for any system and is not SDN specific.

Scalability for SDN is widely considered with many papers looking at different architectures. The only thing to note on this subject is the fact that a hybrid system could be realised by combining reduced flow to the controller and multiple controllers. The main problem is the synchronisation between the controllers and the question of staleness. This will remain a problem as with any distributed system, but the best method it seems is an ad-hoc approach where updates occur when a network-wide event occurs. This is because it reduces the amount of synchronisation required, and thus traffic, whilst maintaining central control between switches and their local controllers.

As discussed the reliability and delay of the system become more pertinent for high-criticality networks so the controller placement is the place to start. The following section introduces the controller placement and outlines the current literature on the techniques for solving the problem.

2.3 The Controller Placement Problem

The controller placement problem was first formulated in [4], as stated previously the authors look at both where the controller should be placed, and how many controllers are needed to support the network. In order to solve the problem a full search method is used, whereby every possible combination of controller placements is assessed, and the best solution is used. However, the full search method is infeasible for larger networks due to the extensive computation involved. Importantly the work posed as a catalyst for further research in the area of controller placements, and many extensions have been proposed. Much of the work to date can largely be categorised by the method it is solved, the two main categories are that of Linear Programming and approximation heuristics. The nuances of the individual papers are often within the optimisation criteria and aim to find an optimal solution according to a specific networking characteristic. Some of the characteristics include: node-to-controller delay, reliability (also survivability) and energy usage. The most insightful papers ([43], [44],[45] and [46]) compare their own solutions to that of others, and include the results of both such that a comparison can be made. For example, it is useful to see how an optimised solution for reliability compares to one optimised for delay, as the trade-off between different optimisations can be seen. In particular, if a solution for reliability is found, it may be the case that the node-to-controller delay may or may not be at an acceptable level and therefore both aspects need careful consideration. The reviews that follow look at the current literature on controller placement, this then sets up and leads on to the work in Chapter 3.

2.3.1 Integer Linear Programming (ILP) Methods

The following methods use Integer Linear Programming to solve the controller placement problem. This means the problem is formulated as a set of linear constraints with an objective function. All of the papers in this section (Section 2.3.1) use pre-coded Linear Programming solvers, such as CPLEX [47], and the novelty lies in the formulation of the problem. In particular, the

objective function can be altered to change the optimality criteria depending on the required optimisation.

The paper by Muller et al. [19] proposes a new controller placement strategy called Survivor which has the following benefits: reduction of connectivity loss, more realistic controller placement strategy and smarter recovery mechanisms. This is achieved by splitting the problem in to two sub-problems, first of all the placement of controllers is dealt with, and then a list of backup controllers is formulated. To find the placement of controllers an ILP is used, but unlike other controller placement formulations the objective function is to maximise the average connectivity of disjoint paths between the forwarding nodes and the controllers. The constraints in the ILP ensure that the placement, capacity, and connectivity of controllers provides a suitable solution with respect to survivability. A heuristic for the second sub-problem is proposed and two metrics are considered. One of the metrics is the distance between controllers, and a list of backup controllers for each individual controller is formulated based on this. The other metric is the residual capacity, whereby the list of backups is composed by looking at the spare capacity of the controllers. The method is evaluated on three different WAN networks (Internet2 (10 nodes, 15 links), RNP (27 nodes, 33 links) and GEANT (40 nodes, 61 links)) and compared against a Min Cut Centroid method of controller placement. It is shown that the Survivor method performs better with respect to the probability of maintaining connectivity and the probability of becoming overloaded when a failure occurs. One major downfall of the evaluation however is that the controller to node latency is not considered. This is a disappointment as the paper gives a one sided view on how the proposed method improves a certain aspect of controller placement, but does not compare this to the conventional method. This would then provide a welcome indication of the trade-off between survivability and minimum latency.

A simple method for a resilient version of the controller placement problem is investigated in [48]. The problem is formulated as an integer linear program (ILP), but is solved using a commercial program GUROBI. The objective function used considers both the total cost for the deployment of the controllers and the expected routing costs from the controllers to the

switches. The resilience is created within the constraints as each switch must be managed by a controller at a specified resilience level. Each controller also has a capacity, which adds another constraint to prevent the total incurred load from the switches from exceeding the controller's capacity. Further constraints ensure connectivity between the switches and controllers. In the evaluation only 5 networks are used, with the amount of nodes and links ranging from 11 - 42 and 18 – 77 respectively. This is a limited selection of networks and it is mentioned that further work would consider heuristics to support larger networks. The paper also only evaluates the proposed method, so there is no comparison or trade-off analysis performed. This is unfortunate as it would be particularly interesting to compare this method to a delay only one as the resilience is a constraint, and not part of the objective function.

In [49] the controller placement problem is again formulated as an integer linear program, much like many other methods. The key difference is that an extra constraint is added to balance the load on each controller. The constraint also ensures that each controller can not exceed a certain load, and the load is said to be balanced optimally if each controller has an equal share of traffic. Within the new constraint both the load and the delay are considered, and the trade off between the two can be controlled by altering the variables. The ILP is solved with and without the new constraint on 10 different networks of limited size, with the largest having 65 nodes, it is therefore noted that further work would be to use a heuristic. The variable controlling the trade-off is altered and it is shown there is a clear trade-off between load balancing and finding the minimal delay.

Another ILP formulation is given in [50]. The difference in this paper is that three optimization criteria are summed within the objective function. These include the following costs: setup of a controller, linking controllers to switches and linking controllers together. The cost of linking nodes is quantified using a cost parameter along with the distance between the nodes. Similarly a cost is associated for the setting up of each of the controllers, with a binary variable used to indicate whether to use a particular controller or not. The problem is solved using CPLEX, and the results are shown for

networks with up to 200 nodes and 20 controllers. It is mentioned that the optimiser had a 30 hour restriction to solve the problem and was unable to find the optimal solution for four of the 27 networks tested. Theses networks were the larger ones tested and therefore for practical implementations it is stated that heuristics must be used. An observation of the paper is that the number of controllers being placed significantly effected the runtime of the method, this would be another aspect that would benefit from heuristics.

An energy aware formulation is proposed in [43], where the problem is constructed as a Binary Integer Program (BIP). A BIP is simply an ILP with the restriction that the solution space is limited to 0s and 1s. The only real difference to other methods is that each link has an assigned amount of energy that is required for transmission. The objective is then to minimise the amount of energy used when assigning the controller placements. Much like other papers it is noted that a linear program is infeasible for larger networks, so a genetic algorithm is proposed. The energy savings are quantified by the amount of active links that are required in the solution. It is shown that the percentage of links used is reduced when using the BIP and Genetic algorithm, but at the cost of increased delay in the network when compared to a conventional controller placement algorithm. This paper has many strengths in it's evaluation due to the comparisons between other methods, both in terms of the objective and the algorithms used. Importantly the optimality trade-off between an optimal solver, the BIP, and an approximation, the genetic algorithm, is shown, but also this is compared to a random placement and a capacitated controller placement (CCPP). Furthermore, the objective of the CCPP is to minimise the maximum latency between nodes and controllers so a comparison between the energy aware solutions and a conventional delay based method can be seen.

2.3.2 Heuristic Methods

It is mentioned in many of the ILP formulations that heuristics must be used when considering larger networks. This is the case because of the computational complexity of linear programming. On the other hand, heuristics have

the benefit of reduced complexity, but at the expense of optimality. Again, the better papers include a comparison between the proposed method and previous methods. This both highlights and quantifies the advantages and disadvantages of the methods, which allows you to see the trade-off between scalability and optimality. This is an aspect of the work in Chapter 3. Similarly to the ILP formulations, many of the novel aspects between the following papers are the metrics considered.

In [44] the focus is on the reliability of the controller placements rather than latency. The analysis is done by applying a probability of failure to each path and seeing how a controller would perform under this restriction. The reliability metric is defined as the expected percentage of control path loss, where the control path loss is defined as the number of broken control paths due to network failures. The optimisation target is then to minimize the expected percentage of control path loss, rather than the latency. The proposed algorithms are random placement, simulated annealing and a variety of greedy algorithms with different parameters. The performance of the algorithms is analysed through a comparison against an integer linear program (which is assumed to be optimal). The solutions with the reliability metric are compared to the conventional solutions based on latency. It is shown that a solution for an improved average and worst case latency requires an increased expected percentage of control path loss, albeit not a huge increase in average or worst latency. The main downfall of the paper is that the analysis is only performed on a limited number of networks, in this case only seven, with a limited range of the number of nodes (34 – 315) and links (32-1945). Although the NP-hardness of the problem is proven by observing the relationship to the Dominating Set problem, the question of the scalability of the algorithms is not addressed. Also, from experience, it is unlikely the same analysis could be performed for larger networks due to the requirement of an optimal solution being found using an integer linear program, see Chapter 3.

The case for Software defined networking in a mobile setting has some nuances that are addressed in [51]. The main purpose of the paper is to look at how a dynamic allocation policy compares to that of a static one (pre-

determined, fixed placement). In this case the number of controllers can vary in order to satisfy certain QoS constraints. The authors use a particle swarm intelligence optimizitation (PSO) along with a grey wolf optimizer and chaotic mapping. The grey wolf optimiser is so called because it uses a hierarchial method that has tiers depending on how good a solution is. The solutions can then be honed to converge to the best solution (as a wolf pack would do when finding prey). The use of chaotic mapping is a technique to stop convergence to a local optimum (this is similar to choosing a random start point for the algorithm, but is more selective). The emphasis of the paper is on the algorithm which is split in to two functions. One to find the best controller placements and the other to connect the switches to the controllers (as with other ways of finding controller placement). The new algorithm is compared to a simple PSO algorithm and is shown to be better in performance by 1% with 68% less computation. The major downfall of the paper is that only one fixed topology is used for analysis with 10 nodes. This is because the emphasis is on finding a solution when there is a varying arrival rate at the controller.

The authors in [52] propose a k-means based approach for controller placement, and formulate an 'optimised' version. The results show that the optimised version improves performance over a standard version of k-means. The optimised version simply chooses an initial controller, then finds the node furthest away to be used as a controller in the next iteration. This causes the algorithm to become deterministic in the solution, where as for the original k-means++ the centres are chosen according to a probability distribution, which means the solution can change each time. The authors claim this is an improvement since only one instance is needed, where as the k-means requires more instances to find the best solution (100 instances are used by the authors in this particular paper). Using a deterministic algorithm in this way may be unwise as it may not always lead to the best solution. Furthermore, the algorithms are only used on two networks, which shows the results are very limited and perhaps untrustworthy. The paper states that the minimum maximum distance is reduced using this method, but in the algorithm the sum of latencies is used as the metric. This makes the objective unclear and

perhaps the fact that the maximum latency is reduced could be coincidental.

An advantage of using heuristics is the speed at which solutions can be found, this is particularly useful in a mobile scenario. This is the case in [45] where controllers are required to facilitate Device – to – Device (D2D) transfers to alleviate resources and energy consumption in the network. The problem is initially formulated as an ILP. The objective is to find the minimum number of controllers, and their corresponding locations, such that every user is able to connect to at least one controller. One key difference in this paper is that the state space is a 2-dimensional plane, rather than a fixed network. In order to solve the problem a convex hull algorithm is used rather than linear programming. This recursively partitions the state space, and places controllers within the partitions such that each device is connected to a controller. The algorithm is compared to a naïve approach that simply splits the space in to squares and places a controller in the middle of each square, and a random approach where the controllers are placed such that they are within a certain distance of the users. The algorithms are evaluated with different values of the range they can cover, and it is shown the proposed algorithm performs the best, and is less computationally expensive than the random approach. A significant strength of this paper is the use of different methods for comparison as it shows the benefit of using a more intelligent approach. However, a significant weakness is the fact that there is no optimal solution given for comparison, so the optimality gap remains a question for further research.

The work in [46] computes controller placement based on different metrics. Initially a full search method is used, but then heuristics are implemented since finding placements for large networks is infeasible in terms of computation for a full search method. The only algorithm implemented is simulated annealing, hence the main work is the fact that multiple metrics are used, rather than a focus on the algorithms. The metrics consider delay, resilience (in terms of node or controller failures), load balancing (within the control plane), and a mixture of these requirements. The solutions for multiple mixtures of requirements are presented, and a Pareto curve is shown so a user can choose a solution dependent on their own requirements for a controller

placement. This leads to the point that a lot of the work has a focus on the Graphical User Interface (GUI) rather than the actual algorithms, but has the advantage that it is openly available for anyone to use.

The controller placement in a WAN is considered in [9] and uses a spectral clustering approach to sub-divide the network and then place controllers. The algorithm aims to perform cuts on the network according to a min-max objective function, but in reality simply uses a k-means algorithm to perform the partitioning. The sub-clusters generated from the partitioning are then subjected to a full search to find the controller placement, which is optimised according to the average latency. The metric is said to consider propagation delay, load balancing and reliability, but this is a by-product of the partitioning rather than a specific design characteristic. This is because by partitioning the network according to k-means the clusters are more balanced in size. This is opposed to only considering a delay metric over the whole network, which can cause sub-clusters to have a more uneven amount of nodes. The results show that the proposed method performs better when compared to simply considering average latency. However, this is not the case for each individual controller. A better comment would have been to say that the method generates a more consistent metric for each controller. This is because the results show the latency for each controller, and it is clear that the spectral method has less variability between each controller. Where as, the average-latency method has one controller that performs particularly badly in each test, but also some controllers that perform better, which gives a wider spread. This also means that the spectral method performs better in this case *overall*, but maybe not necessarily better in other situations. Finally, the algorithm is supposed to be used for a WAN topology, but the scalability of the algorithm is not considered. In particular, the algorithm is only tested on one topology, with only 36 nodes, which limits the viability of the method.

2.3.3 Summary

The main aspect that is missing from each controller placement paper is that of scalability and a comparison of the different methods with respect to the size of the network. This is the main theme for the research in Chapter 3 where a numerical and theoretical comparison between full search, linear programming, local search and a k-means method is performed in terms of complexity. The accuracy of the methods is also analysed to show the trade-off between the complexity and accuracy of the methods. Chapter 3 explains the methodology of how the controller placement problem is solved, and the modelling used to formulate the problem.

Chapter 3

Controller Placement: Algorithms and Evaluation

3.1 Introduction

The literature review made it apparent that end-to-end delay [1] and reliability are two of the main concerns for high criticality application. While SDN introduces programmability to the system, it does so by including one or more controllers, which introduce additional delay and reliability issues. In terms of delay, the flow setup time is often overlooked in the literature, which focuses on stability over the long term[14, 15]. Although this emphasis may be appropriate for some applications, it is not the case for high criticality applications, which typically require tight delay guarantees.

On the question of system reliability, many publications look only at the reliability of the controller [24, 12, 25], which may not coincide with application-level reliability, especially for high criticality applications. Specifically, it is not just the failure of controllers that can cause delay guarantees to be violated, but also the failure of links, which could increase path lengths in the network. Improving reliability is often a major motivation for adding more controllers in an SDN system, even when their contribution to reducing delay is small [53]. This is an important topic, but not one we study in this

[1] including both node-to-controller and node-to-node delay

thesis.

In Section 3.2, we recapitulate a precise formulation of the controller placement problem from [4], and show that one version of it is equivalent to the k-medians problem, which we also state. We then introduce approximation algorithms for the k-median problem and the topologies we evaluate relating to SDNs. Finally, we present our results for the approximation algorithms we define in 3.4 in terms of accuracy (optimality) and complexity (time to find a solution).

3.2 The Controller Placement Problem

Given a network topology, the controller placement problem is formulated informally [4] as:

1. How many controllers are needed?

2. Where in the topology should they go?

The question of how many controllers are needed depends on the delay bounds that need to be met; the more controllers there are, the more stringent the delay bounds that can be met. However, too many controllers incur additional cost, and they could even potentially slow down the system due to the controllers requiring synchronisation, an issue looked at in [25].

The question of where in the topology the controllers should go depends on the objective. Two natural objectives are considered in [4]: minimising the average delay, and minimising the maximum delay. In general, these can lead to different optimal placements, as depicted in Figure 3.1.

In order to address the above questions fully, we need a precise and formal mathematical statement of the problems, which we now provide. Let $G = (V, E, \boldsymbol{w})$ be an arbitrary directed and edge-weighted network, with vertex set V, edge set E, and non-negative edge weights $w_e, e \in E$. The weights represent the latency or propagation delay across the corresponding edges, which might consist of both transmission and queueing delays. We will address the topic of queueing delays further in Chapters 5 and 6. Now, for

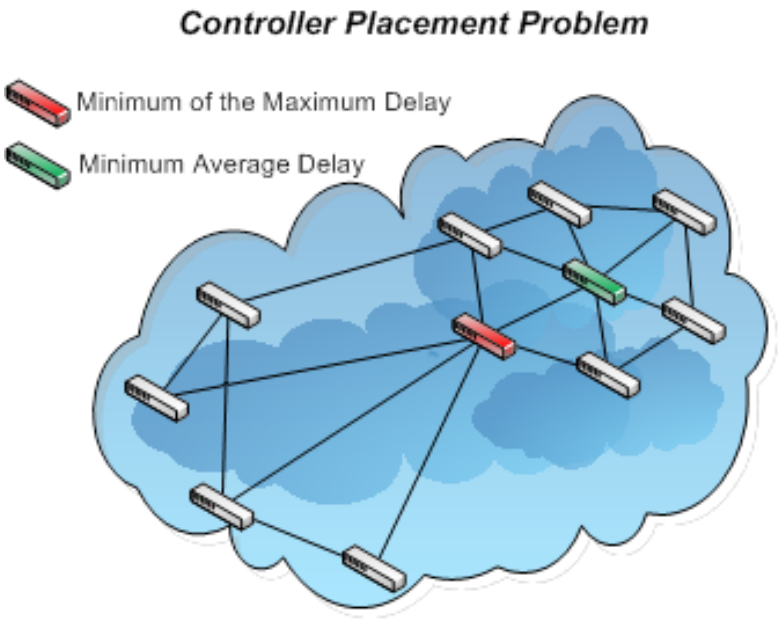


Figure 3.1: The Controller Placement Problem example solution for the minimum average delay and minimum of the maximum delay

$u, v \in V$, we denote by $d(u, v)$ the distance between u and v, defined as the weight of a minimum-weight path from u to v. If the graph G is undirected and the edge weights are symmetric, i.e., $w_{(u,v)} = w_{(v,u)}$ for all $u, v \in V$, then the distances will also be symmetric, i.e., $d(u, v) = d(v, u)$. We now define two objective functions for the controller placement problem, following [4]: for a subset $S \subset V$ of nodes at which controllers are placed, the resulting average and maximum latencies are given by

$$L_{ave}(S) = \frac{1}{|V|} \sum_{v \in V} \min_{u \in S} d(v, u), \quad L_{max}(S) = \max_{v \in V} \min_{u \in S} d(v, u). \tag{3.1}$$

The controller placement problem is then the following optimisation problem:

$$\min_{S \subseteq V} L(S) \text{ subject to } |S| \leq k, \tag{3.2}$$

where $L(S)$ could be either $L_{ave}(S)$ or $L_{max}(S)$, and k is the number of controllers available. It is clear that the inequality constraint can be replaced

by an equality constraint without loss of generality, as placing additional controllers can never increase either of the objective functions.

An alternative formulation of the controller placement problem is to specify a bound on the objective function of interest (average or maximum) delay, and ask for the minimum number k of controllers required to meet this constraint, as well as a placement that does so. This variant of the problem can be solved by repeatedly solving the controller placement problem for different k. A bisection search approach to solving for k implies that we need solve only about $\log_2 |V|$ iterations of the controller placement problem for fixed k. Thus, the computational complexities of the two versions of the problem are not too dissimilar.

One natural generalisation of the optimisation problem in (3.2) is to consider a budget constraint, with different costs for placing a controller at different nodes; in other words, the constraint in (3.2) becomes $\sum_{v \in S} c_v \leq C$, where c_v is the cost of placing a controller at v, and C is the total budget. This is known as the facility location problem. Another generalisation would be to consider weighted sums in defining L_{ave}, i.e., to set

$$L_{ave}(S) = \frac{1}{|V|} \sum_{v \in V} \min_{u \in S} w_v d(v, u),$$

where w_v is a weight representing the amount of traffic flowing through node v. We don't consider these variants further, beyond mentioning that all the algorithms we study can be easily adapted to them.

We now establish the equivalence of the controller placement problem with the average latency criterion to the k-medians problem in the special case that the distances $d(u, v)$ are symmetric, i.e., $d(u, v) = d(v, u)$ for all $u, v \in V$. Assuming this is not too restrictive in our setting, as we are typically interested in round-trip delays $d(u, v) + d(v, u)$, the time for a node to query the controller about a packet or flow, and get a response specifying rules for handling it. Round-trip delays are symmetric by definition.

The version of the k-medians problem most relevant to us is as follows. We are given a finite set V and a distance function $\rho : V \times V \to \mathbb{R}_+$ which satisfies the properties of a metric, namely:

1. $\rho(x, y) = \rho(y, x)$ for all $x, y \in V$.

2. $\rho(x, y) \geq 0$ for all $x, y \in V$, with equality if and only if $x = y$.

3. For all $x, y, z \in V$, $\rho(x, z) \leq \rho(x, y) + \rho(y, z)$.

Then, the k-medians problem is the following:

$$\min_{S \subset V} \sum_{v \in V} \min_{u \in S} \rho(u, v) \text{ subject to } |S| = k. \tag{3.3}$$

Comparing (3.3) with (3.2), where $L(S)$ is specified in (3.1), we see that the problem of minimising average latency is equivalent to the k-medians problem, with $\rho(u, v) = d(u, v)/|V|$.

3.3 Approximation Algorithms for K-Medians

In the last section, we showed that the controller placement problem with the average latency criterion reduces to the k-median problem. Unfortunately, the k-median problem is known to be NP-Hard [54]. This leads us to consider approximation algorithms for the k-median problem, which have been the subject of a considerable amount of research. Below, we summarise a selection of the approximation algorithms that have been proposed for this problem, and their approximation guarantees. The problems will be parametrised by two variables, the number of vertices in the network, denoted by n, and the number of medians required, denoted by k. In practical SDN applications, n could be of the order of thousands to millions, while k, the number of controllers, could be anywhere between one and a few tens, or even a few hundreds. Thus, it can be helpful to consider asymptotic measures of complexity and performance of the algorithms, as both n and k tend to infinity.

We reproduce a survey of results from [54], where methods to find both the optimal solution on specific topologies and numerous approximations are described. The k-median problem can be solved exactly on paths and trees, in polynomial time: see (Table 3.1).

Table 3.1: Exact algorithms on specific topologies.

Author	**Complexity**	**Notes**
Tamir[55]	$O(n^2k)$	Solves on a tree from the leaves towards the root.
Tamir[55]	O(kn)	Input graph is a path.

One approach to solving the k-medians problem on general graphs is then to embed the graph into a tree, and solve the resulting problem on the tree using the methods of Tamir from (Table 3.1). However, such an embedding introduces some distortion; the distances between nodes on the graph is not exactly preserved in the tree embedding, but only up to some multiplicative factor. Thus, the solution of the problem on the tree only provides an approximation to that on the original graph. A number of such algorithms, along with their approximation guarantees (the ratio of the cost obtained by the algorithm to the true minimum cost) are shown in (Table 3.2).

Table 3.2: Algorithms exploiting tree structures.

Author	**Approximation Guarantee)**	**Notes**
Bartal[56]	$O(\log^2 n)$	Network is formed in to tree structures then solved optimally.
Bartal[57]	O(log(n)log(log(n)))	Improved version using partitionable metrics.
Fakcharoenphol, Rao, Talwar[58]	O(log(n))	Decompose the graph by growing the clusters diameters then forming trees.
Charikar et al.[59]	O(log(n)log(log(n)))	Find the trees using a relaxed linear programming approach.

It can be seen from this table that the approximation ratios are rather large, and grow with n. This raises the question of whether there are efficient algorithms that can achieve constant approximation ratios, or even ratios

arbitrary close to 1. Algorithms of this form are presented in (Tables 3.3 and 3.4). In Table 3.4 ϵ is an arbitrary parameter for specifying precision, and provides a trade off between complexity and the approximation guarantee.

Table 3.3: Constant factor approximations.

Author	**Approximation Guarantee**	**Notes**
Charikar et al[60]	$6\frac{2}{3}$	Uses linear programming to form trees, then is solved optimally on the trees (forest) formed.
Jain and Vazirani[61]	6	Uses a primal-dual schema which makes the algorithm more efficient as linear programming is no longer used.
Charikar and Guha[62]	4	Uses Lagrangian relaxation (moves a constraint to the objective function) over two problems and then combines them.
Arya et al [63]	3 + 2/p	Local search is used, with p swaps.

Table 3.4: Arbitrary precision algorithms

Author	**Complexity**	**Approximation Guarantee**	**Notes**
Arora et al[64]	$n^{O(1+\frac{1}{\epsilon})}$	$1+\epsilon$	Divides the network in to segments. ϵ is an arbitrary parameter for specifying precision
Arora et al[64]	$n(O(log(n/\epsilon))^{2k}$	$1+\epsilon$	Approximates the median locations but only for small k.

Finally, we look at algorithms that also relax the constraint on the number of medians. Such a relaxation reduces the complexity of the algorithms. An algorithm is described as being α-OPT with β-k-medians if it uses no more than β-k-medians and if its cost is no more than α times the optimal cost. A selection of such algorithms is presented in Table 3.5, where ϵ has the same interpretation as the previous table.

A number of different extensions of the k-median problem have also been

Table 3.5: βk Median approximations.

Author	**Complexity**	**Optimality (αOPT)**	**Medians (βk)**	**Notes**
Lin and Vitter[65] [66]	Polynomial	$1+\epsilon$	$\beta = (1+1/\epsilon)(\ln(n)+1)$	Linear programming with relaxations then rounding.
Korupolu et al[67]	Polynomial	$1+\epsilon$	$\beta = 10 + 17/\epsilon$	Uses local search and permutes until a solution is settled.
Korupolu et al[67]	Polynomial	$O(\epsilon^3)$	$\beta = (5 + 1/\epsilon)$	Uses local search and permutes until a solution is settled.
Indyk[68]	$O(n^2)$	$(1+\gamma)3(2+\epsilon)$	$\beta = 2$	γ is a confidence parameter , the algorithm uses a sample of points in the network, runs on the sample then takes the points furthest away from the medians and runs again.

studied in the literature. The k-median with diameter, the capacitated or robust k-median problem and the facility location problem are all of particular relevance to the SDN setting. The k-median problem with a diameter imposes a minimum distance between locations of medians. The capacitated k-median imposes a limit on the maximum number of nodes in each cluster (closest to each median). Identifying medians with controller locations, both have the potential to improve reliability by reducing the number of nodes effected by the failure of a given link or a given controller. In the facility location problem, there is a cost for placing a controller at each node and a budget, but no explicit bound on the number of controllers. This could be realistic in the SDN setting, where the system operator only has access to some node locations for placing controllers, and where these locations may greatly differ in cost for legal or logistical reasons.

3.4 Empricial Evaluation: Algorithms and Topologies

In this section, we present three different algorithms for the controller placement problem, and evaluate their performance and complexity on a large number of different topologies. The first algorithm we study is a linear programming (LP) relaxation of the k-medians problem, which is itself an integer linear programming (ILP) problem; this algorithm was proposed in [65]. The second is a local search heuristic, which is a novel contribution of this thesis. The last is an algorithm for the k-means rather than the k-medians problem, known as k-means++, which was proposed in [69]. Both the local search of k-medians, and the k-means algorithm, have not previously been applied to controller placement in SDN, to the best of our knowledge. Our final contribution in this chapter is an empirical evaluation of these three algorithms on a wide range of Internet topologies, including two of particular relevance to SDN implementations.

As in the last section, we will use n to denote the number of nodes in the network and k to denote the number of controllers. Throughout this section, we assume that we are given an undirected graph $G = (V, E)$, together with a symmetric matrix of shortest distances between all pairs of nodes in G. If only edge lengths are given, then shortest distances can be calculated easily using any of a number of well-known algorithms. As these distances will be used repeatedly by any of our algorithms, it would make sense to calculate and store them, and we will do so as part of a pre-processing stage. For an arbitrary graph on n nodes, the complexity of this pre-processing step is $O(n^3)$ using standard algorithms such as the Floyd-Warshall algorithm; it could be substantially less for sparse graphs.

We now present the algorithms that we will evaluate. The topologies used for evaluation are described in the final subsection of this section.

3.4.1 LP relaxation of k-medians

In Section 3.2, we reduced the controller placement problem to a k-median problem, stated formally in equation (3.3). The latter is, in fact, an instance of an integer linear programming (ILP) problem, which we now state:

$$\begin{aligned} \text{Minimise} \quad & \sum_{u \in V} \sum_{v \in V} \rho_{uv} x_{uv} & \\ \text{subject to} \quad & \sum_{v \in V} x_{uv} = 1 & \forall\, u \in V \\ & \sum_{v \in V} y_v \leq k & \\ & x_{uv} \leq y_v & u, v \in V \\ & x_{uv}, y_v \in \{0, 1\} & u, v \in V \end{aligned} \tag{3.4}$$

The variables x_{uv} and y_v introduced in the optimisation problem have the following interpretation: y_v is the indicator that a controller is placed at node v, or equivalently that node v is chosen as one of the medians, while x_{uv} indicates that node u belongs to the cluster around the median v, or equivalently is served by the controller at v. If we ignore the integrality constraints, then we allow fractional assignment of nodes to controllers, but constrain these fractions to add up to 1. We also allow fractional placements of controllers, but require that these fractions add up to no more than k controllers. Thus, relaxing the integrality constraints to non-negativity constraints, we obtain the linear programming (LP) problem:

$$\begin{aligned} \text{Minimise} \quad & \sum_{u \in V} \sum_{v \in V} \rho_{uv} x_{uv} & \\ \text{subject to} \quad & \sum_{v \in V} x_{uv} = 1 & \forall\, u \in V \\ & \sum_{v \in V} y_v \leq k & \\ & x_{uv} \leq y_v & u, v \in V \\ & x_{uv}, y_v \geq 0 & u, v \in V \end{aligned} \tag{3.5}$$

An existing method is to solve this LP and then randomly round the solution to find an integer solution. The method is summarised in algorithm 1.

Algorithm 1 Linear Program

Inputs Network: $G = (V, E)$, distance matrix $\rho(u, v)_{u,v \in V}$, number of controllers: k , number of iterations: R.
Solve the linear program in eq. 3.5 using an LP solver (e.g. CPLEX or built-in MATLAB function)
for $i = 1 : R$ **do**
  Randomly choose S with $|S| = k$ from the non-zero elements of y
  for $v \in V$ **do**
    Identify the closest node in S, breaking ties arbitrarily: $\phi_S(v) \in argmin\{\rho(u, v), u \in S\}$
  end for
  Calculate the total cost $C_{total} = \sum_{v \in V} \rho(v, \phi_S(v))$
  if First iteration **then**
    $C^*_{total} = C_{total}$
    $S^* = S$
  else
    if $C_{total} < C^*_{total}$ **then**
      $C^*_{total} = C_{total}$
      $S^* = S$
    end if
  end if
end for
Outputs
C^*_{total} : the minimum sum latency
S^* : an optimal controller placement

3.4.2 Local Search

The first novel algorithm we present for solving the controller placement problem uses a local search method. In the local search method initial clusters are formed randomly then nodes are swapped between clusters in each iteration. The algorithm is called a local search as improvements between iterations are achieved by swapping nodes that are found close to the current cluster of interest i.e. the node that has been *searched* for and is being swapped is *local* to the current cluster of interest. If the total cost is decreased by a

swap, then the configuration is updated, otherwise the previous configuration is used and another swap is performed. This approach is repeated for the number of iterations specified by the user as shown in the pseudo-code in algorithm 2.

The Local Search Algorithm provides a fast method for finding a solution, which is a significant advantage if a controller placement is required in a dynamic setting. The speed at which the method can compute a solution is also an advantage as the algorithm can be initiated multiple times, and the best cost is taken from a number of instances. This is an advantage as the possibility of the final solution converging to a local optimum is reduced.

3.4.3 K-means++ Adaptation

The k-means algorithm is arguably the most well known clustering algorithm to date, yet to the best of our knowledge we are the first to propose using it as an approximation for solving the controller placement problem. We now give a formal mathematical formulation of the k-means problem. Let $G = (V, E)$ be a graph and let $\boldsymbol{\rho} = \rho(u, v)_{u,v \in V}$ be a symmetric matrix of pairwise distances between vertices, satisfying the properties of a metric. Then the k-means problem is as follows:

$$\min_{S \subset V} \sum_{v \in V} \min_{u \in S} \rho^2(u, v) \text{ subject to } |S| = k. \tag{3.6}$$

A commonly used approach to tackling this problem is to start with an arbitrary initial clustering of the nodes into k clusters, and alternate between two steps. In one step, given a clustering, one finds a node in each cluster that minimises the sum of squared distances from other nodes (i.e., solve a 1-means problem for each cluster). In the other step, given a collection of k centres, one computes a new clustering by associating each node with its closest centre. Both steps reduce the sum of squared distances and so, by monotonicity, the algorithm converges. However, it may converge to a local rather than a global optimum.

We follow the k-means++ algorithm described in [69] to choose the initial

Algorithm 2 Local Search

Require: Network $G = (V, E)$, number of controllers k , number of swaps: R

Randomly Form Initial clusters of equal size, $M = \{M_1, \cdots, M_k\}$

Initialise the set of controller locations $S = \emptyset$

Start sub-function

for $j = 1 : k$ **do**

Solve the controller placement problem with $k = 1$ exactly for each cluster: $v_j = argmin_{v \in M_j} \sum_{u \in M_j} \rho(u, v)$

Save the controller location v_j in S such that $S = S \cup v_j$

Save the placement cost $C_j = \sum_{u \in M_j} \rho(u, v_j)$

end for

Compute the total cost: $C = \sum_{j=1,\cdots,k} C_j$

Save the controller locations as $S = \{v_1, \ldots, v_k\}$

End sub-function

for $i = 1 : R$ **do**

Choose a random cluster from $M_1, \cdots, M_k$: denote it M_{random}

Find the nearest node $\notin M_{random}$ from the remaining clusters $(M \setminus M_{random})$

Label the nearest node m , and its corresponding cluster $M_{nearest}$

Redefine M_{random} as $M_{random} \cup m$

Redefine $M_{nearest}$ as $M_{nearest} \setminus m$

Let M' be the set of new clusters with m interchanged

Compute the new total cost and controller locations using the **sub-function**, call them C' and $S' = \{v'_j, \cdots, v'_k\}$

if $C' < C$ **then**

Keep the new clusters: $M = M'$

Set $C = C'$

Update the controller locations: $S = S'$

end if

end for

return Clusters: M , Controller locations: S, Total cost: C

clustering. It works as follows:

1. Choose one centre u uniformly at random from the vertex set, V. Set $S = \{u\}$.

2. For each vertex $v \in V \setminus S$, compute $D(v, S)$, the distance between v and the nearest centre that has already been chosen: $D(v, S) = \min_{u \in S} \rho(u, v)$.

3. Choose one vertex $v \in V \setminus S$ at random as a new centre, using a weighted probability distribution where v is chosen with probability proportional to $D(v, S)^2$.

4. Repeat steps 2 and 3 until k centres have been chosen, i.e., until $|S| = k$.

After the initial centres have been chosen, we alternate the following steps as explained above:

1. **Assignment step:** Partition the vertex set V into clusters $M_1, M_2, \ldots, M_k$ by assigning nodes to the closest centre. In other words, given centres $u_1, u_2, \ldots, u_k$, assign vertex v to M_i if $d(u_i, v) < d(u_j, v)$ for all $j \neq i$. Break ties arbitrarily. Then, $M_i \cap M_j = \emptyset$ for all $j \neq i$, and $\cup_{i=1}^{k} M_i = V$.

2. **Update Step:** Calculate the centres of the new clusters:

$$u_i = argmin_{v \in M_i} \sum_{w \in M_i} \rho^2(v, w) \tag{3.7}$$

The algorithm ends when the assignments no longer change. In practice, one might need to set a bound on the number of iterations as convergence could be very slow.

As we are concerned with the k-median problem, we find clusters according to the k-means criterion for a certain number of iterations, then terminate by finding the median of each of the final clusters. The median of each cluster is found by solving the 1-median problem exactly on each cluster, which is not computationally expensive. Our method can be formulated as in algorithm 3.

Algorithm 3 K-means adaptation

Require: Network: $G = (V, E)$, number of controllers: k , number of initialisations: R

Initialisation step: Form clusters by using the k-means and k-means++ algorithms

Choose one centre u_1 uniformly at random from the vertex set, V. Set $S = \{u_1\}$.

for $i = 2 : k$ **do**

For each vertex $v \in V \setminus S$, compute $D(v, S)$, the distance between v and the nearest centre that has already been chosen: $D(v, S) = \min_{u \in S} \rho(u, v)$.

Choose one vertex $u_i \in V \setminus S$ at random as a new centre, where v is chosen with probability proportional to $D(v, S)^2$.

Update S with the new centre such that $S = S \cup u_i$

end for

Initialise the while loop $S^* = \emptyset$, $M_i^* = \emptyset$ for $i = 1, \cdots, k$

while $S \neq S^*$ **AND** $M_i \neq M_i^*$ for $i = 1, \cdots, k$ **do**

Assignment step: Partition the vertex set V into clusters $M_1, M_2, \ldots, M_k$ by assigning nodes to their closest centre (the centres are contained in the set $S = \{u_1, \cdots, u_k\}$).

for Each $v \in V \setminus S$ **do**

Assign vertex v to M_i if $d(u_i, v) < d(u_j, v)$ for all $j \neq i$. Break ties arbitrarily.

end for

Update Step: Calculate the centres of the new clusters $M_1, \cdots, M_k$:

for $i = 1 : k$ **do**

$u_i = argmin_{v \in M_i} \sum_{w \in M_i} \rho^2(v, w)$

end for

Save the current configuration: Let $S^* = S$, $M_i^* = M_i$ for $i = 1, \cdots, k$

end while

Reinitialise $S = \emptyset$ as we require the optimal k-medians for the clusters M_i for $i = 1, \cdots, k$

for $j = 1 : k$ **do**

Solve the controller placement problem with $k = 1$ exactly for each cluster: $v_j = argmin_{v \in M_j} \sum_{u \in M_j} \rho(u, v)$

Save the controller location v_j in S such that $S = S \cup v_j$

Save the placement cost $C_j = \sum_{u \in M_j} \rho(u, v_j)$

end for

Compute the total cost: $C = \sum_{j=1,\cdots,k} C_j$

Save the controller locations as $S = \{v_1, \ldots, v_k\}$

return Clusters: $M_1, \cdots, M_k$, Controller locations: S , Total metric: C

3.4.4 Complexity of the algorithms

We briefly comment on the complexity of the algorithms described above. The theoretical complexity estimate of each algorithm is then complemented with numerical timings.

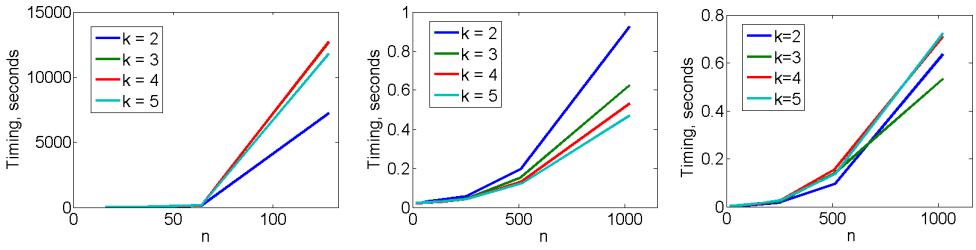
Distance Computation The pre-processing step common to all our algorithms requires the computation of pairwise distances between nodes given the edge lengths. This requires solving the all-pairs shortest path problem. Standard algorithms for this problem, such as the Floyd-Warshall algorithm and Dijkstra's algorithm have a complexity of $O(n^3)$ on arbitrary graphs. The algorithms can be significantly quicker on sparse graphs.

Linear Programming The complexity of linear programming is dominated by the interior point algorithm described in [70]. This method has complexity $O(n^4/ln(n))$ [71]. Common implementations of LP generally use the simplex method, whose worst-case complexity is not polynomial in n, but which works well in practice.

Local Search Given an initial choice of cluster centres, it takes nk steps to identify the closest centre for each of n nodes, incurring a cost of nk for forming the initial clusters. Subsequently, each swap involves two clusters, whose centres need to be updated. Once the centres are updated, it needs to be checked for every node whether it needs to be reassigned to a different cluster, but the only possible reassignments are to the two clusters affected, not all k of them. Thus, a swap only requires $O(n)$ time, and not $O(nk)$. As the local search method has a user defined amount of repetitions, R, the complexity becomes $O(n(k+R))$.

K-means++ The initialisation phase of k-means++ requires $O(n)$ computations to pick each of k centres, which adds up to a total of $O(nk)$. The iteration phase has been shown to be bounded in complexity by $2^{\Omega\sqrt{n}}$, which is the dominant term [72]. This is a worst case theoretical bound and the practical results show much better performance.

Numerical timings For each of the methods described, the numerical timings are calculated by implementing the algorithms in MATLAB version 2012a. The system used an intel i7 3GHz processor with 8GB RAM running on a 64-bit version of windows 7. The initial network used to test the numerical timings is made up of a single 16 node network with unit length links. The 16 node network is then duplicated to create a network of size 32, 64, 128, 256, 512 and 1024. The algorithms were all run 100 times on the different sized networks for $k = 2, 3, 4, 5$ and the average time over the 100 runs is presented in Figure 3.2. The size of the networks tested for the Linear Program was limited to 128 nodes due to timing constraints, but the timings for the local search and k-means++ were calculated on all of the different sized networks. In Figure 3.2 200 swaps were performed for the local search algorithm and the k-means++ method used 1 initialisation.



(a) Linear Program (b) Local Search (c) K-means++ Adaptation

Figure 3.2: Numerical Timings to find controller placements for various values of k

3.4.5 Topologies for Evaluation

The topological challenges of software defined networks make the controller placement problem unique. In traditional networks for example, a control element is co-located within each networking device, meaning the controller placement problem does not exist. Furthermore, although the controller placement problem can be widely used in different networking scenarios such as transportation, the focus of a solution is different. For example, in a transportation network the geographical scale may be large (national, international, etc.), but the number of nodes/locations may be limited (e.g. due

to property costs). In this scenario the solution can also be computed over a longer time period due to external factors such as planning permission, and an optimal solution would likely be required to minimise the financial cost to the organisation.

What makes the controller placement problem specific in SDNs is the large scale, flexibility (in terms of where controllers can be placed), and requirement for fast computation. In this case we may not be able to get an optimal solution due to complexity limitations, and therefore approximation algorithms must be used, this is the aim of the approximation algorithms proposed in this thesis. To highlight these requirements the first network we evaluate our algorithms on is a large rail network. This was chosen as it is directly applicable to the problem of using SDNs in high criticality infrastructure.

A topology frequently used for SDN research is the internet2 network [9, 4, 73]. The internet2 network is relevant to SDNs as it has a layered, hierarchical, structure. Further examples of networks can be found on the internet topology zoo [7], and we show a worked example solution from the Intellifibre network. We chose the Intellifiber network as it is owned by a company that promotes the use of SD-WAN (Software Defined Wide Area Networks) as a networking solution. In particular, this network will show the application of our approximation algorithms at a larger scale than the internet2 topology. Example solutions to the internet2 and Intellifibre networks are presented in Section 3.5.3. This highlights the relevance and applicability of our algorithms to SDN, particularly at a larger scale, which is not possible using conventional methods such as a full search approach.

3.5 Empirical evaluation: Results

We evaluated each of the three algorithms described earlier in this chapter on a number of different topologies. We looked at the trade-off between their speed and performance, i.e., how quickly they produced a feasible solution, and its total cost. The cost function we looked at was the sum of the latencies, which is equivalent to the average latency (up to a scaling factor).

For each of the methods described, the solutions are calculated by implementing the algorithms in MATLAB version 2012a. The system used an intel i7 3GHz processor with 8GB RAM running on a 64-bit version of windows 7. We calculated the total cost of the best solution found by each of the algorithms. We took the LP-based algorithm as our baseline, and compared the total cost achieved by the other two algorithms to that achieved by the LP algorithm.

3.5.1 Results for the Railway Network

The first test network we looked at was a large rail network. This was chosen for the applicability of SDNs in high criticality infrastructure, see Section 1.2 for an explanation of high criticality. The network is formed of 124 stations (nodes), which we take as the vertices in our network and assume at most 1 controller would be required in each location. A feature of this network is that each vertex only has 1, 2 or 3 edges linking to other vertices. This means the network is sparse in nature. In each test the algorithms were run 100 times in order to compute the statistics in Figure 3.3 and Table 3.6.

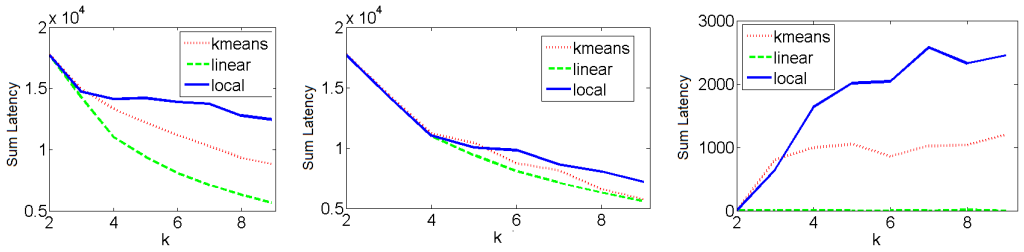
A number of results were found, first of all it is apparent that the linear programming method was superior to the other methods in terms of accuracy. Figure 3.3 shows that the linear programming method gives the smallest total cost. Furthermore, the metric is consistent over each initialisation, shown by the small standard deviation. This is believed to be because the non-integer linear program provides a good initial solution, so the rounding to an integer solution does not effect the cost in each instance.

Second, it can be seen that the number of repetitions improves the local search method significantly, up to a point where it is competitive with the linear program. At 10 and 100 repetitions the algorithm performs particularly badly, but when increasing the number of repetitions to 200 a significant improvement in performance can be seen. Increasing the repetitions further to 500 and then 1000, the minimum cost is very close to that obtained by linear programming. This is an interesting point as the complexity of the local search method is significantly less than that of linear programming, and

so it can be used on much larger scale problems.

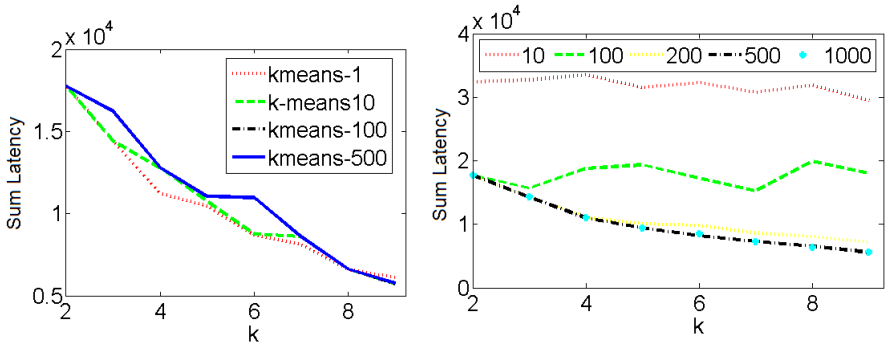
Although the cost improves with the number of repetitions for the local search method, this is not the case for the k-means++ method. When looking at Figure 3.4 it can be seen that having a larger number of initialisations does not improve the minimum cost. This could be caused by the method finding a local minimum from the initialisation each time or due to the rounding involved to find the median. Furthermore, the standard deviation of the method is smaller than that of the local search method, indicating less variation, which in this case is not always a good thing as the variation can lead to an improved solution.

In order to show the differences in the methods table 3.6 shows the raw statistics of 100 instances of each method. The local search method was run with 1000 repetitions and the k-means with 1 initialisation. In particular, the interquartile range (IQR) shows significant differences between the methods. While the IQR for linear programming has a value of 0 for all the values of k, the local search and k-means methods have a large IQR for values of k = 4 and above. The variance in the results for the local search method is believed to be because of the random initial clusters, this also partially determines how good the solution becomes after the swaps have occurred. The variance is slightly reduced in the case of the k-means due to the probability weighting applied to the initial clusters.



(a) Means (b) Minimums (c) Standard Deviations

Figure 3.3: Performance comparison between the K-means++ method with 1 initialisation, Local Search with 200 Repetitions & the Linear Program



(a) K-means++ (b) Local Search

Figure 3.4: Performance comparison between the K-means++ method with an increasing number of initialisations and the Local Search method with an increasing number of swaps

Table 3.6: Raw data

Linear Programming	k = 2	k = 4	k = 6	k = 8
Mean	17693.67	11004.04	8099.15	6311.207
Standard Deviation	0	0	0	0
Min	17693.67	11004.04	8099.15	6311.207
Max	17693.67	11004.04	8099.15	6311.207
Interquartile Range	0	0	0	0
Local Search	k = 2	k = 4	k = 6	k = 8
Mean	17693.67	12465.46	9245.76	7227.499
Standard Deviation	0	1046.997	880.6258	556.6958
Min	17693.67	11004.04	8475.173	6321.83
Max	17693.67	13458.47	12195.89	8647.53
Interquartile Range	0	2188.92	558.4767	994.2567
K-means++	k = 2	k = 4	k = 6	k = 8
Mean	17750.43	13335.10	11199.17	9339.12
Standard Deviation	0	993.05	853.52	1034.32
Min	17750.43	11216.09	8743.77	6612.96
Max	17750.43	14926.76	13460.34	11527.81
Interquartile Range	0	1167.42	657.1	1716.47

3.5.2 Results for Internet Zoo Topologies

In order to test the accuracy of the algorithms further a sample of topologies from the Internet topology zoo were used [7]. Although these networks are not all SDN specific they are closely related to SDN due to the potential application of SDN to internet topologies. The sample taken consisted of 127 different topologies, with sizes varying from 13 to 113 nodes. In order to show the performance of both the k-means++ and local search on a single graph the CDF is presented for the minimums of both algorithms in Figure 3.5 and a scatter plot in Figure 3.6, with a normalised metric with respect to the linear program. The normalised metric for the k-means++ algorithm is calculated by taking the metric for the k-means++ on a particular network and subtracting the metric from the Linear Program for the same network, and then dividing by the Linear Program metric. The normalised metric is also calculated for the local search algorithm in the same way. In this way, by assuming the linear program metric is closer to optimal i.e. a smaller value the normalised metric will always lie in the range $[0, 1]$. Furthermore, a normalised metric close to 0 implies the metrics are similar, and a normalised metric of 1 suggests a large disparity. The mathematical formulation for the normalised metric , C^* is:

$$C^* = \frac{C_{comp} - C_{LP}}{C_{LP}} \tag{3.8}$$

where

C_{LP} is the total metric for the Linear Program

and

C_{comp} is the total metric for the comparator (k-means or local search)

In our tests the local search method used 200 swaps and the k-means used 1 initialisation to maintain a similar timing constraint, the value of k = 4 was used to give significant variation between methods. The main result from Figure 3.5 shows that the local search method generates a metric the same as the linear program for 70% of networks, where as the k-means++

algorithm only remains the same for 25%, showing the local search method outperforms the k-means++ in this scenario. A similar trend can be seen in the scatter plot (Figure 3.6), where each index on the horizontal axes represents an individual network.

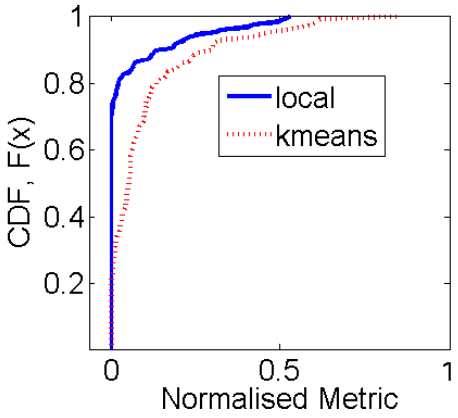


Figure 3.5: CDF over the Internet topology zoo of the normalised costs using the k-means++ and local search methods, k = 4

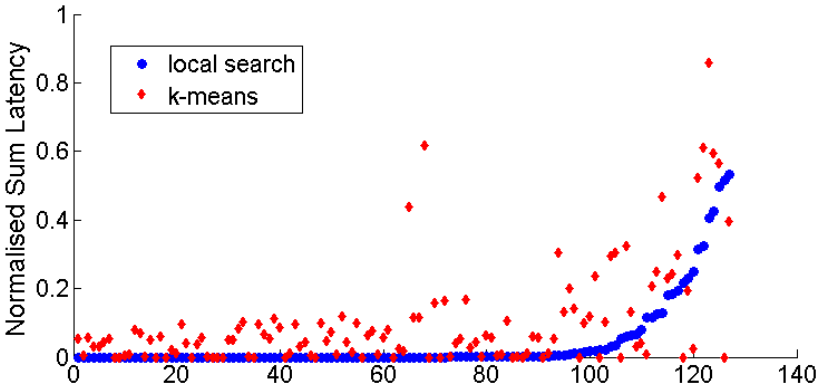


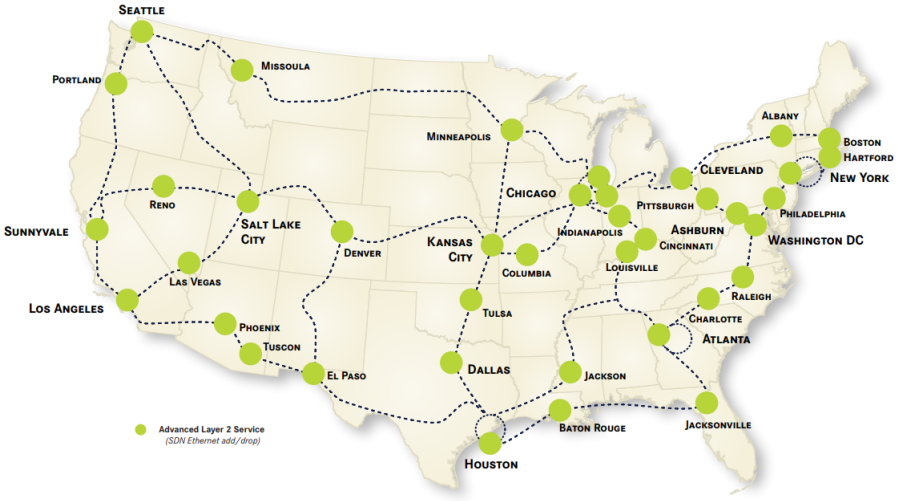
Figure 3.6: Scatter plot over the Internet topology zoo of the normalised costs using the k-means++ and local search methods, k = 4. Each index on the horizontal axis corresponds to a single network, the plot is ordered with respect to the local search normalised metric.

3.5.3 SDN-specific Topologies

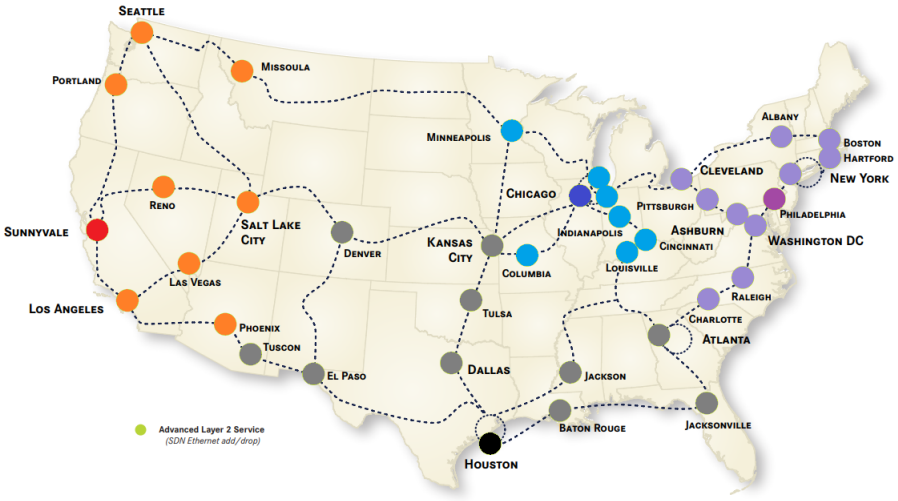
A topology frequently used for SDN research is the internet2 network [9, 4, 73]. The internet2 network is relevant to SDNs as by design it has a layered, hierarchical, structure. An example solution to the controller placement for this topology is depicted in figure 3.7. The solution shows that each controller is based near to the centre of it's surrounding connected nodes, and that each controller has a similar number of nodes connected to it. This is an expected result, as both of these characteristics will help to minimise the overall metric that was defined in equation 3.3.

The algorithms presented in this chapter have been tested for their accuracy on 127 networks from the internet topology zoo [7] in section 3.5.2. Here we present a specific example in figure 3.8, which is taken from the Intellifiber network. The Intellifiber network is owned by Windstream Communications, which promotes the use of SD-WAN (Software Defined Wide Area Networks) as a networking solution. In particular, this network shows the application of our algorithms at a larger scale than the internet2 topology (in terms of the number of nodes: internet2 has 39 nodes with 102 edges, intellifiber has 73 nodes with 190 edges). The use of a larger network shows the ability of the k-means and local search algorithms to find a solution to the controller placement problem in a much shorter time frame than the linear programming method. Furthermore, when the number of nodes is increased further it becomes infeasible to calculate a solution using the linear programming method due to the complexity of the algorithm.

To quantify how well the local search and k-means algorithms perform in terms of optimality on these networks the minimum metrics are presented and compared to the linear programming method (the assumed optimal method) in tables 3.7, 3.9, 3.8 and 3.10. Different instances of the local search and k-means were used to generate solutions to show a comparison between using a different number of swaps or initialisations. The local search 100, 500 and 1000 methods simply use a different number of total swaps to generate a solution. The local search repeat method used 100 swaps, but was initialised 10 times and the minimum metric over the 10 runs was taken. The k-means

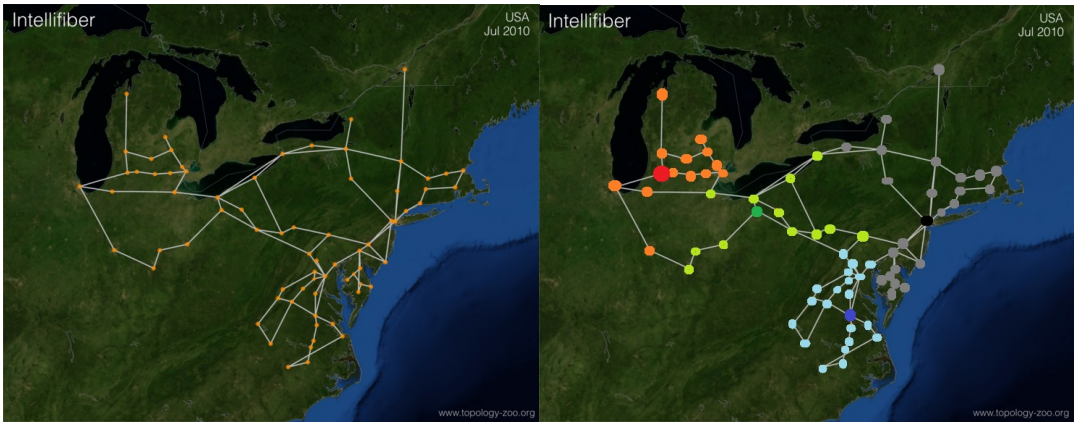


(a) Internet2



(b) Internet2, $k = 4$

Figure 3.7: Internet2 Topology



(a) Intellifiber (b) Intellifiber, $k = 4$

Figure 3.8: Intellifiber Topology

, k-means 10 and k-means 100 used 1,10 and 100 initialisations respectively, with the metric recorded being the minimum one created from all the initialisations. To compute how long the algorithms take to find a solution the methods were repeated 10 times and the average time taken is recorded in Tables 3.9 and 3.10.

Although the metrics are not optimal when compared to the linear programming method the computation time is significantly less. This is due to the complexity of the algorithms as discussed in section 3.4.4. This aims to prove that although the approximation algorithms are not guaranteed to be optimal, they are applicable to software defined networking since large scale deployments may require fast computation times. In particular, this would be a requirement in SDNs where reactive, or dynamic controller placement is required [74, 75].

We can examine the tables more easily by looking at the percentage increase (or decrease) between the metrics and timings. We can see there is a significant difference between the timing results, where in some case the k-means algorithm only takes 0.001% of the time taken by the linear program to compute a solution. This is at the expense of optimality, in the case where the timing was 0.001% (Intellifibre network k-means algorithm, $k = 8$) there is an increase in the metric value of 24.14%. There is a similar

pattern for the other methods, but as the complexity of the k-means and local search algorithms is increased using a larger number of initialisations and swaps respectively, the computation time increases. However, although the complexity is increased the optimality of the metric becomes closer to the linear program. As an example, the case for $k = 7$ from the internet2 network in Table 3.11 shows in general that the methods with an improved metric (closer to 100%) have an increased timing cost.

Another noticeable difference is that the metrics for the k-means and local search algorithms become increasingly worse in terms of the optimality of the metrics as the number of controllers, k , is increased. This is likely to be caused by the nature of the algorithms, and would need careful consideration if large values of k were required. In terms of the nature of the algorithms the linear program considers the whole network simultaneously when computing the metric, where as the k-means and local search first consider the nodes in each cluster before calculating a global metric. This may be the cause of the k-means and local search algorithms performing worse when k is increased.

Table 3.7: Internet2 metrics (Sum Latency)

	k=2	k=3	k=4	k=5	k=6	k=7	k=8	k=9
Linear Program	883.62	557.92	435.66	376.15	326.95	278.83	247.39	217.2
Local Search 100	896.15	559.96	450.75	407.37	360.04	331.78	297.74	277.3
Local Search 500	896.73	559.03	441.49	395.06	362.03	315.28	290.09	281.06
Local Search 1000	884.54	559.4	448.41	400.46	338.87	313.21	286.3	266.46
Local Search Repeat	883.62	557.92	436.32	378.23	332.08	286.97	257.39	235.33
k-means	890.17	608.86	479.32	424.35	350.22	332.83	304.42	265.19
k-means 10	887.51	589.45	445.74	376.15	337.78	301.43	269.01	248.19
k-means 100	887.51	589.45	445.74	376.15	341.03	303.08	269.03	235.48

Table 3.8: Intellifibre metrics (Sum Latency)

	k=2	k=3	k=4	k=5	k=6	k=7	k=8	k=9
Linear Program	490.37	383.16	323	289.45	264.58	244.85	222.9	201.37
Local Search 100	499.33	407.59	424.96	423.02	411.17	390.2	401.51	367.94
Local Search 500	527.2	385.23	344.55	308.43	283.15	264.42	248.16	232.04
Local Search 1000	490.37	386.87	347.78	307.15	286.15	261.99	243.78	231.27
Local Search Repeat	490.37	385.15	344.37	329	330.76	316.78	312.84	274.26
k-means	498.06	396.92	367.52	330.4	312.42	299.52	276.7	259.38
k-means 10	498.06	392.08	349.41	320.08	312.52	285.1	259.74	240.36
k-means 100	498.06	392.08	349.41	323.8	313.52	288.1	257.69	242.44

Table 3.9: Internet2 Timings (seconds)

	k=2	k=3	k=4	k=5	k=6	k=7	k=8	k=9
Linear Program	8.4435	8.4639	9.7184	10.017	11.797	10.445	10.48	9.6987
Local Search 100	0.015786	0.014202	0.015599	0.015171	0.014523	0.015426	0.020338	0.017647
Local Search 500	0.081009	0.067923	0.065046	0.069998	0.068997	0.071275	0.084785	0.078133
Local Search 1000	0.16142	0.12984	0.12933	0.15187	0.13464	0.13938	0.15692	0.15694
Local Search Repeat	0.14743	0.13551	0.13871	0.15013	0.14431	0.16958	0.17187	0.16294
k-means	0.0039841	0.0033936	0.0036259	0.0040863	0.0041911	0.0045355	0.0049368	0.0045713
k-means 10	0.012966	0.016283	0.019265	0.021508	0.020372	0.024755	0.026973	0.023377
k-means 100	0.10488	0.13174	0.1773	0.19806	0.20001	0.21891	0.23643	0.22678

Table 3.10: Intellifibre timings (seconds)

	k=2	k=3	k=4	k=5	k=6	k=7	k=8	k=9
Linear Program	269.87	283.8	379.84	392.83	410.43	530.74	1099.6	392.44
Local Search 100	0.020734	0.017936	0.017772	0.018748	0.018228	0.018852	0.021773	0.020182
Local Search 500	0.089442	0.082976	0.077945	0.075096	0.075438	0.076666	0.088991	0.083214
Local Search 1000	0.18712	0.16122	0.15001	0.15223	0.1494	0.15227	0.16976	0.16289
Local Search Repeat	0.19635	0.17866	0.17774	0.18406	0.1792	0.18699	0.2128	0.20394
k-means	0.0077553	0.0081417	0.0088793	0.0099368	0.0094055	0.01006	0.010755	0.010501
k-means 10	0.021404	0.025882	0.030345	0.038851	0.041884	0.046948	0.050511	0.053648
k-means 100	0.16548	0.20204	0.25949	0.32549	0.35996	0.4059	0.45631	0.46344

Table 3.11: Percentage changes for the metrics and timings between the Linear Program and k-means/local search methods

Internet2 Metrics

	k=2	k=3	k=4	k=5	k=6	k=7	k=8	k=9
Linear Program	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%
Local Search 100	101.42%	100.37%	103.46%	108.30%	110.12%	118.99%	120.35%	127.67%
Local Search 500	101.48%	100.20%	101.34%	105.03%	110.73%	113.07%	117.26%	129.40%
Local Search 1000	100.10%	100.27%	102.93%	106.46%	103.65%	112.33%	115.73%	122.68%
Local Search Repeat	100.00%	100.00%	100.15%	100.55%	101.57%	102.92%	104.04%	108.35%
k-means	100.74%	109.13%	110.02%	112.81%	107.12%	119.37%	123.05%	122.09%
k-means 10	100.44%	105.65%	102.31%	100.00%	103.31%	108.11%	108.74%	114.27%
k-means 100	100.44%	105.65%	102.31%	100.00%	104.31%	108.70%	108.75%	108.42%

Internet2 Timings

	k=2	k=3	k=4	k=5	k=6	k=7	k=8	k=9
Linear Program	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%
Local Search 100	0.19%	0.17%	0.16%	0.15%	0.12%	0.15%	0.19%	0.18%
Local Search 500	0.96%	0.80%	0.67%	0.70%	0.58%	0.68%	0.81%	0.81%
Local Search 1000	1.91%	1.53%	1.33%	1.52%	1.14%	1.33%	1.50%	1.62%
Local Search Repeat	1.75%	1.60%	1.43%	1.50%	1.22%	1.62%	1.64%	1.68%
k-means	0.05%	0.04%	0.04%	0.04%	0.04%	0.04%	0.05%	0.05%
k-means 10	0.15%	0.19%	0.20%	0.21%	0.17%	0.24%	0.26%	0.24%
k-means 100	1.24%	1.56%	1.82%	1.98%	1.70%	2.10%	2.26%	2.34%

Intellifibre Metrics

	k=2	k=3	k=4	k=5	k=6	k=7	k=8	k=9
Linear Program	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%
Local Search 100	101.83%	106.38%	131.57%	146.15%	155.40%	159.36%	180.13%	182.72%
Local Search 500	107.51%	100.54%	106.67%	106.56%	107.02%	107.99%	111.33%	115.23%
Local Search 1000	100.00%	100.97%	107.67%	106.12%	108.15%	107.00%	109.37%	114.85%
Local Search Repeat	100.00%	100.52%	106.62%	113.66%	125.01%	129.38%	140.35%	136.20%
k-means	101.57%	103.59%	113.78%	114.15%	118.08%	122.33%	124.14%	128.81%
k-means 10	101.57%	102.33%	108.18%	110.58%	118.12%	116.44%	116.53%	119.36%
k-means 100	101.57%	102.33%	108.18%	111.87%	118.50%	117.66%	115.61%	120.40%

Intellifibre Timings

	k=2	k=3	k=4	k=5	k=6	k=7	k=8	k=9
Linear Program	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%
Local Search 100	0.008%	0.006%	0.005%	0.005%	0.004%	0.004%	0.002%	0.005%
Local Search 500	0.033%	0.029%	0.021%	0.019%	0.018%	0.014%	0.008%	0.021%
Local Search 1000	0.069%	0.057%	0.039%	0.039%	0.036%	0.029%	0.015%	0.042%
Local Search Repeat	0.073%	0.063%	0.047%	0.047%	0.044%	0.035%	0.019%	0.052%
k-means	0.003%	0.003%	0.002%	0.003%	0.002%	0.002%	0.001%	0.003%
k-means 10	0.008%	0.009%	0.008%	0.010%	0.010%	0.009%	0.005%	0.014%
k-means 100	0.061%	0.071%	0.068%	0.083%	0.088%	0.076%	0.041%	0.118%

3.6 Discussion and Conclusion

In this chapter, we presented two algorithms that have not previously been used to find a solution to the controller placement problem. These were the local search method and the k-means adaptation, which were described in sections 3.4.2 and 3.4.3 respectively. We compared these algorithms to an existing linear programming method that was described in section 3.4.1. It was found that although the linear programming method produced the best solution with respect to the optimisation problem defined in equation 3.2, the solution required a large computation time that would not be adequate for an agile environment.

Another method for finding controller placements would be to use the facility location problem. In the facility location problem instead of having a specified number of controllers ,k, the number of controllers is increased until a specified constraint is satisfied. In this case the number of controllers would vary depending on the tightness of the constraint. This is relevant to our application as an unspecified number of controllers may be required for safety and reliability requirements. However, the facility location problem requires more computation in general, as it is an iterative process to find the correct number of controllers to satisfy the given metric. Further work would therefore be to compare the difference in computation time required between the k-median and facility location problems.

In conclusion, this chapter addresses the important issue of scalability for the placement of controllers. As the k-median problem is a known NP hard problem, the scalability of the placement of controllers is linked directly to the complexity of the algorithms used to find a solution. It was found the complexity varied significantly between the methods discussed; the linear programming method was found to be useful for networks of a limited size and the local search method and k-means++ the most scalable approaches. However, the scalability is traded off with the accuracy of the methods. It was found that although the local search method approaches the metric found by the more complex linear program, the standard deviation varies considerably. The main reason for this could be the randomness involved in finding initial

clusters, but using a more intelligent way of finding initial clusters increases the complexity as with the k-means++ method. This can also hinder the optimality of the algorithms as it was shown the k-means++ has a reduced variance in solutions, but the solutions are not necessarily as good. Overall, it was found there is a definite trade-off between complexity and accuracy.

Chapter 4

Redundancy: Literature Review

4.1 Introduction

The work in Chapter 3 addresses a design problem of how to minimise latency in the control plane in SDNs. We now turn our attention to latency in the data plane, and how SDNs can be used to reduce this. One approach, which SDN shares with other networking paradigms, is optimal routing. By routing packets along the minimum latency path between source and destination, delays are reduced. There has already been a substantial amount of work on this problem, and we do not pursue it further. Instead, we observe that SDN can enable a new approach not available in traditional networks; that of replicating data packets and using distinct routes for the replicates. In particular, the redundancy can be dynamic, and responsive to changing traffic patterns, due to the programmability that SDN provides.

To introduce the topic of redundancy, a literature review on the current research in this area is performed, which includes both mathematical modelling methods such as queuing theory, and also simulation and emulation work. The mathematical methods give a solid theoretical basis for studying delays in systems with redundancy, while much of the emulation aims to give real world examples of the benefits. The novel contributions of this thesis

to analysing the impact of redundancy on latency are presented in Chapters 5 and 6; Chapter 5 focuses on redundancy achieved through straightforward replication of packets, whereas Chapter 6 studies a more sophisticated approach based on erasure coding.

4.2 Analytical Methods

4.2.1 Queueing Theoretic Models

A queueing theoretic analysis of SDN is carried out in [76], which looks at the interaction of the switch and controller with the OpenFlow protocol. The OpenFlow protocol takes the following actions when a new flow arrives in the system:

1. A packet (or part of a packet) of the flow is sent by the switch to the controller, assuming that the switch is not configured to drop unknown packets.

2. The controller computes the forwarding path and updates the required nodes in the data path by sending entries to be added to the flow tables.

3. All subsequent packets of the flow are forwarded based on pre-calculated forwarding decisions and do not need any control plane action.

From this it is observed that for the model to represent an OpenFlow based SDN the following two points must be obeyed: (1)A packet coming to any node in the data plane will visit the controller at most once and (2) only a fraction of the external traffic and not a fraction of the net input traffic will go to the controller. The following additional assumptions are made:

- The overall traffic arrival process at the switch and the controller is Poisson.

- Service times of a packet at the switch and the controller are exponentially distributed, independent and identically distributed (iid) across packets.

- Switches have infinite buffers.
- A single queue at the switch is assumed instead of a separate queue per line card.

These assumptions are made in order to model the system as a Jackson network. The analysis then utilises known results for Jackson networks, such as the average packet sojourn time, and the distribution of time spent by a packet in the network.

The analysis is first carried out for the simplest network, with one controller and one switch. This is then extended later to include two switches, showing how the model can be enhanced for use on larger networks. The performance measures are then compared against a discrete event simulator and shown to be a good approximation. The arrival and service rates used are taken from practical systems; however, it is noted that the Poisson and exponential assumptions are limiting factors of the model. The accuracy would be increased if network traces or different distributions were used instead. This is a common problem in applying queueing theory to Internet traffic, and is discussed in section 4.5.

Queuing theoretic methods are also used to characterise the performance of SDN systems in [77].This paper describes Multipath Source Routing (MSR), which is an extension to a load balancing mechanism in an ad-hoc network known as Dynamic Source Routing (DSR). The initial model proposed looks at a source and destination with N paths in-between, each having cross traffic as shown in Figure 4.1. It is assumed that each flow entering the system is split amongst a subset of the paths; in this sense, no redundancy is added, but the flow is simply split amongst different paths. This is in contrast to DSR where only a single path is used per flow. The optimisation problem is then how to distribute each sub-flow amongst the subset of paths. The mean system delay is used as an objective function, and it is shown that by balancing load across multiple paths, MSR is superior to DSR in terms of end-to-end delay. It is further shown that the variation of the delay is also reduced with MSR. The analysis performed is relatively straight forward, but is an example of the application of queueing theory to SDN.

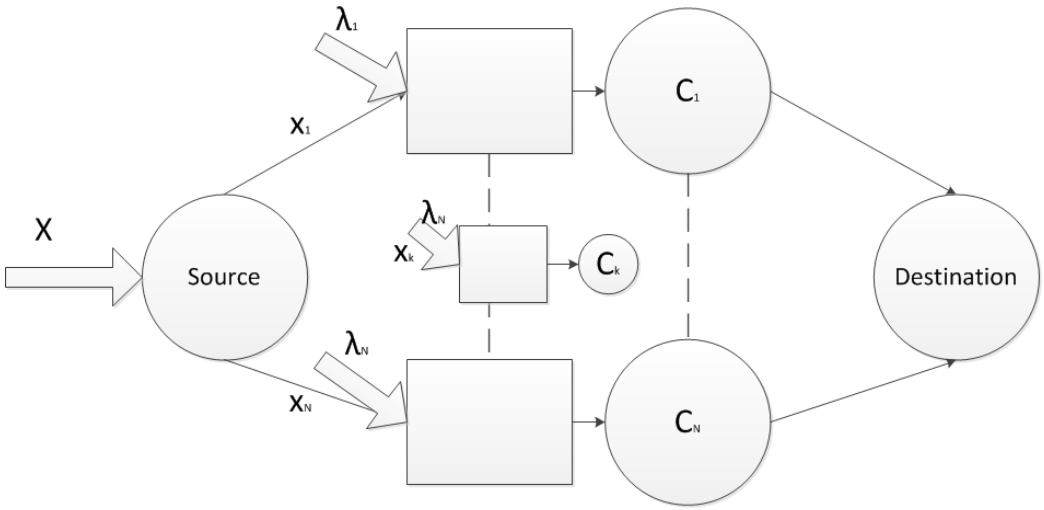


Figure 4.1: Load Balancing of Multipath Source Routing in Ad Hoc Networks - Queuing Model

4.2.2 Network calculus: A viable option?

Network calculus provides a way of modelling the network by using bounds on the arrivals and departures. The main advantage of this is that a complex network can be modelled more simply, which can then be used to give bounds on the delay and buffer size of the system. However, the model bounds the arrivals and departures with a curve, so the bounds are very conservative. This means that systems have to be heavily over-provisioned to give the required bounds supplied by network calculus; one such example is the AFDX system used in aircraft [78]. The main text on network calculus, [79], describes the basics but also extends the theory to more complicated analysis such as: schedulers, smoothing, shapers and systems with losses. However, the derived bounds remain very conservative, especially when variation is introduced as the bounds must cover all eventualities.

Perspectives on network calculus: no free lunch, but still good value [80] In this particular paper it is shown that tight bounds can be found on single queues using network calculus. However, this is not extended in general to networks of queues, although significant progression on this is

made in [81]. Some of the benefits of network calculus over queueing theory are explained such as: arrivals do not necessarily have to be Poisson, scheduling can be abstracted by suitable constructed service processes, convolution of multi node networks in to a single node, and the fact that analysis is much easier as the non-linear queueing system is approximated to a 'somewhat looking' linear system.

However, some of these benefits are not unique to network calculus. For example, the Poisson arrival process is not a necessity for queueing theory but it does make analysis easier, particularly for closed form expressions and existing theory. Although scheduling can be abstracted in network calculus, methods exist to incorporate scheduling methods in queueing theory, which makes analysis harder, but is not impossible. It is common for bounds to be formed rather than exact analysis; for example, this is part of the analysis of MDS queueing systems which is looked at in section 4.4. Furthermore, the convolution of nodes can be performed in queueing theory; for example, aggregate arrival and departure rates can be used to analyse multiple queues in a single node. Finally, the main advantage of network calculus is the fact that the system can be approximated to a linear looking system, which does make the analysis easier, but also means there is a trade-off between accuracy and ease of analysis.

An analytical model for software defined networking: A network calculus-based approach [82] This paper looks at the use of network calculus to analyse SDN, whereby both the controller and switches are considered. The paper provides bounds for the delay and buffer size based on a deterministic model; these bounds can be calculated quickly as they are analytical. A good summary of network calculus and queueing theory are provided in the paper, in particular network calculus provides worst case bounds, whereas queueing theory provides more accurate average quantities.

An important observation from the paper is that the arrivals at packet level and flow level are different. In particular, the arrival rates at the controller and network nodes need to be carefully considered. As stated in the paper the analysis provides a quick method for finding the worst case bounds

for an SDN system, but the work needs to be extended to improve the accuracy; for example, using stochastic network calculus. Another simplification is the use of the multiplexing rule where multiple switches are expressed as a single node: this further reduces the accuracy of the system but makes it easier to analyse.

Overall, network calculus is a useful tool when analysing networks. On one hand the analysis is simpler, but on the other the accuracy is reduced. The accuracy is further decreased when looking at networks of queues. This leads me to believe that queueing theory is a better approach for modelling a network, although network calculus may be used to give bounds on the systems.

4.3 Redundancy via replication

The basic idea is to transmit multiple copies of each packet along disjoint paths. The latency is the time until at least one of these copies arrives at the destination. If delays along distinct paths are modelled as independent random variables, then the latency becomes the minium of independent random variables; its expectation is smaller than that of any single one of these random variables. This suggests that replication should reduce latency, but this simple argument overlooks the impact of the increase in traffic caused by replication. If we assume that all flows adopt the same strategy, then there will be a significant increase in traffic, and it is unclear whether replication offers any benefit. One of the contributions of this thesis is to show that it does, if the system is sufficiently lightly loaded. But first, we review some of the work to date on this topic. Much of this work has focused on server farms and storage systems rather than communication networks. In particular, there does not appear to be work specifically advocating the use of replication in SDN networks.

Low latency via redundancy [83] In [83] the authors look at the trade-off between latency and load by first looking at how replication can improve latency, which is achieved by taking the minimum latency of all the requests

across all queries. A utilisation threshold value is found and analysed, which is where the replication starts to increase latency for a given arrival rate (due to congestion).

The authors initially use a queueing model. The model assumes N servers where all the servers are identical and independent. k copies of the request are made and sent to k of the N servers. Initially it is assumed redundancy does not cost anything for the user i.e. there is no additional delay, but there is an increase in server utilisation. Simulations are also performed on the model and it is found that the queuing analysis closely resembles the simulation when N, the number of servers, is increased. Emulation is then used to compare against the modelled system. This is achieved by measuring the response times from a set of client nodes using Emulab. The main findings of the paper can be summarised in to two subcategories, that of queuing theory with simulation, and that of emulation.

The main results from the queueing theory and simulation are:

- Replication improves the mean latency but is of most benefit in the tail i.e. extreme values of delay.
- If the overhead of replication equals the mean latency then replication can not improve the mean latency, but it may reduce the variation.
- If redundancy has no user cost then there is strong evidence to suggest the utilisation threshold value (the point at which it is not valuable to replicate) has to be more than 25%, and this is independent of the service distribution. In other words, it is always worthwhile to use replication when the utilisation is below 25%.
- The performance improvement is greater when there is more variability in the service time distribution; this is confirmed with a number of theorems.

For emulation ,the findings confirm the analysis performed on the queuing model, however, specific values are also given for the latency improvements. It is also shown that for a small file size changing the mean file size or

distribution does not significantly change the improvement in latency (due to the bottleneck being the latency when locating a file on disk). This is a key observation as it shows when modelling the service time for small files that the seek time is an important measure.

Overall, the paper analyses how replication affects the utilisation and at what point replication is useful. The second point is that replication is more valuable when there is little overhead. This limits the number of applications for which replication is useful. The two most beneficial uses were the ones with little overhead but big reward. These were DNS requests and connection establishment. This is because the reliability in these scenarios is key for the rest of the process to work, but the initial amount of data required is small. Another finding was that the biggest difference between using redundancy and not lies in the tail of the distribution, meaning that replication causes the distribution to be less variable.

The main criticism of the paper is that it is assumed that all traffic is replicated, not only a proportion of traffic, such as TCP connections. This means the theorems on the utilisation threshold will become invalid if only a fraction of traffic is replicated, as is likely in a real system. This point is expressed in [84], the paper below. Furthermore, although the emulation work looks at a range of the number of copies sent, the analysis of the theoretical model is restricted to 2 copies. This is a severe limitation and it would be interesting to extend the analysis to an arbitrary number of copies.

More is Less: Reducing Latency via Redundancy [84] Unlike the paper above, this paper only uses hardware to show replication offers improved performance. An important observation of the paper is that, for given networks, as the fraction of flows duplicated is increased, the total load does not increase linearly with it, e.g., duplicating the smallest 33% of flows only increases the total load by 2%. This is important as it is noted in another paper that the data centre applications that are latency critical are often small in size.

Contrary to the point that the flows which benefit most from replication are the ones that are least expensive to replicate, the authors comment

that some latency-sensitive tasks are large, such as real-time video streams. Another problem is that if the overhead is high, a replicated request could be marked as lower priority, so it can be dropped if it interferes with other work. The authors also make the argument that the need for consistent reduced latency outweighs the need to save bandwidth, for example in the wide area internet where the network is underutilised. Hence, the extra capacity could be used for redundancy. Another point explained in the paper is that although replication has been used as a way to deal with failures in DTNs (delay tolerant networks) and multi-homed web proxy overlays, it has not been widely deployed. This makes replication an important application for an SDN, as redundancy may be employed dynamically via the controller in order to improve the network.

The future research directions considered in the paper offer an interesting perspective and potential use cases, they include:

- Automating Redundancy: replicating the first k packets of a TCP flow in order to reduce latency, even if it is slight.

- Path Selection: The authors ask whether or not it would be more beneficial to choose multiple paths that are the most independent rather than the ones with the best mean performance, similar to [85].This would mean the flows would be more reliable as the probability of failure would be reduced.

- Selfishness: to look at the effect of letting users replicate their own flows and seeing how this affects performance. This is interesting as an individual replication packets would only have a slight effect on the total load of the network, but would become more prevalent as the number of users replicating increases.

- Security: whether replication would make malicious attacks more difficult as a user would be able to identify corrupt data (Replicated packets would be inconsistent).

Examples where redundancy might be useful Examples of where replication may be used in a network are repeated in the papers [83, 84] so a summary of the examples is given here.

- **Connection establishment (TCP):** the aim of redundancy for TCP is to reduce the probability of failure. It has been shown in another paper that by sending back to back copies the probability of failure reduces from $\sim$ 0.0048 to $\sim$ 0.0007 (although not from p to p^2 in general).

- **DNS:** By querying multiple DNS servers simultaneously, it can be tested whether or not the latency of a reply decreases. It was found the method worked, but when increasing the number of servers queried further, the savings have diminishing returns.

- **Memcached:** It was found when the data was cached that although replication improved the mean latency, it was only by a little.

- **Replication in data centres:** It was found improvement was greatest at intermediate loads, as at low load there was enough capacity for replication to not make a difference, and at high loads there is a hindering affect.

- **Multipath routing:** It was shown that the mean latency of the network was reduced using multipath routing, although not by very much, and the most benefit comes in the tail of the distribution.

- **Quality of Service:** It is noted that there are many benefits to replication as a mode of quality of service compared to previous methods (e.g. intserv). It has the following advantages: ease of deployment, no requirement for data plane QoS support, reuse of existing pricing models, flexibility to be applied to a sub-set of traffic and unpredictable events can be dealt with, such as router failures.

The most promising example, other than for connection establishment and DNS requests, seems to be for QoS. This is because other methods of

QoS can be complex when compared to redundancy. Furthermore, dynamic redundancy in SDN could be applied to specific flows such as TCP in order to enhance reliability and latency without significantly effecting the load on the network. Varying amounts of redundancy could also be used for varying states of QoS in order to create differentiated services. This is also particularly relevant to SDN as the control element enables dynamic control of the network, which means dynamic redundancy can be used to influence the QoS.

Best-Path vs. Multi-Path Overlay Routing [86] The paper studies the benefits of using best path vs multipath routing on a testbed. It does this by looking at the loss rate and latency of both methods. It was shown that the benefits of using multiple paths could be replicated by introducing a delay for replicated packets on the best path method. An average improvement of 6ms was achieved for latency optimised best path routing whereas there was a 2-3ms improvement for the multiple path method. It was also stated that the latency reduction was insignificant for both methods when the latencies were below 50ms. Presumably this is because as the latencies decrease there is a smaller number of 'good' paths available, or one path is much better than the others.

A limitation in this paper is the method of replication used, for both methods all the packets are replicated, rather than a select few. Furthermore, the number of paths replicated on is limited to two. Both of these points could be extended by looking at what happens when only a limited number of packets are replicated and increasing the number of paths used for replication.

Dispersity Routing [87, 88] In dispersity routing the packets are split into parts and sent over individual links, so each link transmits only a fraction of the packet. Thus, it isn't exactly replication, but it does show some of the benefits of using multiple independent paths for transmission. The first paper shows analytically that dispersity routing reduces the mean and variance of delay in a network. The second paper extends and updates the previous one, to investigate dispersity routing in current networks (2007). It discusses how

dispersity routing has become less useful for wired networks but may still be useful for wireless networks. This is because advances in wired networks in terms of reliability and capacity outweigh the enhancements dispersity routing would create. However, it is argued that this is not the case for wireless networks, which still suffer from relatively low capacity and high error probabilities.

4.3.1 Queueing with redundant requests: an exact analysis

In most of the work cited so far in this section, performance evaluation is either carried out through simulation, or only bounds are provided, or the analysis is carried out subject to a number of strong simplifying assumptions, on top of the modelling assumptions. One of the very few works to perform an exact analysis of a queueing model with replication is [89].

The paper analyses redundancy by looking at queueing models. The key insight for the paper is viewing the servers as having a single queue/buffer, in which the jobs are queued in the order they arrive (FCFS).

The main theorem is used to show the redundant data experiences a response time distribution identical to an M/M/1 queue (even though it is not M/M/1). This is an unexpected result as there are extra arrivals in the system.

It is shown that for the fully redundant traffic streams that exist in the system, the distribution of time responses are not effected even if other traffic streams become redundant. Another contribution of the model is that non-redundant data sees better performance if the other data is simply redundant rather than using an opt-split or jsq (join shortest queue) protocol. This is because there is no prioritising of the additional redundant data, which would cause the non-redundant data to be delayed. Furthermore, as the number of servers increases, the redundant traffic benefits and the non-redundant traffic suffers less. This is due to the fact that there are more servers serving both classes.

The paper also hypothesises that the response time distribution is related

to the degree of redundancy. It says that fully redundant traffic streams see exponentially distributed response times where as partially redundant or non-redundant traffic see generalised hyper-exponential distributions. This could simply be because the redundant classes see a sum of service times due to each element of a redundant class being sent to a separate server, with each having an exponential service time.

A criticism of the paper is that the arrival rates and service rates for the modelled queues are seen as aggregate rates, for example a total service rate may consist of a full rate from one server and a partial rate from another. It is also assumed that the packets are not replicated when using join the shortest queue or opt-split methods, hence the packets will only be served by a maximum of one server, rather than many (which is the assumption of the redundancy model) and therefore the comparison is not completely fair.

A limitation of the paper is that although the analysis is exact for a single node, the closed form becomes convoluted for large networks, so simulation is used. This limits the applicability of the exact analysis. Another limitation shown is that having a shared server which serves all the arrivals becomes overloaded when the number of independent arrival flows increases. This simply shows that a load threshold has to observed.

Perhaps one of the most useful sections from this particular paper is the section on related literature. The section shows other queuing models for redundancy are available but explains that exact analysis remains 'elusive'. The following points provide direction for further study: coupled processor systems, fork join systems, and systems with flexible servers. All the models share the same theme that they need flexibility in the network to be able to be implemented. The most promising model from the related literature section looks to be the (n, k, r) system model and this is similar to methods looked at in section 4.4.

4.4 Redundancy via coding

An alternative to replication is to split a packet into chunks and supplement these with additional coded chunks in such a way that the packet can be

recovered from only a subset of them. An (n, k) erasure code has the property that it starts with k chunks, and adds $n - k$ coded chunks, such that the original k chunks can be recovered from any subset of k out of the n chunks. When this idea is applied to coded transmission, it means that the message packet is split into k chunks, coded into n chunks in total, and these are transmitted along independent paths; the packet is recovered as soon as the k^{th} chunk reaches the receiver (assuming negligible reconstruction overhead). Thus, the latency is now the k^{th} smallest of n independent transmission delays rather than the smallest. On the other hand, the traffic overhead is smaller, as the traffic intensity is only increased by a factor of n/k. There have been a number of recent packets studying this idea, mainly in data centre scenarios. We survey some of the main ones below.

On the latency and energy efficiency of erasure-coded cloud storage systems [90] The focus of this paper is the analysis of how a varied coding rate effects the latency and network utilisation on cloud storage systems. As coding is used, the model incorporates the fact that the data is split into parts. This paper uses existing queueing theory on Fork-Join systems to give bounds for the latency; the bounds are shown to be good when compared against simulations. Latency is shown to be inversely proportional to energy efficiency. The title is therefore a bit misleading as the authors focus on the latency rather than energy efficiency in the analysis.

The analysis of the paper shows that increasing the coding rate reduces the network bandwidth, but increases latency. This means, in the limit, the minimum latency is found when the data is not coded as only one server needs to be accessed to complete the process. Furthermore, it was shown that there exists an optimal number of servers which maximize energy efficiency and provide near minimal latency. This shows the number of servers is a consideration when looking at the latency, not just the coding rate and amount of redundancy. Another contribution was that the authors showed an optimal code rate exists for heavy-tailed service distributions. Lastly, it was shown that increasing redundancy for a data class helps to reduce its average latency and as a consequence, the overall latency decreases.

The limitations of the paper are within the model used. The model assumes no penalty for removing redundant requests, but once a process is completed, the other requests in that group immediately exit the system. This is an unrealistic assumption and is considered in other papers. Furthermore, the model looks at multiple servers with separate queues but it is assumed that there is complete synchronisation between servers, so all servers know when a process has been completed. This is also an unrealistic assumption, although a solution using wireless synchronisation is considered in [91].

The MDS queue [92], [93], [94] These papers analyse a data storage system using MDS (Maximum Distance Seperable) codes with the use of queueing theory. The analysis looks at two simpler methods that bound the performance of an MDS queueing system. Quasi-birth-death (QBD) processes are used to analyse the simpler methods, and their stationary distributions are found using a software tool: SMCSolver. It is shown the approximation methods give performance characteristics very close to a simulated MDS queueing system.

The model has a total of n servers whereby requests enter into a (common) buffer of infinite capacity. Although the file sizes are assumed to be identical, the service distribution being exponential is attributed to the variability of the storage disks. As the system uses MDS codes, each request comprises a batch of k jobs that can be served by an arbitrary set of k distinct servers on a first come first served basis (FCFS). However, the assumption that there is a single buffer for all the servers is a limitation as this is similar to assuming an M/M/C rather than an M/M/1 queue, which means requests can be serviced simultaneously.

Two theorems in [94] show the effects of including replication in the system. The first proves that having more redundancy reduces the latency and the second shows that optimum amount of redundancy is equal to the amount of servers available. However, the two theorems do not consider having a penalty for removing requests.

When the model is extended to have a penalty in [93] and [94], whenever

a job being processed by a server is cancelled, the server must stay idle for a time which is distributed exponentially with a rate 10x the service distribution. This, in some sense, corresponds to a cancellation cost of 10%. It is shown that with a penalty, there is a threshold for the arrival rate where the latency starts to increase with an increased amount of redundancy. A summary of the findings when using a penalty are given in [93], the results are displayed in table 4.1.

[93] and [94] also include general service times where heavier tails than the exponential are considered. By looking at different service distributions, it is concluded that if a server has a sufficiently high service rate then redundancy has little effect as it will not improve the performance. Conversely, when a service rate is lower, then redundancy will improve the service as there is more chance for the request to be completed.

n	k	Arrivals	Service (i.i.d.)	Buffers	Penalty	Load	Optimal policy
Any	1	any	memoryless	centralized	0	Any	send to all
Any	any	any	memoryless	centralized	0	Any	send to all
Any	1	any	heavy-everywhere	centralized	0	High	send to all
Any	1	any	light-everywhere	centralized	Any	High	no redundancy
Any	1	any	memoryless	centralized	>0	High	no redundancy
Any	any	any	memoryless	distributed	0	Any	send to all
Any	1	any	heavy-everywhere	distributed	0	High	send to all
Any	1	any	light-everywhere	distributed	Any	High	no redundancy
Any	1	any	memoryless	distributed	>0	High	no redundancy

Table 4.1: Service distributions: heavy (light) everywhere means it is heavy (light) tailed compared to exponential. I.e. for a light tail distribution the bulk of the probability is close to zero, for a heavy tailed more probability is at the tail than the exponential so it is more likely for something to take longer.

An interesting aspect of the most recent paper ,[92], is the discussion on degraded reads whereby partial data can be recovered from any d ($d \leqslant n$) servers, i.e. if a failure has occurred while less than n servers remain. It is shown that the degraded reads have a lower latency than that of reconstructing the entire file. This is an interesting point which could be applicable to

certain applications where latency is of high criticality but the data can still be understood by approximation, an example would be audio applications.

Overall the papers give a good insight into how to analyse a system through approximation, specifically by bounding the latency through simpler methods, and then comparing this to a simulated version of the more complex method. The models are also extended to analyse a penalty for removing redundant requests and the papers show it is an important aspect to consider. Furthermore, the papers use different service distributions which are important to improve the accuracy of the model. However, only a fixed rate penalty is considered which limits the analysis and a variable penalty could be considered in future work.

A retrial queue with redundancy and unreliable server [95] In this paper a system is considered where the servers have no buffers. Instead, the data sent is retransmitted if it is not served i.e. a retrial queue. Further to this, the model is extended to consider redundant packets where a separate arrival process is considered for the redundant requests. The paper is not particularly well written or clear but the references to other papers are useful. The analysis uses queuing theory and performance measures are derived, but ultimately the paper would only be useful when considering a system with no buffers.

4.5 Enabling Redundancy, A Whistle Stop Tour

It has become clear that if we want to increase reliability and reduce latency in the network a key mechanism needed is for independent paths in the network. Although not explored in depth, it is useful to know there exists polynomial complexity methods for finding the k shortest paths in a network. This is useful as existing algorithms can be used for any work requiring more than one path in a network. These methods fall into two categories: where loops in the path are allowed and where loops are not allowed (loop-less).

One basic method for finding loop-less paths is explored in [96] where the shortest path is found with a complexity order of $O(KN^3)$, where K is the number of paths required and N is the size of the network. Another method for finding k-shortest paths (may contain loops) [97] has complexity order $O(m+nlog(n)+k)$ where m is the number of edges, n is the number of vertices and k is the specified number of paths. A method to find disjoint paths is described in [98], and a protocol that finds the path with the minimum delay and then uses a path that is 'maximally disjoint' for the second path is described in [85]. Overall, as long as methods for finding k-paths exists, the redundancy approach can be used.

An extension in some of the papers is to include a delay to simulate a penalty for removing redundant requests. A practical example is explored in [91], where wireless cards are used to transfer information between servers. The paper gives multiple algorithms for online and offline methods. A perfect channel is assumed in the model, which they state to be unrealistic. Along with many of the other papers, the mechanism is tailored for data centres.

In conclusion, this provides evidence that some of the techniques required for redundancy in networking is possible with the enabling features described above. Further ways redundancy can be achieved is to fully exploit SDN as an enabler. An SDN can employ the rules for packets to be sent along multiple paths, as provided by the k-shortest paths algorithms, and redundant requests may be eliminated using such methods as a low bit rate, or inexpensive wireless channel to provide additional information to the nodes in the network.

Next, we look at some of the assumptions that are often made when modelling networking behaviour to help simplify the analysis. One very common assumption is that the arrival process in a network can be modelled as a Poisson process.

Where mathematics meets the internet [99] Although the paper dates back to 1998, and the internet is constantly expanding, many of the comments in the paper are useful. For example, it is noted that Poisson arrivals are well suited aggregated arrivals, which supports the Palm-Khintchine theorem

that states the superposition of many independent and properly normalized renewal processes forms a Poisson process. However, a Poisson process may be ill-suited to internet traffic modelling due to the burstiness involved. Evidence is given to support this claim where it is shown the Poisson based model traffic smooths out as the time scale is increased, however, this is not the case for the measured data.

In order to find a distribution that is a better fit for internet traffic, Fractal distributions are introduced. It is shown that fractal distributions give much more realistic results, especially at different time scales, which is not true for Poisson. However, the downside is that the measured data is inconsistent as the data changes when the time and location of the traffic is changed.

Another contribution of the paper is that it shows the exponential growth of internet traffic between 1985 and 1995, a trend which has continued. Also, it is stated that there is solid evidence that the arrival times of humans to the internet is Poisson. This shows that the underlying arrival process may be Poisson but the computer systems themselves change the traffic distribution.

In conclusion, the paper provides a good insight in to another method for predicting internet traffic, rather than a Poisson process. Although voiding the Poisson assumption makes analysis in queueing theory much more difficult it may improve the accuracy. The distributions may then also be applied to simulation models for queuing and compared against the Poisson model and traffic traces.

A nonstationary Poisson view of Internet Traffic [100] This paper looks at whether the Poisson assumption is now valid for an increased amount of traffic, as the previous comparison was 10 years previous. In order to fight for the Poisson process, multiple traces are used to analyse the traffic pattern. It is found that backbone traffic is well described by an exponential distribution, with packet sizes and inter arrival times appearing independent. This is confirmed using the Box-Ljung statistic which provides a 95% confidence interval. Furthermore, for links that are overprovisioned, the distribution tends to contain most probability in the 'idle' position. It is found that the

distributions of the duration of the busy/idle period, as well as the number of packets or bytes in a busy period, are well approximated by exponential distributions. Linear least squares fitting shows the CCDF (complementary cumulative distribution function) of two of the traces is well described by an exponential with confidence 99.99% and 99.89%. The conclusion, therefore, is that the Poisson assumption is well suited to backbone traffic, probably due to the aggregation of flows.

4.6 A Simplified Analytical Model

The literature review demonstrates that adding redundancy is a promising approach to reducing latency in both storage and communication systems. In order to quantify the potential benefits, we need an analytical model of the system with redundancy. Below, we present one such model, which does involve significant simplifying assumptions. While this is a drawback, it is still an advance on much of the work to date, which assumes both Poisson arrivals and exponential service times. We relax the latter assumption.

The model proposed is shown in Figure 4.2. The figure shows an arrival rate, λ, corresponding to the rate that flows enter the system. Each flow is replicated k times and feeds k independent paths, which results in a total arrival rate of $k\lambda$. Each path then has an independent service, but importantly the wait time for any particular flow is the first flow to traverse the system, so the wait time can be taken as the minimum over all of the paths. Using this as a base model the effects of having k-paths with redundancy can be analysed using queueing theory. This is formalised in Chapter 5 where the theoretical analysis is presented.

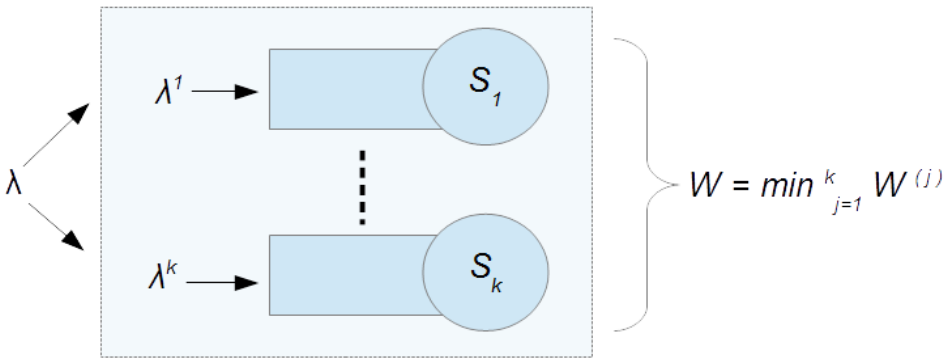


Figure 4.2: Networking with Redundancy Queuing Theory Model

Chapter 5

Redundancy: Replication

Much of the work in this chapter was published in the Proceedings of the 29th International Teletraffic Congress (ITC 29), which was technically co-sponsored by the IEEE Communications Society (IEEE ComSoc) and the ACM Special Interest Group on Communications (SIGCOMM) [101].

5.1 Introduction and Related Work

The work in this chapter is motivated by high-criticality applications running over wired or wireless networks, which require guaranteed low latency. One technique that has been studied to reduce latency in wired networks is redundancy, as reviewed in the last chapter. By sending multiple copies of the same data, the latency may be reduced as it is the minimum time to receive any one of the transmitted replicates. The downside is the increased load on the network. This leads to a trade-off between the number of replicates and the delay. It should be noted that redundancy is not the same as retransmission since the redundant packets are transmitted at the same time as the original.

The programmability of SDN makes it possible to transmit multiple copies of a packet along distinct and specified routes - something that cannot easily be achieved by existing protocols such as TCP. The work in this chapter therefore promotes the use of replication in networking using mathematical

arguments and exploits SDN as the enabling technology. This is an idea that has been tested and applied in data centres for the back-up of data, where SDN has been used to simultaneously route the replicated data to a primary and back-up server [102]. A system and method for creating a replication-aware SDN is described in a patent filed by Cisco Technology Inc. showing the applicability of replication to SDN [103]. SDN has also been considered as an enabling technology when problems arise using replication, such as deleting unnecessary replicates [104]. This shows the work presented in this chapter is theoretically possible, and is achievable using SDN as an enabler.

The current literature mainly focuses on data centres, and much of it, especially analytical results, is restricted to exponential service times. In addition, temporal variability in channel conditions, which is common in wireless networks, is not considered. The contributions in this chapter are as follows. After presenting results for exponential service times as a baseline, the framework is first generalised to consider phase-type service distributions. These provide a rich modelling framework as they can be used to approximate the distribution of any non-negative random variable. Secondly, explicit models of time-varying channels are considered, motivated by wireless networks. In both cases, an approach to deriving exact expressions for the latency is outlined. However, these expressions are extremely unwieldy, and an approximation scheme based on approximating delays by the dominant exponential term is presented. It is also used to derive an optimal replication factor that minimises the expected delay for any given load. Extensive simulation results are presented to validate the approximation scheme. Finally, applications that require stringent latency guarantees are considered. By analysing the tail of the delay distribution, probabilistic guarantees on the latency are derived, and tested against simulations.

Related work on redundancy dates back to Maxemchuk [87], who first proposed dispersity routing. The original analysis assumes exponentially distributed interarrival and service times, but the work is extended in [105] by looking at blocking probabilities of switches. The work presented in this chapter differs by looking at retransmission probabilities instead, motivated by wireless networks with unreliable and time-varying channels.

Vulimiri *et al.* [83] study the use of redundant requests to multiple servers in data centres. A variety of service distributions are considered, but the analysis is restricted to replicating each request exactly twice; the main metric considered is the threshold value of load at which replication is no longer beneficial. Shah *et al.* [106] look at the average latency with exponential, heavy tailed and light tailed service distributions. The analysis performed aims to give the optimal policy, but is limited to a binary answer of 'send to all' or 'no redundancy'. In contrast to these papers, the work presented in this chapter identifies the optimal level of replication across the whole range of loads.

Gardner *et al.* [107] analyse the benefits of redundancy when there are also independent non-redundant arrivals in the system. They also compare redundancy with the opt-split and JSQ (join-Shortest-Queue) policies. The paper gives exact analysis for the models presented, but only exponential service times are considered. The model studied allows for the cancellation of redundant requests when they are no longer needed, i.e., when a copy has completed service. Such cancellation may not be possible in real-world systems. Therefore, the work presented in this chapter does not allow for cancellation of work that has entered the system. It also considers phase type service distributions, and analyses both the mean and the tail of the latency distribution.

5.2 Models and Problem Formulation

The analysis in this chapter pertains to the latency of a single index flow in a general communication network, wired or wireless, that carries a large number of flows between many different source-destination pairs, and along multiple routes. However, these details are abstracted out. The analysis focuses on a single source-destination pair, and assumes that a fixed number k of vertex-disjoint paths between them are given. In other words, the routes to be used are fixed in advance and cannot be changed based on dynamic traffic information. Each packet of the flow is replicated across all k paths; the special case $k = 1$ corresponds to no replication. The metric of interest

is the delays incurred by packets from this index flow, with all other packets being treated as cross-traffic.

The main modelling assumption is now as follows: it is assumed that each route can be modelled as a single-server first-in-first out (FIFO) queue, and that these queues are *mutually independent.* While independence is a strong assumption, it is expected that it is not too unrealistic in large and well-connected networks. If the number of nodes and links is large, and there is a large number of available vertex-disjoint routes between any pair of source and destination nodes, then it is expected that distinct routes carry traffic multiplexed from mostly non-overlapping subsets of source-destination pairs, and hence are largely independent. An implicit second assumption is that each route carries traffic from a large number of flows, and that the contribution of any single one of these flows to the total traffic is negligible. Thus, even though the index traffic stream is common to the routes along which it is replicated, this introduces no dependence because its own contribution to the total traffic along any of these routes is negligible; the waiting times it incurs are almost entirely due to the service times of cross-traffic. An additional assumption in this chapter is that, once a packet is replicated, it remains in the system until served. This is in contrast to purging replicated packets where, once any of the replicated packets are served, the remaining replicates are removed. As a result, having replicated packets increases the total load on the network, and therefore a penalty is incurred for replication. There is thus a trade-off between replication and delay, which is the focus of the analysis in this chapter.

Given the above assumptions, a very simple the system model of k parallel queues is obtained. Packets of the index flow arrive into the system according to a Poisson process. Each packet is replicated across k parallel queues, where k is a specified parameter; the case $k = 1$ corresponds to no replication. Next, each of the queues is itself modelled as an $M/G/1$ queue; in other words, the j^{th} queue sees Poisson arrivals at rate λ_j, the customers (packets) have independent and identically distributed (iid) service times denoted $S_i^{(j)}, i \in \mathbb{Z}$ with a general distribution, and they are served in their order of arrival. While the aggregate arrival rate λ_j into the j^{th} queue will depend on the

replication factor k, the dependence is not made explicit in the notation. Let $\rho_j = \lambda_j \mathbb{E}[S_1^{(j)}]$ denote the load in the j^{th} queue; we assume that $\rho_j < 1$ for all j between 1 and k, i.e., that all queues are stable. Under this assumption, each queue has an invariant distribution under which waiting times are almost surely finite. Let $W^{(j)}$ denote a random variable with invariant distribution of the waiting time in the j^{th} queue. Then, the latency in steady state has the same distribution as the random variable

$$W = \min_{j=1}^{k} W^{(j)}. \tag{5.1}$$

This random variable, and its distribution, are the primary object of study in this chapter. By the independence assumption on the queues, we have

$$\mathbb{P}(W > x) = \mathbb{P}\Big(\bigcap_{j=1}^{k} W^{(j)} > x\Big) = \prod_{j=1}^{j} \mathbb{P}(W^{(j)} > x). \tag{5.2}$$

Closed form expressions for the CDF of $W^{(j)}(\cdot)$ are typically not available, except in the $M/M/1$ queue, and so the evaluation of the above expression is not completely trivial. Expressions for the Laplace transform of this distribution are more readily available.

The main novel contribution of this chapter is to derive tractable approximations and tail bounds on the distribution of the latency, W, for service times belonging to the class of phase-type distributions, defined below. It is also demonstrated how a scenario of practical interest, namely an On-Off channel, can be modelled using a combination of phase-type service time distributions, and an $M/G/1$ queue with exceptional first service. The analytical results for all the models studied are supplemented with simulations, and are also used to obtain expressions for the optimal value of k, the replication factor.

We first recall results on the waiting time distribution of the $M/G/1$ queue. We lighten notation by focusing on a single queue and dropping unnecessary subscripts and superscripts. Consider a single-server FIFO queue into which customers (or packets) arrive according to a Poisson process of

intensity λ. The time required to serve customer i is random, and is denoted S_i; the random variables S_i are assumed to be iid, with cumulative distribution function (CDF) $G(\cdot)$. The Laplace-Stieltjes transform (LST) of the service time distribution is denoted by $g(\cdot)$, and is defined as follows:

$$g(x) = \mathbb{E}\left[e^{-xS_1}\right] = \int_0^\infty e^{-xt} dG(t). \tag{5.3}$$

The traffic intensity (or load) is defined as $\rho = \lambda \mathbb{E}[S_1]$, where $\mathbb{E}[S_1]$ is the mean service time. It is assumed that $\rho < 1$, i.e., that the queue is stable. Let W denote a random variable with the waiting time distribution in steady state, and let $W^*(s) = \mathbb{E}[e^{-sW}]$ denote its LST. Then, we have by the Pollaczek-Khinchin (P-K) formula (see [108], for example) that

$$W^*(s) = \frac{(1-\rho)s}{s - \lambda(1 - g(s))}. \tag{5.4}$$

Next, we introduce the class of phase-type distributions, to which we will apply the above result. A phase-type distribution is defined as the distribution of the time to absorption in a finite-state absorbing Markov process started in a specified distribution. In more detail, a phase-type (PH) distribution is parametrised by a sub-probability vector $\boldsymbol{\alpha} \in \mathbb{R}^m$ (a vector with non-negative elements whose sum is no bigger than 1) and a subgenerator matrix $S \in \mathbb{R}^{m \times m}$, namely an $m \times m$ matrix all of whose off-diagonal elements are positive, none of whose row sums is positive, and at least one of whose row sums is negative. The phase-type distribution with these parameters, denoted $PH(\boldsymbol{\alpha}, S)$, is defined as the time to absorption of the Markov process with generator

$$Q = \begin{pmatrix} 0 & \mathbf{0} \\ \mathbf{s^0} & S \end{pmatrix},$$

and initial distribution $(\alpha_0, \boldsymbol{\alpha})$, whose unique absorbing state is the first state. Here $\alpha_0 \geq 0$ is such as to make the elements of the row vector $(\alpha_0, \boldsymbol{\alpha})$ sum to 1, and the column vector $\mathbf{s^0}$ is such as to make the rows of Q sum to zero, so that it is a generator matrix. It is possible to explicitly write down the CDF $G(\cdot)$, and its LST $g(\cdot)$, for the phase-type distribution. It is given

by

$$\begin{aligned} G(t) &= 1 - \boldsymbol{\alpha} e^{tS} \mathbf{1}, \\ g(x) &= \alpha_0 + \boldsymbol{\alpha}(xI - S)^{-1} \mathbf{s^0}, \end{aligned} \tag{5.5}$$

where I is the $m \times m$ identity matrix. Note that the distribution $G(\cdot)$ has an atom of size α_0 at zero, and is continuous on $(0, \infty)$. The $\text{Exp}(\mu)$ distribution is a special case with $\boldsymbol{\alpha} = 1$ and $S = -\mu$.

Phase-type distributions can be used to model a wide variety of application scenarios. The service time of a packet includes not only the physical transmission time, which would only depend on the packet size and the link bandwidth, but also a random "contention time". The contention time might either be the time to be scheduled at a switch in a wired network, or the time to access the channel under a medium access control (MAC) protocol in a wireless network. These times could have rather general distributions, and phase-type distributions have the advantage of being able to approximate the probability distribution of any non-negative random variable. Similarly, wireless channels are inherently time-varying due to conditions such as fading. Consequently, the service time over such a channel would exhibit wide variability, which can be captured by phase-type distributions.

5.3 Replication and Mean Latency

In this section, the impact of replication on mean latency in three different queueing models is studied. It is assumed throughout that packets of the index flow arrive into the network according to a Poisson process, and are replicated across k routes. Different models for the service time at a queue are considered. In order to account for both the costs and benefits of replication, it is assumed that all traffic into the network, including cross-traffic encountered by the index flow, is replicated by the same factor, k. Hence the total arrival rate into a queue is denoted by $k\lambda$.

5.3.1 Exact latenices for $M/M/1$ queues

Service times are assumed to be iid with an Exp(μ) distribution. While this is a highly simplistic assumption, it can be partially justified on the basis that the service time is comprised mostly of the time to serve packets of all queued cross traffic; the latter is known to have exponentially decaying tails in considerable generality (see, e.g, [109]). Thus, the model is not as unrealistic as it might first appear, especially for high-bandwidth wired networks multiplexing a large number of independent flows, each of which individually contributes a small fraction of the total traffic. The service rates μ_i in different bottleneck queues would typically be different. For notational convenience, and to facilitate graphical display of the results, the number of parameters in the models is reduced by taking $\mu_i = \mu$, a constant. This is also the most interesting parameter setting as the benefits of replication are greatest when delays are similar across the available routes. If mean delays differ vastly across the routes, then the same route will usually achieve the minimum, and the benefits of diversity will be minimal.

The LST of an Exp(μ) service time distribution is $g(s) = \frac{\mu}{\mu+s}$. The effect of replicating every packet across k edge-disjoint routes is to increase effective arrival rates by k. Therefore, we denote the arrival rate into the queue by $k\lambda$ and the traffic intensity by $k\rho$. Substituting these into the P-K formula (5.4), the LST of the waiting time distribution is given by

$$W^*(s) = (1 - k\rho) + k\rho\frac{\mu(1 - k\rho)}{\mu(1 - k\rho) + s}. \tag{5.6}$$

Inverting $W^*(s)$, the probability density of the waiting time is given by

$$f_W(t) = (1 - k\rho)\delta(t) + k\rho(\mu - k\lambda)e^{-(\mu-k\lambda)t},$$

where δ denotes the Dirac delta. In other words, the waiting time distribution is a mixture of an atom at zero, and an Exp($\mu - k\lambda$) distribution. The corresponding CDF is

$$F_W(t) = 1 - k\rho e^{-(\mu-k\lambda)t}. \tag{5.7}$$

Let L_k denote the latency, which is the minimum of k iid waiting times. Substituting (5.7) in (5.2), we obtain that

$$\mathbb{P}(L_k > t) = \prod_{i=1}^{k} k\rho e^{-(\mu - k\lambda)t}.$$

Thus, the CDF and mean of the latency are given by

$$\begin{aligned} F_{L_k}(t) &= 1 - (k\rho)^k e^{-k(1-k\rho)\mu t}, \\ \mathbb{E}[L_k] &= \frac{(k\rho)^k}{k(1-k\rho)\mu}. \end{aligned} \tag{5.8}$$

The CDFs are plotted in Figure 5.1 for $\lambda = 0.1$, 0.2 and 0.3, and $k = 1$, 2 and 3. Observe that the CDF plots can cross each other, meaning that no single value of the replication factor k may be optimal for all t. However, at a low load the CDF plots for $k = 2$ and 3 are above those for $k = 1$ for a wide range of t, demonstrating that replication significantly increases the probability of meeting a specified delay bound t.

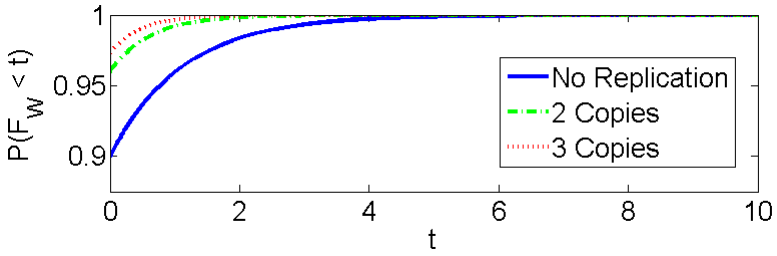
5.3.2 Approximate latencies for $M/PH/1$ queues

The exponential distribution is a special case of the much larger class of phase-type distributions, which were described in Section 5.2. The following lemma yields the LST of the waiting time distribution in an $M/PH/1$ queue; it is an immediate consequence of the P-K formula, and is well known, so the proof is relegated to the appendix.

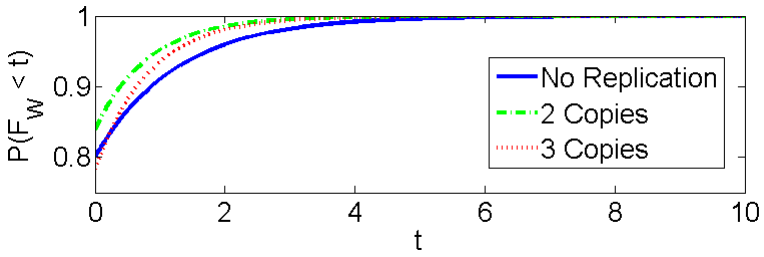
Lemma 5.3.1. *Consider an $M/PH/1$ queue with Poisson(λ) arrival process, and iid service times with a $PH(\boldsymbol{\alpha}, S)$ distribution of order m. The LST of the waiting time distribution is given by:*

$$W^*(s) = \frac{1 + \lambda \boldsymbol{\alpha} S^{-1} \mathbf{1}}{1 - \lambda \boldsymbol{\alpha} (sI - S)^{-1} \mathbf{1}}, \tag{5.9}$$

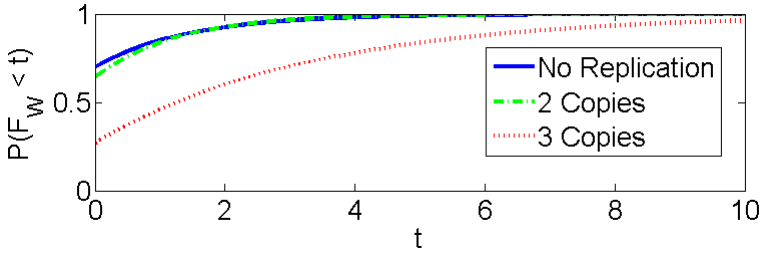
where I denotes the $m \times m$ identity matrix, and $\mathbf{1}$ a column vector of all ones of length m.



(a) $\lambda_k = 0.1$



(b) $\lambda_k = 0.2$



(c) $\lambda_k = 0.3$

Figure 5.1: Latency CDF with exponential service under varied load: *note scale changes on vertical axis*

Proof. The proof is contained in Appendix A.1. □

The lemma shows that $W^*(s)$ is a rational function (ratio of polynomials) in s. For generic values of the matrix S, the denominator polynomial has distinct roots $z_i(\lambda)$, $i = 1, 2, \ldots, m$, and so $W^*(\cdot)$ admits the partial fraction expansion

$$W^*(s) = c_0(\lambda) + \sum_{i=1}^{m} \frac{c_i(\lambda)}{s - z_i(\lambda)}, \tag{5.10}$$

where the dependence of the roots z_i and the coefficients c_i on λ has been made explicit in the notation. The z_i could be complex in general. But if

they are all real, all c_i are positive and all z_i negative, then the LST is easily inverted to yield the waiting time density

$$f_W(t) = c_0(\lambda)\delta(t) + \sum_{i=1}^{m} c_i(\lambda) e^{z_i(\lambda)t},\ t \geq 0. \tag{5.11}$$

In words, the waiting time is a mixture of an atom at zero (which must have mass $1-\rho$, as this is the probability of the queue being empty) and $\text{Exp}(-z_i(\lambda))$ distributions. This observation motivates the proposition of the following approximation for the waiting time distribution. Define

$$\eta(\lambda) = -\max_{i=1}^{m} \text{Re}(z_i). \tag{5.12}$$

Assume that $\eta(\lambda) > 0$. Then, $\eta(\lambda)$ captures the dominant term, or more precisely the slowest decaying exponential, in the expression for the waiting time density. This leads to the approximation of the waiting time density and CDF by

$$f_W(t) \approx (1-\rho)\delta(t) + \rho\eta(\lambda)e^{-\eta(\lambda)t}, \quad F_W(t) \approx 1 - \rho\eta(\lambda)e^{-\eta(\lambda)t}, \tag{5.13}$$

which has the same form as in the $M/M/1$ case, with $\eta(\lambda) = \mu - \lambda$. It is easy to calculate the latency using this approximation. As the arrival rate increases to $k\lambda$ and the traffic intensity to $k\rho$ for k-fold replication, the latency has CDF and mean are given by

$$F_{L_k}(t) \approx 1 - (k\rho)^k e^{-k\eta(k\lambda)t}, \quad \mathbb{E}[L_k] \approx \frac{(k\rho)^k}{k\eta(k\lambda)}. \tag{5.14}$$

Example: Hyper-exponential service times

The special case of a hyper-exponential service time distribution, namely a mixture of exponentials, is now illustrated. Consider a two-component mixture with density

$$f(t) = p\mu_1 e^{-\mu_1 t} + (1-p)\mu_2 e^{-\mu_2 t}.$$

This is a $PH(\boldsymbol{\alpha}, S)$ distribution with

$$\boldsymbol{\alpha} = \begin{pmatrix} p & 1-p \end{pmatrix}, \quad S = \begin{pmatrix} -\mu_1 & 0 \\ 0 & -\mu_2 \end{pmatrix},$$

and can be used to model highly variable service time distributions by taking μ_1 and μ_2 to be very far apart. Straightforward but tedious calculations now yield that the LST of the waiting time distribution is given by

$$W^*(s) = \frac{(1-\rho)(s+\mu_1)(s+\mu_2)}{(s+\mu_1)(s+\mu_2) - \lambda s - \lambda(p\mu_2 + (1-p)\mu_1)}, \tag{5.15}$$

where

$$\rho = \lambda \mathbb{E}[S] = \lambda\Big(\frac{p}{\mu_1} + \frac{1-p}{\mu_2}\Big).$$

This can be inverted explicitly to get the waiting time density and CDF

$$f_W(t) = (1-\rho)\delta(t) + \frac{e^{-x_1 t} - e^{-x_2 t}}{z_1 - z_2}, \quad F_W(t) = 1 - \frac{x_2 e^{-x_1 t} - x_1 e^{-x_2 t}}{x_1 x_2 (z_1 - z_2)} \tag{5.16}$$

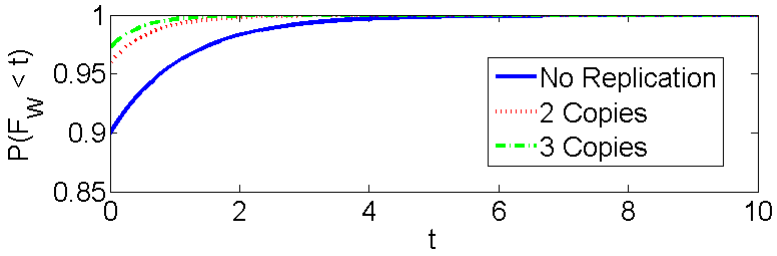
where z_1 and z_2 are the roots of the denominator polynomial in (5.15), and

$$x_i = -(1-\rho) z_i (z_i + \mu_1)(z_i + \mu_2).$$

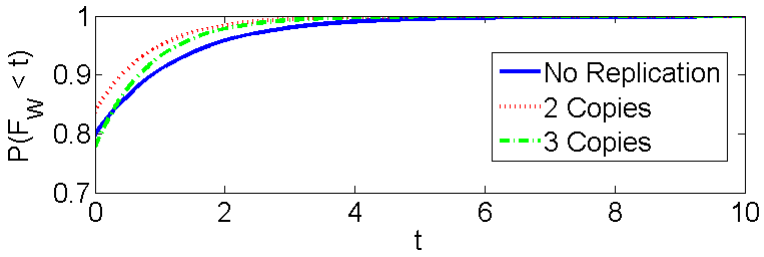
Note that $\eta(\lambda) = \min\{x_1, x_2\}$ is the dominant exponential term in the waiting time distribution.

It is straightforward to calculate the latency when replication is employed, using the explicit waiting time distribution given in (5.16). The corresponding CDFs are plotted in Figure 5.2, for the parameters $\mu_1 = 1$, $\mu_2 = 0.1$, $p = 0.999$, and three different values of the arrival rate. To demonstrate the validity of the approximation by the dominant exponential given in (5.16), Figure 5.2(d) shows the CDFs of the approximation and the exact density, which are seen to be very close.

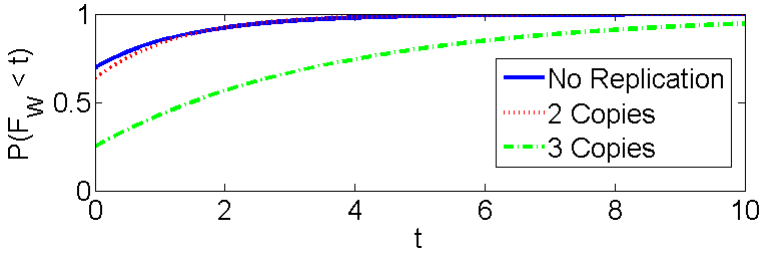
The results seen in the figures are similar to those in the exponential case, and show that the benefits of replication are most apparent at low loads.



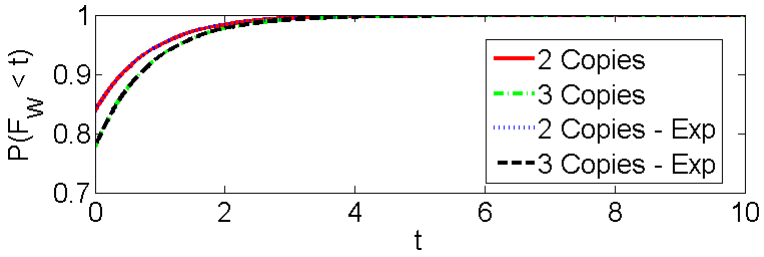
(a) $\lambda = 0.1$



(b) $\lambda = 0.2$



(c) $\lambda = 0.3$



(d) Dominant exp. approx. - $\lambda = 0.2$

Figure 5.2: CDF of the wait time with Hyper-Exponential service

5.3.3 Channel or server variability

The model of exponential service times is now returned to but, motivated by wireless channels, whose capacity varies over time due to phenomena such

as fading, assume that the server works at a variable rate. The service rate is modelled as a Markov process that evolves independently of the arrival process. Let R denote the rate matrix (generator) of this Markov process, on a finite state space $\{1, 2, \ldots, m\}$, and let μ_i denote the service rate when the Markov process is in state i. This model shares many similarities with the phase-type service distribution. In fact, conditional on the initial state of the Markov process, the service time is of phase type. However, the initial state itself is dependent on the length of the busy period that has elapsed, and so service times are not iid and the P-K formula cannot be applied. The model can be analysed using matrix-geometric methods, but rather than introduce those here, a special case that is of interest in its own right is considered, and which can be handled by transform methods.

This is the case of an On-Off channel or server. Let α denote the rate at which the channel goes from the Off to the On state, and β the rate for going from On to Off. Thus, the invariant or steady-state probability of being in the On state is $\alpha/(\alpha + \beta)$. The service rate in the On state is denoted μ; it is 0 in the Off state. Note that the channel necessarily has to be in the On state at a service completion time. Hence, a customer that is at the head of the queue when a service completes starts its service immediately thereafter. The service time of this customer can be worked out easily. The minimum of the times to service completion or the channel going Off has an $\text{Exp}(\mu + \beta)$ distribution. With probability $\mu/(\mu + \beta)$, this time corresponds to a service completion. With the residual probability, the customer has to wait a further $\text{Exp}(\alpha)$ time for the channel to return to the On state, and resume service. On resumption, the customer needs a further $\text{Exp}(\mu)$ service time, due to the memoryless property of the exponential distribution. This description lends itself to an easy calculation of the LST of the service time distribution, which turns out to be

$$g(s) = \frac{\frac{\mu}{\mu+\beta}\frac{\mu}{\mu+s}}{1 - \frac{\beta}{\mu+\beta+s}\frac{\alpha}{\alpha+s}}. \tag{5.17}$$

Detailed derivations are left to the appendix in A.3.

Now, the above description applies to all customers served during a busy cycle except the one that initiated it. That customer, which entered an empty

queue, had a non-zero probability of arriving when the channel was Off, and having to wait to begin service. Consequently, its service time LST is given by

$$\begin{aligned} g_0(s) &= (1-q)g(s) + \frac{\alpha q}{\alpha + s} g(s), \\ \text{where } \; q &= \frac{\beta}{\alpha + \beta + \lambda}. \end{aligned} \tag{5.18}$$

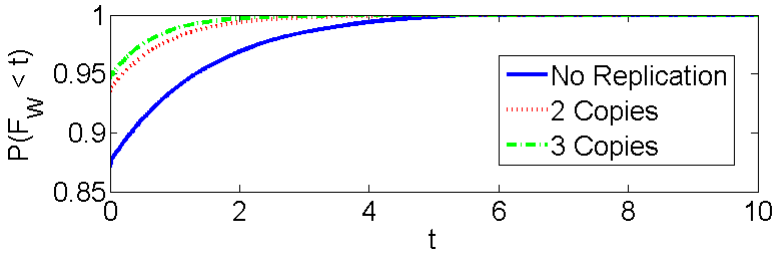
Here, q is the probability that the first customer to arrive after the queue becomes empty finds the channel in the Off state.

This model is known as the $M/G/1$ queue with Exceptional First Service, and an analogue of the P-K formula is given in [110] for the LST of the waiting time distribution:

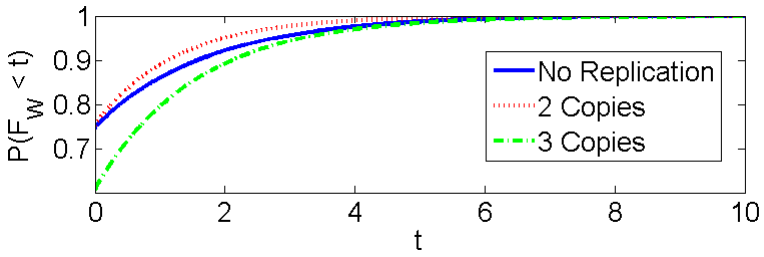
$$\begin{aligned} W^*(s) &= A \frac{s - \lambda g_0(s) + \lambda g(s)}{s - \lambda + \lambda g(s)}, \\ \text{where } \; A &= \frac{\alpha(\mu - \lambda)(\mu + \beta) - \lambda\beta(\mu + \alpha + \beta)}{\mu(\mu + \beta)(\alpha + q\lambda)}, \end{aligned} \tag{5.19}$$

for q given in (5.18).

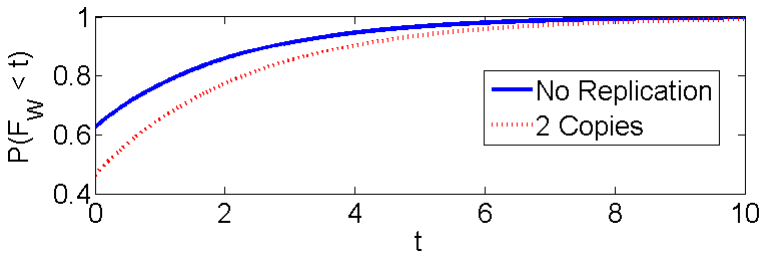
Substituting for g and g_0 from (5.17) and (5.18) in (5.19), an explicit, albeit complicated, expression for the LST of the waiting time is found. The expression can be inverted (with numerical root finding if necessary) to get an explicit expression for the waiting time distribution. The inverted expression is not displayed here as it is rather complicated, so it is left to the appendix in Section A.3.3. This can be used in turn to obtain the latency after replication, using the same procedure as for exponential and phase-type service times. Also, similarly to the phase-type case, the inversion leads to an expression that is a sum of exponential terms, therefore a dominant exponential term can be found. The resulting CDFs for the latency and dominant exponential approximation are plotted in Figure 5.3.3. The parameters chosen are $\alpha = 0.9$, $\beta = 0.1$ and $\mu = 1$. This corresponds to a channel that is available 90% of the time, but the Off period is ten times as long as the service time, on average. Thus, it exhibits considerable variability in the service time, like the hyperexponential model considered earlier.



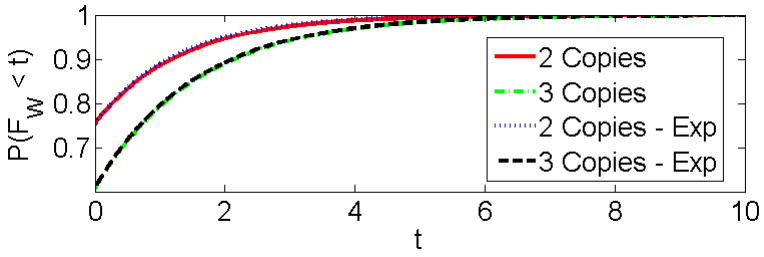
(a) $\lambda = 0.1$



(b) $\lambda = 0.2$



(c) $\lambda = 0.3$



(d) Dominant exp. approx. - $\lambda_k = 0.2$

Figure 5.3: CDF of the latency: On-Off server

5.3.4 Comparing $M/M/1$, $M/PH/1$ and On-Off Models

The previous literature largely investigates exponential service time distributions i.e. the $M/M/1$ queue. The extension to phase type and on-off

channel/server service distributions increases the variability, and thus the realism of the models is improved. This section aims to show the extent to which the novel methods we present differ in terms of the wait time distribution. To show the variability between the methods we have chosen the parameters so that the expected service time, $\mathbb{E}_s$, is very nearly the same in all the models, and equal to 1. An arrival rate $\lambda = 0.2$ has been chosen as this was the most interesting case from the previous sections. A full list of the parameters used to create Figure 5.4 is detailed in Table 5.1.

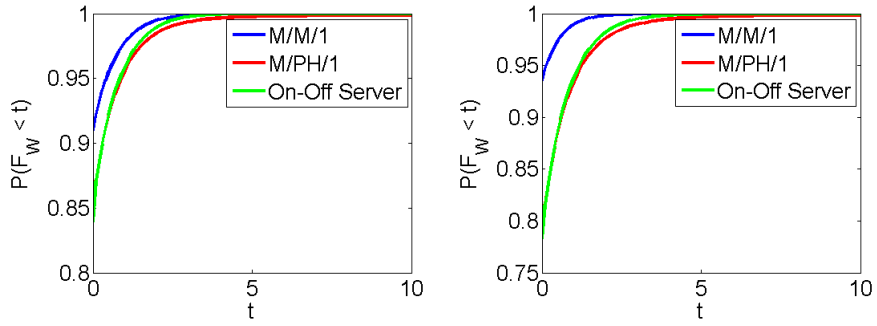
Table 5.1: Comparison parameters

Parameter	M/M/1	M/PH/1	On-Off
λ	0.2	0.2	0.2
μ	1	$\mu_1 = 1.1$, $\mu_2 = 0.0109$	1.001001
$\mathbb{E}_s$	1	$\approx$1	$\approx$1
p / α	N/A	$p = 0.999$	$\alpha = 0.999$

To describe the parameters for the $M/PH/1$ and On-Off server queues in Table 5.1 consider the methods of service as having two different phases of service. If we describe the periods where the server runs the majority of the time as the 'normal' service , and the rest of the time as 'abnormal' service then in the example here both the $M/PH/1$ and On-Off server queues run under normal service 99.9% of the time. This is quantified by the p or α parameter in Table 5.1; we keep the distinction between p and α as p refers to a probability in the $M/PH/1$ queue, whereas α refers to a rate in the On-Off server queue. As we fix the p or α parameter, the service rate parameters (μ in Table 5.1) must be increased to ensure the expected service time approximates 1. In other words, as the server is either not running at full capacity when it is in 'abnormal' service for the $M/PH/1$, or is not performing any service when the server is Off, then when the server is in 'normal' service, it must have an increased service rate to ensure the expected service time remains at 1.

When inspecting Figure 5.4 the results show how the increased variability of the $M/PH/1$ and On-Off server methods affects the wait time distribution when compared to the $M/M/1$ queue. In particular there is a significant decrease in the probability of having a smaller wait time for both the $M/PH/1$

and On-Off queues when compared to the $M/M/1$. This is an unsurprising result, and shows the importance of having realistic communication channel assumptions. Specifically, if the channel was assumed to behave like an $M/M/1$ queue the benefits of replication would be considerably overestimated.



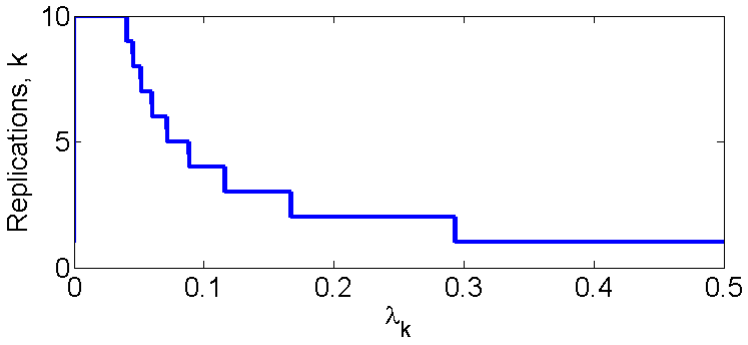
(a) 2 Replicates (b) 3 Replicates

Figure 5.4: Comparison between M/M/1 , M/PH/1 and On-Off server

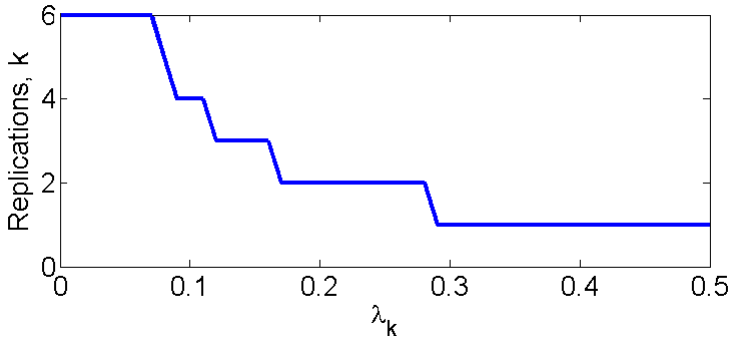
5.3.5 Optimal Replication Factor

Having developed models to represent the network, an important factor to consider is by how much to replicate, since replicating by too much can cause excessive queuing caused by congestion, but not replicating enough will under utilise the available capacity. A simple way to look at the optimal replication factor is to look at the mean wait time whilst varying the load and number of replications. The optimal replication factor will then be the number of replicates that give the minimum mean wait time for a given load. This is the approach taken to produce Figures 5.5 and 5.6. Figure 5.5 shows a clear pattern, in that for a light load the number of replicates should be high, and for a heavy load the number of replicates should be reduced. To quantify the benefit of replication Figure 5.6 shows the optimal expected wait time, with the expected wait time for the unreplicated case also plotted for comparison.

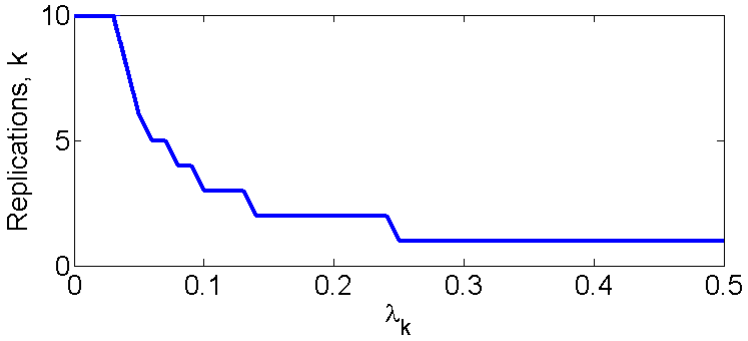
A further point to consider is the limit to which replication reduces the expected wait time. Intuitively, redundancy should improve the wait time up



(a) Exponential



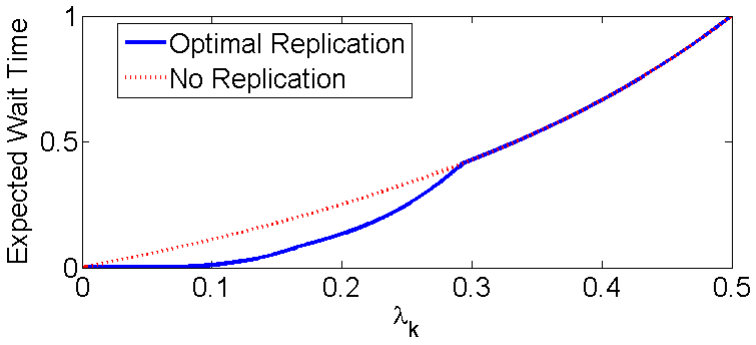
(b) Hyper-exponential



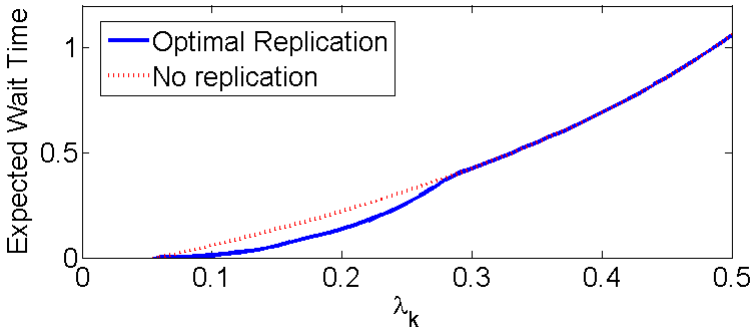
(c) Exceptional First Service

Figure 5.5: Optimal Number of Replications

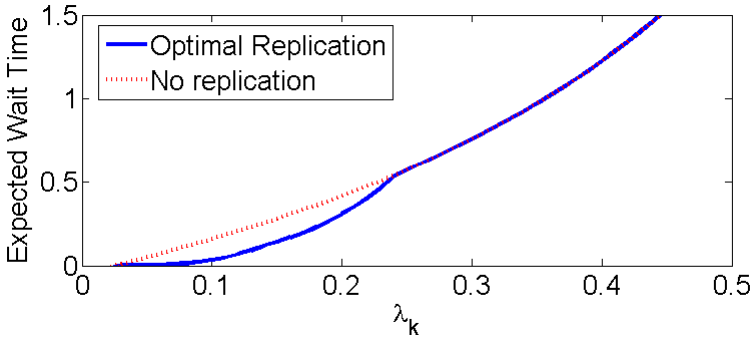
to a load value of 0.5 since at this point replicating by the minimum amount, 2 copies, would cause the load to become 1. However, there is variance to the service time, hence the maximum load to which replication reduces the expected wait time is less than 0.5. A further point to consider is that the



(a) Exponential



(b) Hyper-exponential



(c) Exceptional First Service

Figure 5.6: Optimal Mean vs. No Redundancy

more variable cases of the hyper-exponential and exceptional first service models reduce the limit at which redundancy is useful.

In practice, the external arrival rate may not be known and so a simple dynamic redundancy policy is proposed. This is a distinct advantage of using

SDN as an enabler, as a dynamic redundancy policy may be programmed at a controller, and the rules can be forwarded to the networking elements. In the method used here, an 'instantaneous arrival rate' is calculated to make the decision of by how much to replicate. A simple way to do this is to calculate the interarrival time between the current and previous arrivals, taking the inverse of this then gives the IAR (instantaneous arrival rate) i.e. let the time at which the t^{th} arrival enters the system be denoted a_t then:

$$\text{IAR} = \frac{1}{a_t - a_{t-1}} \tag{5.20}$$

Once the IAR is calculated, it is passed to a decision process that matches the IAR to a level of redundancy taken from Figure 5.5. In the simulation a cap of three replicates was used to represent a cap in the resources available. For the simulation, the arrivals are taken from a Poisson Process with the arrival rate set at 0.2 (0.2 was chosen as it is a more interesting case as seen in figure 5.1), and the service is exponential with rate 1.

The simulation results are shown in Figure 5.7 and agree with the theory in Section 5.3.1. The CDF shows that the simple dynamic policy performs better in this instance than all the other policies. Furthermore, the extreme valued times are mainly seen by the no-redundancy case. Finally, it is important to note there is a consequence of replicating by too much. To see this, observe that there is a crossover between the double and triple redundancy in the CDF, this indicates the double redundancy outperforms the triple redundancy for larger values of t.

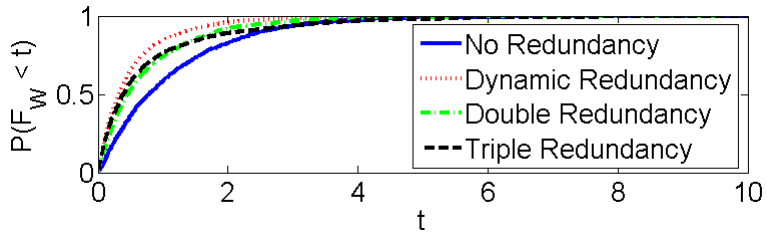


Figure 5.7: CDF Comparison between the dynamic policy and the theoretical values

5.3.6 Tail Bounds on Latency

In some applications, the primary goal is not necessarily to reduce mean latency, but to guarantee packet delivery within a specified bound on the delay. Safety applications and virtual reality are two example situations which require delay guarantees to function correctly. In this work, probabilistic guarantees are considered, i.e., the objective is to ensure that some high proportion of packets, e.g., 99% or 99.9% ($\epsilon = 0.01$ and $\epsilon = 0.001$ in the analysis), suffer delay no greater than a specified bound.

We have found explicit equations for the CDF of the latency with replication for the $M/M/1$, $M/PH/1$ and Channel/Server variability models in this chapter. While the exact expressions are rather complicated, the tail bounds are much simpler; the dominant exponential term, which was proposed as an approximation for the mean latency, becomes exact in the tail. This is made precise below, and used to quantify the benefit of replication in providing probabilistic delay guarantees.

Let W_λ denote a random variable with the stationary waiting distribution, whose dependence on λ, the arrival rate into the queue, has been made explicit in the notation. Define

$$\eta_\lambda = -\limsup_{\tau\to\infty} \frac{1}{\tau} \log \mathbb{P}(W_\lambda > \tau). \tag{5.21}$$

In fact, the limit exists above in all the models we consider. Letting L_k denote latency with k-fold replication, and noting that this increases the arrival rate to $k\lambda$, we have

$$\mathbb{P}(L_k > t) = \mathbb{P}(W_{k\lambda} > t)^k,$$

by the assumption that delays in the k parallel queues over which the packet is replicated are iid. Hence, it follows from (5.21) that

$$\limsup_{\tau\to\infty} \frac{1}{\tau} \log \mathbb{P}(L_k > \tau) \leq -k\eta_{k\lambda}. \tag{5.22}$$

Next, we define τ_ϵ to be the threshold value of delay for which we can provide a probabilistic guarantee that the latency is bounded by τ_ϵ with probability

at least $1 - \epsilon$; more precisely,

$$\tau_\epsilon = \inf\{t > 0 : \mathbb{P}(L_k > t) < \epsilon\}. \tag{5.23}$$

This definition, together with eq. (5.22), motivates the following approximation for τ_ϵ:

$$\tau_\epsilon \approx \frac{-\log \epsilon}{k\eta_{k\lambda}}. \tag{5.24}$$

We evaluate this quantity numerically for different values of k, for each of the queueing models we studied earlier in this chapter.

The threshold τ_ϵ can be calculated exactly for the $M/M/1$ queue, using the exact expression for the latency CDF given in (5.8). Straightforward calculations yield that, for k-fold replication,

$$\tau_\epsilon = \frac{1}{k(\mu - k\lambda)} \log \frac{1}{\epsilon} - \frac{\log \frac{\mu}{k\lambda}}{\mu - k\lambda}.$$

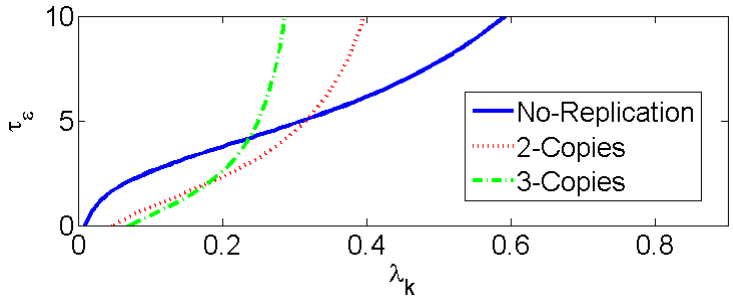
Observe that the approximation in (5.24) coincides with the first term in the RHS above, but does not capture the second term. For the $M/PH/1$ queue, exact calculations are still possible, though messy. But the approximation in (5.24) is easy to calculate. First, observe from (5.13) and (5.21) that η_λ, the tail decay exponent of the waiting time distribution in the $M/PH/1$ queue, is equal to $\eta(\lambda)$, which was defined in (5.12). Next, using the approximation for the latency CDF given in (5.14), we obtain the approximation

$$\tau_\epsilon = \frac{-\log \epsilon}{k\eta(k\lambda)} + \frac{\log(k\rho)}{\eta(k\lambda)};$$

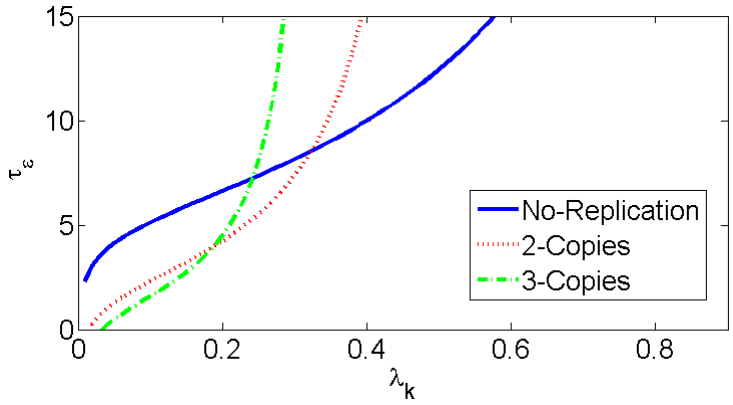
the approximate expression for τ_ϵ in (5.24) matches the first term in the RHS above while failing to capture the second, correction term. The above expression can be evaluated numerically for hyper-exponential service times. Finally, an approximation for τ_ϵ in the On-Off server model can also be obtained numerically from the approximations for the latency CDF obtained earlier.

The results for $\epsilon = 0.01$ and 0.001 are shown in Figure 5.8 for the expo-

nential case and Tables 5.2 & 5.3 for the hyper-exponential and On-Off cases. Note that N/A is displayed in the tables when the traffic intensity exceeds 1. An important aspect of these results is that there are crossover points, where a higher level of redundancy does not necessarily provide better reliability in terms of delay (the best policy is highlighted in bold).



(a) $\epsilon = 0.01$



(b) $\epsilon = 0.001$

Figure 5.8: The delay bound τ_ϵ, with varied λ_k for the exponential service model

Table 5.2: Hyper-exponential, τ_ϵ

$\tau_\epsilon, \epsilon = 0.01$	No Replication	2 Copies	3 Copies
$\lambda = 0.1$	2.6100	0.8880	**0.4920**
$\lambda = 0.2$	3.9370	**2.3930**	2.6790
$\lambda = 0.3$	5.2670	**4.7490**	N/A
$\lambda = 0.4$	**6.9010**	11.6600	N/A
$\lambda = 0.5$	**9.1470**	N/A	N/A
$\tau_\epsilon, \epsilon = 0.001$	No Replication	2 Copies	3 Copies
$\lambda = 0.1$	5.4440	2.3720	**1.6140**
$\lambda = 0.2$	8.6230	**4.4810**	4.7490
$\lambda = 0.3$	13.5750	**8.1490**	N/A
$\lambda = 0.4$	**19.0150**	19.2210	N/A
$\lambda = 0.5$	**24.4600**	N/A	N/A

Table 5.3: Exceptional First Service, τ_ϵ

$\tau_\epsilon, \epsilon = 0.01$	No Replication	2 Copies	3 Copies
$\lambda = 0.1$	3.4700	1.5900	**1.1300**
$\lambda = 0.2$	5.3400	**4.0600**	5.7200
$\lambda = 0.3$	**7.2400**	9.3200	N/A
$\lambda = 0.4$	**9.6800**	>30	N/A
$\lambda = 0.5$	**13.3400**	N/A	N/A
$\tau_\epsilon, \epsilon = 0.001$	No Replication	2 Copies	3 Copies
$\lambda = 0.1$	5.1400	3.5600	**2.7200**
$\lambda = 0.2$	7.5000	**6.9500**	9.3200
$\lambda = 0.3$	**10.1000**	14.6800	N/A
$\lambda = 0.4$	**13.6100**	>30	N/A
$\lambda = 0.5$	**19.0500**	N/A	N/A

5.4 Conclusion and Further work

In this chapter, the use of SDN as a technology for replicating packets and routing the replicas across multiple disjoint paths was proposed. A very simple queueing theoretic model was proposed to abstract the impact of such replication in a communication network. This model was used to study the effect of packet replication on the latency, namely the time to receive any one of the replicas. Delay distributions were calculated using the P-K formula and the LST for different service models. It was found that the expressions for the distributions can be very complex, but that a simple approximation using a dominant exponential term led to satisfying results. The analysis

of the distributions indicated that an optimal policy is dependent upon the network load. Simulations then showed that a simple dynamic replication policy can be used to further reduce the latency in a system. Furthermore, the analysis highlighted that the benefits are more pronounced in the tail of the distribution. This observation led to the study of the tail exponent of the latency distribution, and its use to provide probabilistic guarantees that the latency does not exceed a specified value. Such guarantees can be valuable in latency sensitive applications. Again, the optimal amount of replication was found to depend upon the network load.

One of the main limitations of the work carried out in this chapter is that the mathematical abstraction is overly simplistic. The assumption of independence among the parallel queues was justified on the grounds that they would see independent cross-traffic, and that the impact of the index flow would be negligible. While some portion of the cross-traffic would undoubtedly be independent, it is likely that some portion would also be common, as nearby sources and destinations are likely to use paths that overlap significantly. Similar, while the impact of the index flow, which is common to all the paths, might be small, it is not completely negligible. A more realistic model that accounts for a fraction of the traffic being common could be a topic for further research. There has, in fact, been little rigorous mathematical work on the problem studied in this chapter. One significant recent contribution is the work of Gardner *et al.* [111].

Secondly, the analysis has been restricted to phase-type service distributions. While this is indeed a rich class of distributions, it excludes heavy-tailed distributions such as the Pareto, whose tails decay polynomially rather than exponentially. Extending the analysis to such distributions would be an interesting direction for future work.

It was assumed in this chapter that, when replication is employed, all network traffic is replicated to the same extent. This assumption leads to significantly understating the benefits of replication. As the motivation behind this chapter was to argue that replication has benefits, this was the most conservative assumption to make. But in practice, it is likely that only a small fraction of high-priority and delay sensitive traffic would be

replicated. It would not be difficult to extend the analysis in this chapter assuming that replication is only applied to a fixed fraction of the network traffic.

Chapter 6

Redundancy: Coding

6.1 Introduction and Related Work

The motivation for the work in this chapter is the same as in the previous one: to leverage some of the functionality provided by SDN in order to achieve lower latencies for high-criticality traffic. The mechanism employed is again the use of redundant messages and multipath routing. But instead of simply replicating the data, redundancy is added in the form of linear combinations of the data, an idea with a long history in error-correction coding. These redundant bits can be used to recover the message despite some number of errors or erasures. Coding has traditionally been used to improve the reliability of communication, but here it is proposed as a mechanism to reduce latency.

The notation (n, k) code refers to a code in which k message symbols are used to generate n coded symbols; typically, $n > k$, i.e., coding adds redundancy to the message, which can be used to reconstruct it despite a certain number of errors or erasures. An (n, k) code can be used to code a single packet by first splitting it into k pieces, and then adding $n - k$ redundant pieces, giving a total of n pieces to be sent across the network. A simple example is shown in figure 6.1, which depicts a $(3, 2)$ code. A packet is first split into $k = 2$ pieces, which are then used to generate a third coded piece; here, $+$ is to be interpreted as bitwise addition modulo 2,

or equivalently the XOR operation. The figure shows how the message can be recovered from any two of the three pieces. Thus, the loss or erasure of one piece does not compromise the message (though the loss of two pieces or an error in one piece would). As the two pieces are each half the size of the original packet, the overall impact is to increase the amount of data transmitted by 50%. More generally, an (n, k) code increases the load by a factor of n/k.

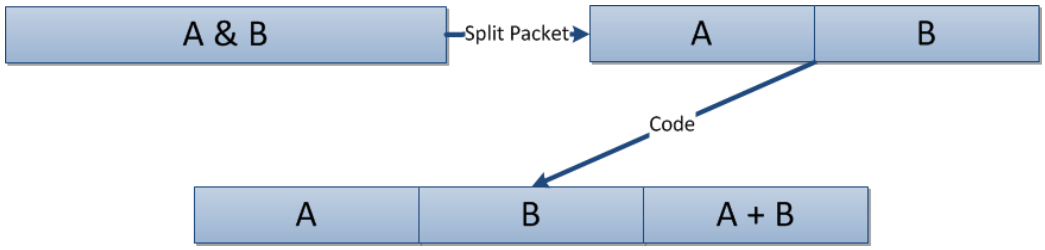


Figure 6.1: A binary $(3, 2)$ code

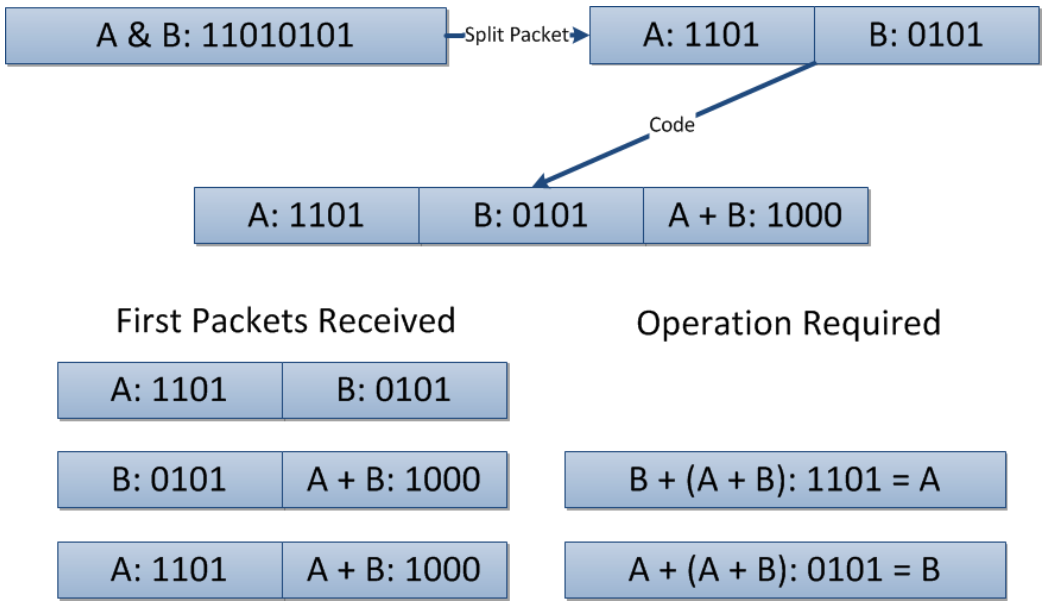


Figure 6.2: Example of a $(3, 2)$ code

Coding has both advantages and disadvantages when compared to replication. One of the main advantages is the ability to increase the load fractionally, by a factor of $\frac{n}{k}$, whereas replication is limited to only increasing the

load by an integer value. This is particularly beneficial when considering the optimal rate as discussed in section 6.3.5, as a fractional rate can be achieved, leading to a smoother transition between optimal levels when compared to replication. On the other hand, latency is potentially larger as we need to wait for the first k out of n pieces, rather than for a single packet (though note that the pieces are smaller than a packet). The other disadvantage is that the encoding and decoding operations add an overhead that contributes additional latency.

The programmability of SDN equips it to support both multipath routing and network coding. These capabilities have been exploited in prior work; see [112, 113]. Coding inside the network is not considered here, but only coding by the source. Network coding in a high-bandwidth network would impose substantial demands, and it may also be undesirable for reasons of security and privacy to allow network elements to modify data packets. In any case, it offers no benefits over source coding for unicast or point-to-point communication. The main contribution of this chapter is an analysis of the potential benefits of implementing redundancy via coding, when applied to latency reduction. Some preliminary steps towards such an analysis using SDN as an enabling technology were taken in [114], and a SDN with the capability of network coding is presented. Our work exploits the fact that SDN can be used as an enabling technology for network coding as presented in [114] and extends the analytical work.

6.2 Model and Problem Formulation

The model is virtually identical to the one in the previous chapter, and so is only recapitulated briefly. We study the latency of a single index flow, which has n vertex-disjoint routes available to it. Packets of the flow are split into k pieces, coded using an (n, k) erasure code, and the n coded pieces are transmitted along n disjoint paths. It is assumed that the packet can be reconstructed from any k pieces. Thus, ignoring overheads due to encoding and decoding, the latency is the time until the k^{th} piece is received. It is assumed that the total delay along each route is well approximated

by the service time in an $M/G/1$ queue. By further assuming that the delay is largely due to cross-traffic, and that the cross-traffic along distinct routes comes from independent flows, the $M/G/1$ queues may be treated as independent. This is justified by the assumption that each route carries traffic from a large number of flows, and that the contribution of any single one of these flows to the total traffic is negligible. Thus, even though the indexed traffic is common to the routes along which it is replicated, this introduces no dependence because its own contribution to the total traffic along any of these routes is negligible; the waiting times it incurs are almost entirely due to the service times of cross-traffic. This is also the argument made in [83] where it is assumed the number of servers is large, and thus the contribution to the total load of a single replicated packet is small. This is also quantified empirically in the paper by showing how applications with small packet sizes, but diverse paths benefit from redundancy such as DNS requests. Finally, it is assumed that there is no route purging, i.e, data that is in transit cannot be cancelled even if it is no longer relevant. In order to fairly account for the increase in traffic due to the added redundancy, it is assumed that all traffic in the network uses the same coding scheme, and hence that the network load is increased by the fixed factor n/k. In practice, only a small fraction of high-priority traffic may have redundancy added; moreover, there may be multiple priority classes with different amounts of added redundancy. While it is not difficult in principle to extend the analysis to such settings, the simplest assumptions lead to the most transparent and interpretable results.

Given the above assumptions, a very simple system model of n parallel queues is obtained. Packets of the index flow arrive into the system according to a Poisson process. Each packet is split into k pieces, which are used to generate n coded packets, transmitted across n parallel queues. Next, each of the queues is itself modelled as an $M/G/1$ queue; in other words, the j^{th} queue sees Poisson arrivals at rate λ_j, the customers (packets) have independent and identically distributed (iid) service times denoted $S_i^{(j)}, i \in \mathbb{Z}$ with a general distribution, and they are served in their order of arrival. While the aggregate arrival rate λ_j into the j^{th} queue will depend on the

parameters k and n, the dependence is not made explicit in the notation. Let $\rho_j = \lambda_j \mathbb{E}[S_1^{(j)}]$ denote the load in the j^{th} queue; it is assumed that $\rho_j < 1$ for all j, i.e., that all queues are stable. Under this assumption, each queue has an invariant distribution under which waiting times are almost surely finite. The LST of the waiting time distribution is given by the Pollaczek-Khinchine (P-K) formula:

$$(W^j) * (s) = \frac{(1-\rho_j)s}{s - \lambda(1 - g_j(s))}, \tag{6.1}$$

where $g_j(s) = \mathbb{E}[\exp(-sS_1^{(j)})]$ is the LST of the service distribution in the j^{th} queue. Let W^j denote a random variable with the invariant distribution of the waiting time in the j^{th} queue. Assuming $W^1, W^2, \ldots, W^n$ are independent, denote the corresponding order statistics by $W^{(1)} \leq W^{(2)} \leq \ldots \leq W^{(n)}$. Then, the latency in steady state, denoted $L_{n,k}$, has the same distribution as the k^{th} order statistic, i.e., we can define

$$L_{n,k} = W^{(k)}. \tag{6.2}$$

This random variable, and its distribution, are the primary object of study in this chapter. In general, expressions for the distribution of this quantity are hard to derive, and very complicated. In order to obtain tractable results that shed light on the relation between the coding parameters and the resulting latency, we assume that the queues are symmetric, i.e., that they all have the same arrival rates and service time parameters. With this assumption, the W^j are iid, and the distribution of the latency is given by

$$\mathbb{P}(L_{n,k} < x) = \sum_{i=k}^{n} \binom{n}{i} F_{W^{(j)}}(x)^i (1 - F_{W^{(j)}}(x))^{n-i}. \tag{6.3}$$

In the remainder of this chaper, we evaluate this expression for all the queueing models studied in the previous chapter, and compare the benefits of coding to those of replication.

6.3 Coding and Mean Latency

The impact of coding on latency is studied in this section, using three different queueing models, with different service distributions. It is assumed throughout this section that packets of the index flow arrive according to a Poisson process of rate λ, and are coded across n routes using an (n, k)-code. A known result using the M/M/1 queue is first presented in 6.3.1. This is then compared to the novel models we propose in sections 6.3.2 and 6.3.3 that extend the service distribution to phase type and exceptional first service. A comparison between all of these methods is also presented in 6.3.4. The analysis from sections 5.3.5 and 5.3.6 is also repeated which provides novel results for the optimal coding rate and probabilistic delay bounds with respect to the coding models.

6.3.1 The $M/M/1$ queue

The simplest model investigates service times that are assumed to be iid with an Exp(μ) distribution. The justification for this is stated in chapter 5, but in short the assumption is that multiplexing a large number of independent flows, each of which individually contributes a small fraction of the total traffic, can lead to a service distribution with an exponentially decaying tail. The service rates μ_i would normally be different, but for notational convenience, and to facilitate graphical display of the results, let $\mu_i = \mu$ for each queue. This is also the convention taken in chapter 5, so a direct comparison can be made for each method by maintaining this choice of service parameter.

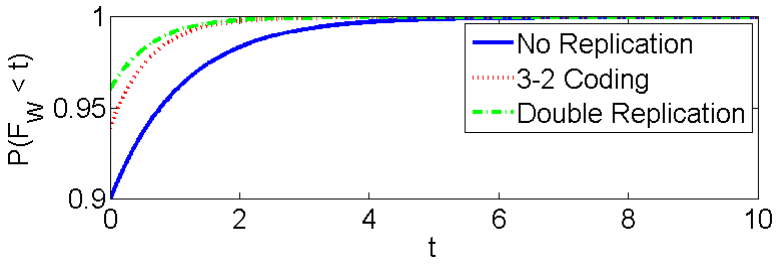
The CDF for the wait time distribution for the $M/M/1$ was found in 5 and is

$$F_W(t) = 1 - \rho e^{-(\mu-\lambda)t}. \tag{6.4}$$

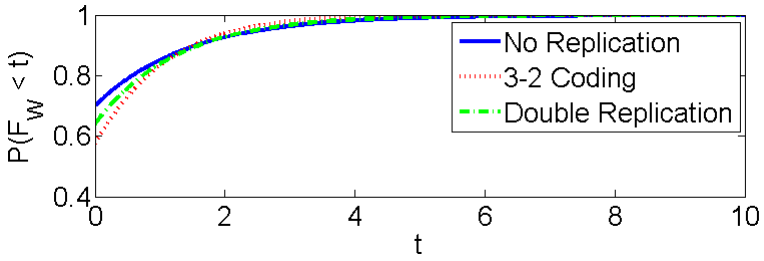
Now consider the effect of replicating every packet across n edge-disjoint routes. This will increase the total arrival rate into each queue by the factor

$r = \frac{n}{k}$. Substituting (6.4) in (6.3), it is found that

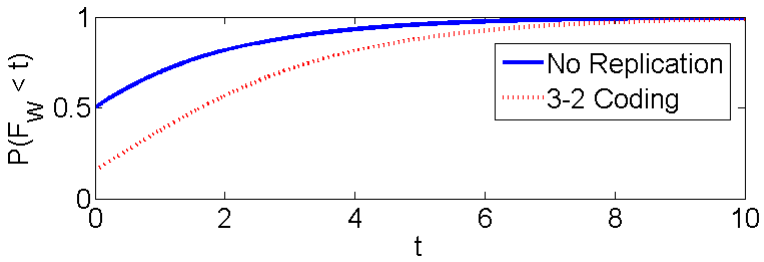
$$F_{L_{n,k}}(t) = \sum_{i=k}^{n} \binom{n}{i} \left(1 - r\rho e^{-(\mu - r\lambda)t}\right)^i \left(r\rho e^{-(\mu - r\lambda)t}\right)^{n-i} \tag{6.5}$$



(a) $\lambda = 0.1$



(b) $\lambda = 0.3$



(c) $\lambda = 0.5$

Figure 6.3: Latency CDF with exponential service under varied load: *scale changes on vertical axis*

The resulting CDFs with no replication, $(3, 2)$ coding and double replication are plotted in Figure 6.3, with various values of λ; $\lambda = 0.1$, 0.3 and 0.5. The CDF plots still cross each other, meaning that no single value of the replication factor or coding rate, r, may be optimal for all t. In particular, when $\lambda = 0.3$, the no replication case performs the best at small t, but then

there are crossover points where both the double replication and $(3, 2)$-coding policies improve the probability of the latency being smaller than the specified value. Another simple observation is that at the lowest load presented, $\lambda = 0.1$, the CDF plots for $(3, 2)$ coding and double replication are above those for no replication. This demonstrates the same finding as in Chapter 5, that replication, and in this case coding, significantly increase the probability of meeting a specified delay bound t at a low load.

6.3.2 The $M/PH/1$ queue

Phase-type distributions can be used to offer a much more realistic service distribution. This is because unlike the exponential case, where the service rate is specified by a single parameter μ, a phase-type distribution is made up of a number of parameters which can be varied to create a much more accurate representation of service. In particular, phase-type distributions can offer more variability, and this variability can represent phenomena within networking. An example would be a fault in the network causing retransmission, or re-routing, this action may happen with a small probability, but would result in a longer delay. In a phase-type distribution this could be represented by giving one phase a larger average delay with a small starting probability (the fault), and another phase a smaller average delay with the remaining probability (normal service).

The analysis from 5.3.2 is repeated to give the wait-time distribution when a phase-type service distribution is employed. Ultimately, the waiting time density can be written as:

$$f_W(t) = c_0\delta(t) + \sum_{i=1}^{m} c_i e^{z_i t},\ t \geq 0.$$

As this shows the waiting time is a sum of exponential distributions with parameters $-z_i$ and constant multiplied by a delta function (which must have mass $1 - \rho$, as this is the probability of the queue being empty), then

again the distribution can be approximated by a single exponential.

$$f_W(t) \approx (1-\rho)\delta(t) + \rho\eta_\lambda e^{-\eta_\lambda t},$$

where,

$$\eta_\lambda = -\max_{i=1}^{m} \mathrm{Re}(z_i).$$

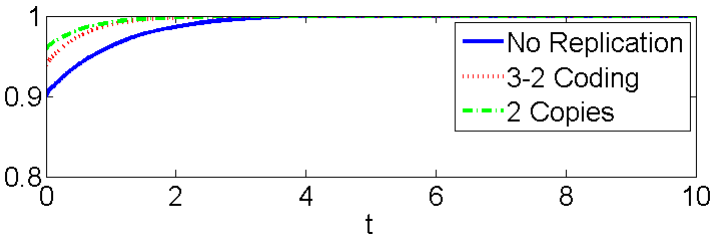
We have made the dependence of the roots, and hence of η, on the arrival rate λ explicit in the notation. Assume that $\eta_\lambda > 0$. Then η_λ is the slowest decaying exponential, which makes it the dominant term in the expression for the waiting time density. As the approximation has the same form as in the $M/M/1$ case, it makes it much easier to calculate the latency using the approximation, and the latency has CDF given by

$$F_{L_{n,k}}(t) = \sum_{i=k}^{n} \binom{n}{i} \left(1 - r\rho e^{-\eta_{\frac{k}{n}\lambda} t}\right)^i \left(r\rho e^{-\eta_{\frac{k}{n}\lambda} t}\right)^{n-i}$$

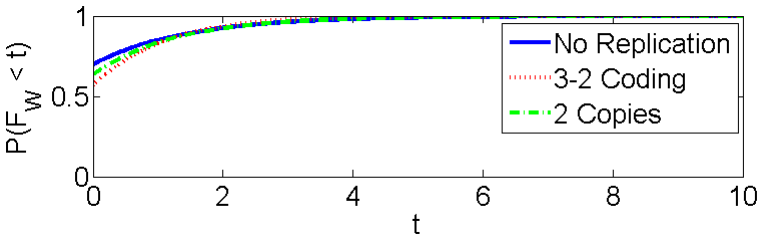
Example: Hyper-Exponential Service Times

The approach is now illustrated for the special case of a hyper-exponential service time distribution with both the LST inversion and the exponential approximation as in 5.3.2. To demonstrate the validity of the approximation Figure 6.4(d) shows the CDFs of the dominant exponential approximation and the exact density. The CDFs are very close in value.

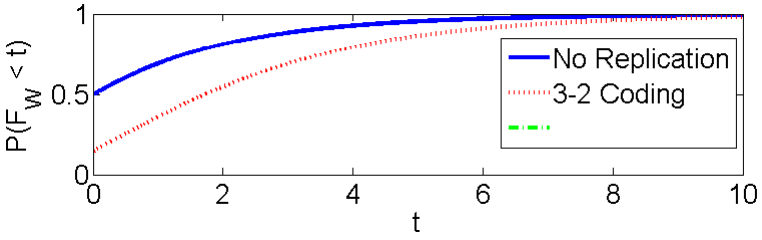
To calculate the latency when coding or replication is employed, the explicit waiting time density produced in section 5.3.2 can be used along with equation 6.3. This leads to a complicated expression, but the corresponding CDFs are plotted in Figure 6.4, with the parameters $\mu_1 = 1$, $\mu_2 = 0.1$, $p = 0.999$. This corresponds to a system where the service rate is 1 for 99.9% of the time, but only 0.1 the remainder of the time. This aims to model a fault, i.e. packet loss or fading, that has a latency that is 10x longer than the 'normal' amount. The results are similar to those in the exponential case, showing that: the benefits of coding are most apparent at low loads, crossover points exists for a moderate load, and there is a limit to the benefit of coding when the load is increased further.



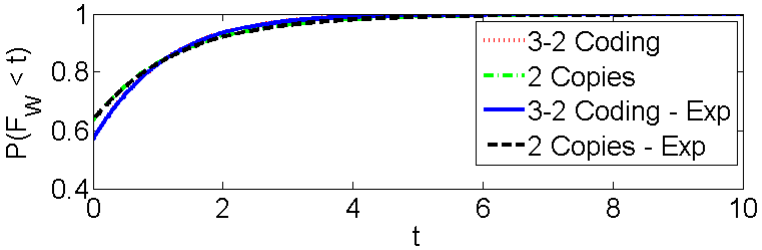
(a) $\lambda = 0.1$



(b) $\lambda = 0.3$



(c) $\lambda = 0.5$



(d) Dominant exp. approx. - $\lambda = 0.3$

Figure 6.4: CDF of the latency with hyper-exponential service time

6.3.3 Channel or server variability

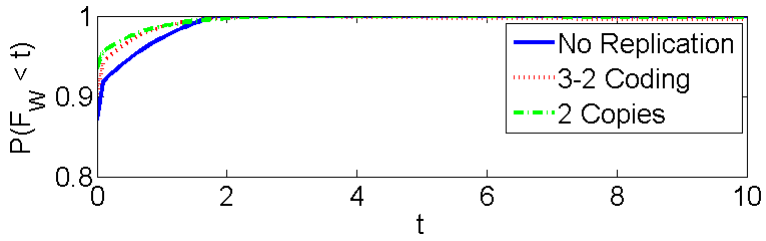
The server variability model is motivated by the fact that a server or networking element may malfunction, which would cause a build up of packets, resulting in an exceptional first service. Or in other words, once the server

returns back to an online state, the first service will be different to the service when the network is running as normal. The analysis for the latency distribution is a repeat of that in section 5.3.3. This can be used in turn to obtain the latency after coding, using the same procedure as for exponential and phase-type service times. Also, similarly to the phase-type case, the inversion leads to an expression that is a sum of exponential terms, therefore a dominant exponential term can be found. The resulting CDFs for the latency and dominant exponential approximation are plotted in Figure 6.5. The parameters chosen are $\alpha = 0.9$, $\beta = 0.1$ and $\mu = 1$. This corresponds to a channel that is available 90% of the time, but the Off period is ten times as long as the service time, on average. Thus, it exhibits considerable variability in the service time, like the hyper-exponential model considered earlier.

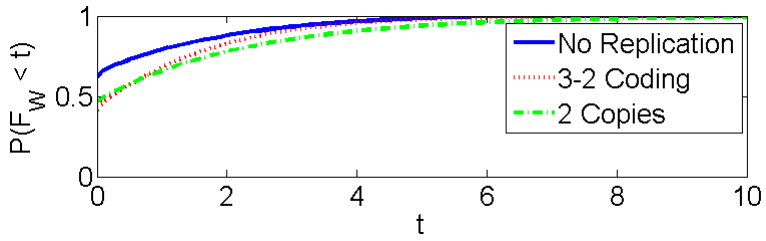
6.3.4 Comparison between $M/M/1$, $M/PH/1$ and On-Off Models

The previous chapter explained the extension from M/M/1 queues with replication to queues with phase type and on-off channel/server service distributions. This section aims to show the extent to which the coding method differs in terms of the latency distribution. To show the variability between the methods we have again chosen parameters such that the expected service time ($\mathbb{E}_s$) for each method approximates 1. In contrast to the previous chapter, an arrival rate of $\lambda = 0.2$ and $\lambda = 0.3$ are shown using a $(3, 2)$ coding scheme. A full list of the parameters used to create the Figure 6.6 is detailed in Table 6.1. See section 5.3.4 for an explanation of the parameters.

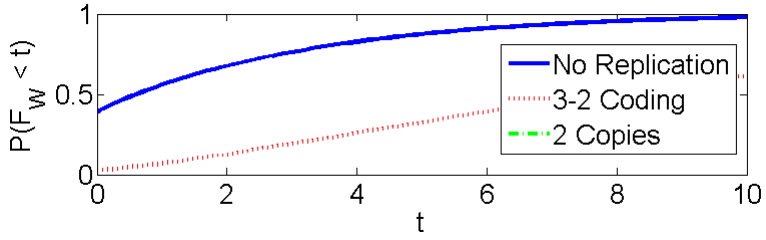
When inspecting Figure 6.6 the results show how the increased variability of the M/PH/1 and On-Off server methods effects the wait time distribution when compared to the M/M/1 queue. In particular there is a decrease in the probability of having a smaller wait time for both the M/PH/1 and On-Off queues when compared to the M/M/1. Although this is an unsurprising result, the difference between all the service distributions is less severe for the 3-2 coding method compared to using replication as presented in section 5.3.4. A possible cause for this would be the fact that 2 packets must be received



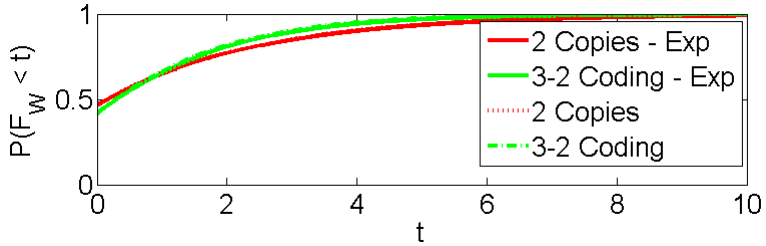
(a) $\lambda = 0.1$



(b) $\lambda = 0.3$



(c) $\lambda = 0.5$



(d) Dominant exp. approx. - $\lambda_k = 0.3$

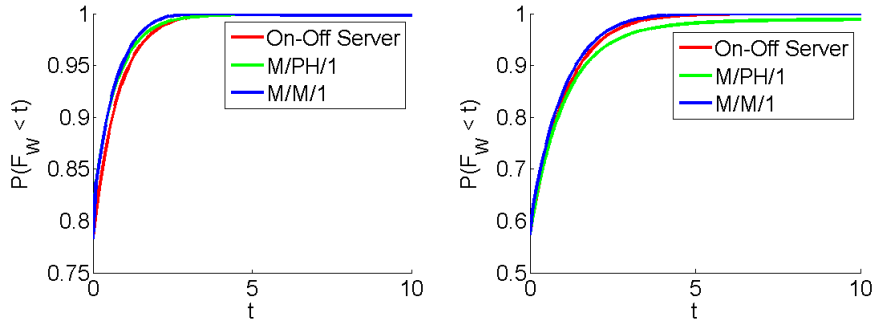
Figure 6.5: CDF of the latency with server variability

for the service to be completed, rather than just 1 as in the replication case. This may reduce the variability for each service method and subsequently between methods as there is a reduced probability of two packets taking an extreme (either very small or very large) amount of time to be served.

Similarly to the replication case the M/M/1 method approaches a probability of 1 first, meaning that a higher proportion of packets would arrive within a specified time under the M/M/1 assumption. This relates to the work in section 6.3.6, showing the probabilistic delay bounds for each of the methods.

Table 6.1: Comparison parameters

Parameter	M/M/1	M/PH/1	On-Off
λ	0.2 / 0.3	0.2 / 0.3	0.2 / 0.3
μ	1	$\mu_1 = 1.1$, $\mu_2 = 0.0109$	1.001
$\mathbb{E}_s$	1	$\approx$1	$\approx$1
p / α	N/A	$p = 0.999$	$\alpha = 0.999$



(a) $(3, 2)$ Coding, $\lambda = 0.2$ (b) $(3, 2)$ Coding, $\lambda = 0.3$

Figure 6.6: Comparison between M/M/1 , M/PH/1 and On-Off server

6.3.5 Optimal Coding Rate

The optimal replication factor was a beneficial addition to the analysis of redundant messages. The main reason for this is the information can be used to develop an optimal policy for replication, as for any particular load on the network there is a level of redundancy that gives the minimal delay. It was also explained that the programmability of SDN is an enabling technology for dynamic redundancy policies, and this extends to the possibility of a dynamic coding policy. For this to be possible the analysis is now further refined to

consider the network coding rate, rather than the amount of replication. The aim is that this will give a smoother curve for the optimal rate, rather than a stepped function as was the case with the optimal replication factor. The disadvantage is not only the increased complexity that a coding scheme would cause, but also the fact that k parts of the n messages sent must be received. In the analysis it is assumed any code rate from 5 to 1 is possible, but realistically the code rate is limited by the coding schemes available and therefore the analysis is only indicative of the achievable gain of coding.

By comparing Figures 6.7 it can be seen the optimal coding rate follows a trend much like the optimal replication factor. For a small load the rate is large and for a higher load the optimal rate is small. The main differing feature is that due to using coding, the coding rate $\frac{n}{k}$ can be a non-integer value, rather than having a fixed replication factor. The possibility of a non-integer rate means that there is more flexibility when choosing an optimal scheme. However, observe that the optimal rate for the minimum mean delay in each case is actually an integer value, with one exception in the exponential model. This indicates that the optimal policy in terms of reducing the delay is to actually use replication rather than coding, and more benefit when employing coding may be in terms of improving the reliability of the transmission.

Moreover, similarly to the replication case, there is a cut off point where it is no longer beneficial to employ coding or replication, this is indicated when the optimal rate is 1 at a load of approximately 0.3. This is also an observation in Figure 6.8 as the optimal mean converges and becomes equal to the uncoded case. The benefit of coding and replication can be quantified by the reduction in delay. The maximum benefit for the exponential service for example is seen at a load of 17.5% ($\lambda = 0.175$), where the delay is reduced from 0.2204 to 0.0675 i.e. a gain of 3.3x. The results for all the methods are shown in Table 6.2. An interesting point from the table is that the load of the biggest gain is similar between the methods, indicating that there is an optimal load for using coding. Logically it might seem sensible that the most benefit of coding is at the lowest load, but since the delay is small at low load then the overall benefit is reduced. In contrast, at a higher load the

optimal policy is not to use any coding, this is due to the increased delay caused by the extra load on the system.

Table 6.2: Coding Delay Gain

Service	Load at Biggest Gain	Gain	Limit of Gain
Exponential	$\lambda = 0.175$	3.3x	$\lambda = 0.296$
Hyper-Exp	$\lambda = 0.151$	3.1x	$\lambda = 0.294$
Vacation	$\lambda = 0.14$	1.17x	$\lambda = 0.19$

6.3.6 Probabilistic Delay Bounds

As in Chapter 5 the bounded delay i.e. the probability that a delay does not exceed a given bound, is an important factor to consider. This is because certain applications rely on a delay not exceeding a threshold for their functionality. This is different to the problem of complete packet loss which is an area that has been widely researched, and many protocols have been developed to combat this [115, 116]. An example would include VOIP or video conferencing, where if a delay threshold is exceeded the transmission would no longer be useful due to synchronisation problems. In order to analyse the delay bounds a similar analysis to that in section 5.3.6 is performed. As a quick reminder ϵ is the threshold value, so a value of $\epsilon = 0.01$ indicates that 99% of packets would be received within time, t, for a given value of λ. Similarly, $\epsilon = 0.001$ indicates 99.9% of packets would be received.

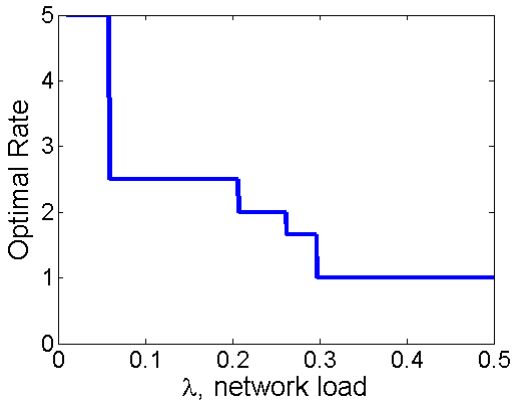
As the expressions are more complicated for coding than the replication case, the numerical equivalents are presented for each method in Figures 6.9 and 6.10. The graphs in Figure 6.9 show the distinction between choosing a value of $\epsilon = 0.01$ and $\epsilon = 0.001$. The curves show a similar pattern for each service distribution, in that the bound on t is increased for the smaller value of ϵ. This is simply a direct consequence of the fact that a higher proportion of packets will arrive within a larger time frame, t. What is more interesting is that there is a crossover point between the coding schemes, as there was with the CDFs for the wait time, this shows that although the double replication

and 3-2 coding schemes perform better initially, as the load increases it is more beneficial in terms of latency to send only one packet.

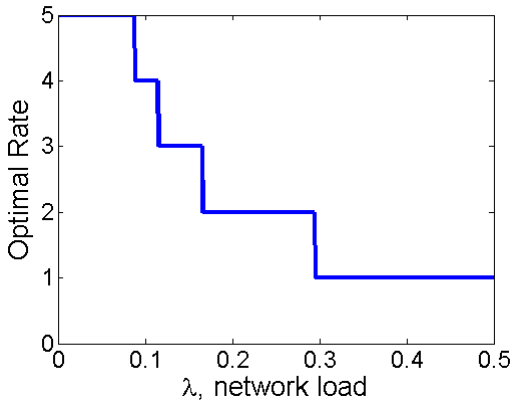
When there is more variability in the service distribution the arrival rate threshold reduces. This is most obvious in Figure 6.9 when comparing the hyper-exponential and exceptional first service models. In particular, although the crossover point is unclear for the exceptional first service model the crossover is clearly less than that for the hyper-exponential. To highlight the difference between the coding schemes Figure 6.10 shows a magnified portion of the graph for the exceptional first service model at a low load. The magnified version shows that the double replication scheme performs the best initially as the t value is the lowest up to a load of $\lambda \approx 0.15$, but then the 3-2 coding scheme performs better until $\lambda \approx 0.275$. If the other graphs were also magnified this pattern would remain the same, this emphasises the fact that the coding scheme directly effects the reliability of delay and is dependent upon the load.

A final point is that the difference between the coding schemes becomes much more evident at higher loads. It is hypothesised that this is a consequence of the total load on the network, or traffic intensity , approaching 100%. This is evident for the double replication scheme in the On-Off server case when $\lambda \approx 0.3$[1] as the value of t begins to rapidly approach the maximum limit of the graph i.e. $t = 30$. Similarly, this is why the $3-2$-coding scheme t-value does not increase more rapidly until $\lambda \approx 0.4$ as the total arrival rate is only multiplied by the coding rate, which is $\frac{3}{2}$, rather than 2. Finally, the no replication scheme does not increase rapidly at all on the graphs shown as the traffic intensity remains less than 100% for the maximum value of λ shown ($\lambda = 0.5$).

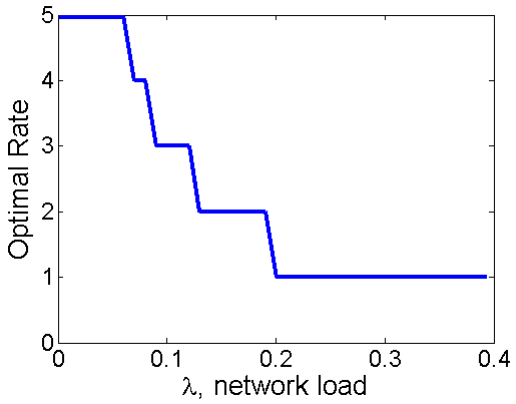
[1]The λ value is only 0.3 as the double replication increases the total arrival rate by 2, which means the effective arrival rate is ≈ 0.6.



(a) Exponential

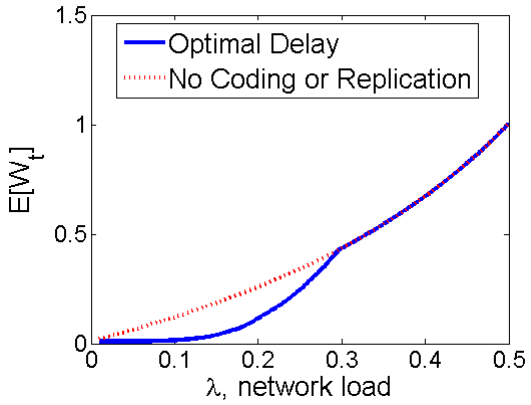


(b) Hyper-exponential

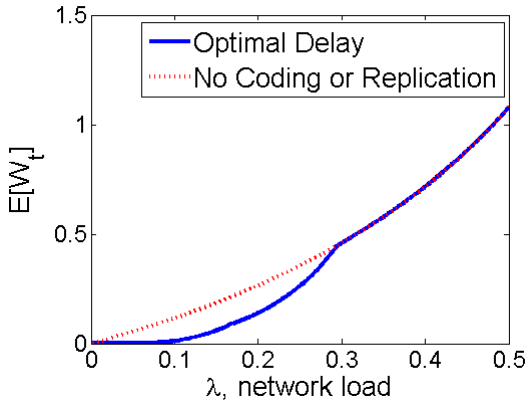


(c) Server variability

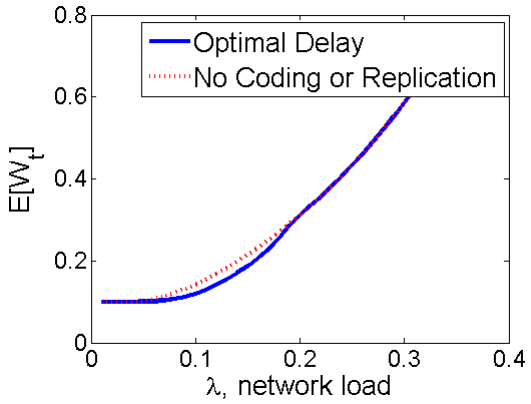
Figure 6.7: Optimal Coding Rate



(a) Exponential

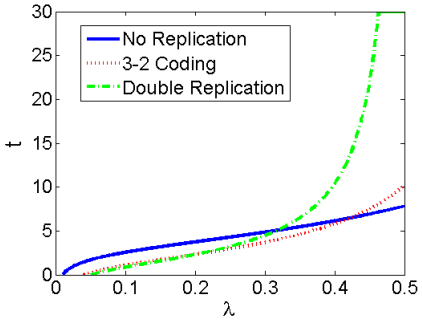


(b) Hyper-exponential

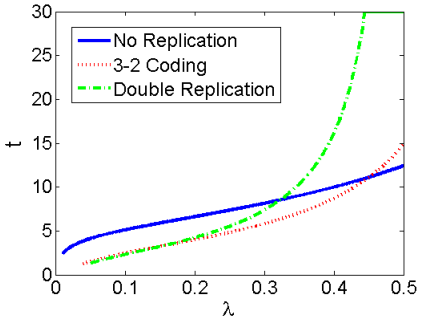


(c) Server Variability

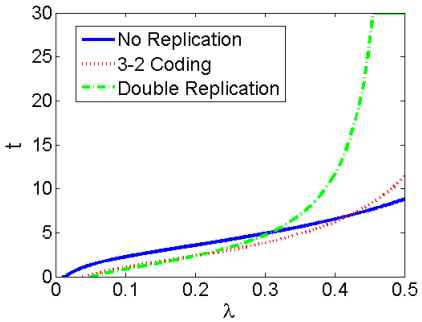
Figure 6.8: Optimal Mean vs. No Coding



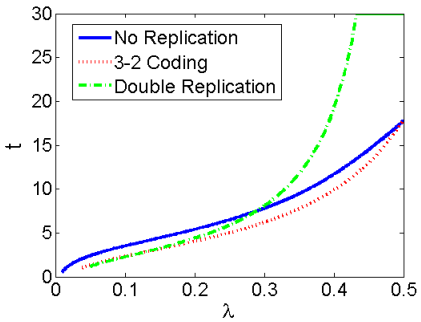
(a) $\epsilon = 0.01$, Exponential



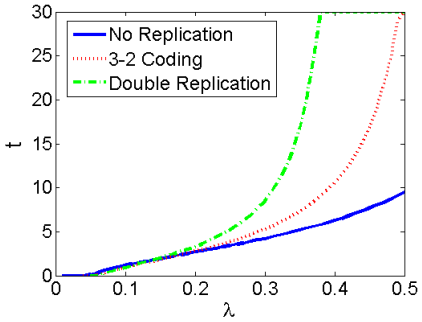
(b) $\epsilon = 0.001$, Exponential



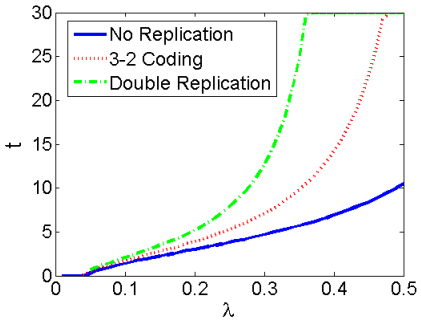
(c) $\epsilon = 0.01$, Hyper-exponential



(d) $\epsilon = 0.001$, Hyper-exponential



(e) $\epsilon = 0.01$, Exceptional First Service



(f) $\epsilon = 0.001$, Exceptional First Service

Figure 6.9: The delay bound τ_ϵ, with varied λ

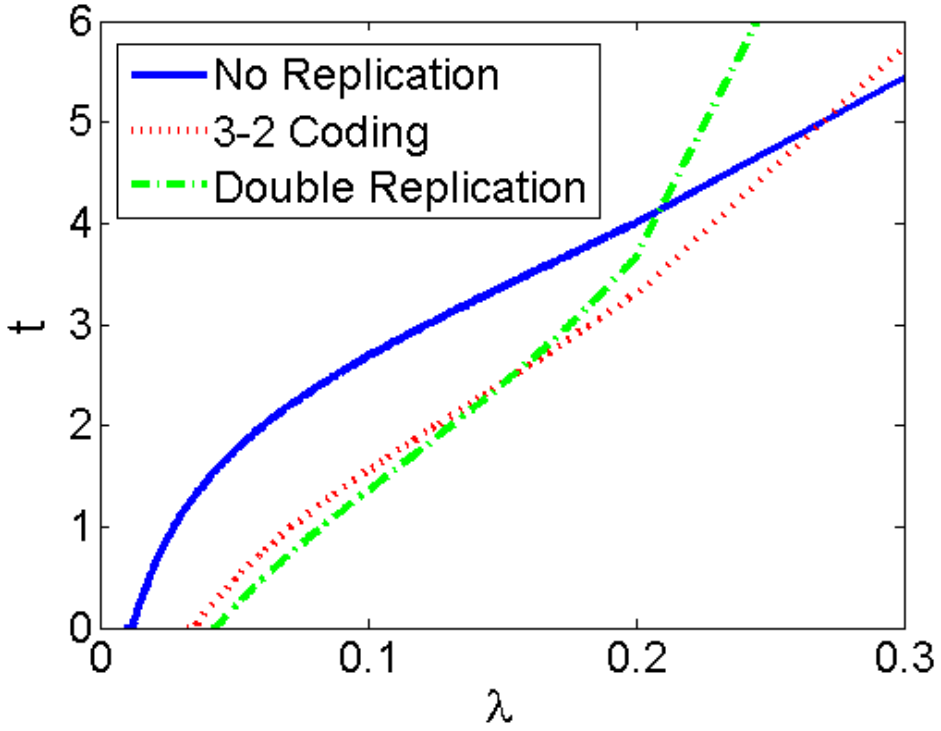


Figure 6.10: Delay Bounds Magnified, On-Off server, $\epsilon = 0.01$

6.4 Conclusion

The purpose of this chapter was to extend the results from Chapter 5 by using network coding. The results from Chapter 5 showed that there was an achievable gain, in terms of a reduction in latency, when using redundancy with a network load below a certain threshold. The results presented in this chapter showed that this is also possible when coding is employed. However, although the coding schemes can increase the network load by a non-integer value, it was found that in general the optimal policy in terms of delay is to use replication rather than coding as the first packet to arrive can be used immediately. Furthermore, the use of network coding can increase the complexity of the transmission and the impact of this in terms of delay remains an open question for further work.

Chapter 7

Conclusion

Many conclusions have been drawn within each chapter of the thesis. This section therefore aims to summarise the contributions made in the area of networking, and explore areas of further work.

7.1 Contributions

The motivation for the work in this thesis is to support high-criticality applications by improving latency. One approach to this might be through routing, which in turn motivated interest in SDN. Within this area, the thesis investigated the problem of controller placement. Another approach is through data replication, and both replication and coding based approaches were investigated.

The thesis started by investigating the open questions for SDN in high criticality, and highlighting the areas of importance. After identifying that a high-criticality network requires bounds on the latency, and a high level of reliability, new methods for the controller placement were investigated. Due to this fact, a number of approximation algorithms were studied. This led to the work in Chapter 3 that analysed the accuracy and scalability (complexity) of multiple approximations for the controller placement.

The first theme of the thesis was to use a physical aspect of SDN to help the enabling of SDN in high criticality. However, the second theme of the

thesis was related to using SDN as an enabler. The use of replicating data in a network with the hope of reducing latency was then introduced. In order analyse the potential reduction in latency a queuing theory model was derived. A general approach was realised and the model included complicated service distributions with characteristics related to wireless channels. One of the outputs of the analysis was an optimal replication factor, which was calculated with respect to the mean delay for a given load on the network. The other main output was the reliability in terms of the delay. Both aspects were shown to have crossover points when the level of replication and load were altered. This showed that the best policy is not always straightforward and gives rise to the potential of dynamic policies in the future. The work in Chapter 5 then led to an extension to the modelling of network coding using queuing theory.

Modelling network coding requires similar, but more complicated analysis to that of replication. The contributions with respect to modelling network coding showed that network coding can provide a smoother transition between increasing the load and reducing latency. This is due to the possibility of a non-integer coding rate. Normally, the analysis on network coding is limited to that of reliability, but by utilising the queuing model the gains in terms of latency can also be seen, both in terms of mean delay and the reliability in terms of the delay.

7.2 Further Work

This thesis provides advancements in the potential use of SDN in high-criticality networking. However, the work presented here has many potential extensions. The controller placement for example is limited by the fact that it is formulated as a k-median problem. This means that a specific value of k must be considered before a placement solution is calculated. This is a severe limitation and future work would be to alleviate this restriction. One such suggestion to remove this restriction is structuring the problem as a facility location problem. This would mean a total acceptable cost would be required as an input and the minimum number of controllers would be

placed in order to fulfil this.

The modelling with respect to queuing theory can be extended in a number of ways. The limitation of the work in this thesis is the strict assumptions made, which make the analysis more tractable in a closed form, but also make the model more unrealistic. In particular, the assumption of independence between queues is unrealistic as when the messages are replicated, there is a dependence created across the service times on each path. This is because the service time is likely to be similar for a specific packet as it has the same size and structure as its replicates. Potential future work for this would therefore be to break the independence assumption, but this would likely lead to much more complicated analysis.

Another extension to the queuing models would be to change the arrival distribution. The assumption that the packets arrive according to a Poisson process makes the analysis much easier, especially since the known results for the $M/G/1$ class of queues can be used. However, there is still a debate to how well internet traffic is modelled by a Poisson process, and other arrival distributions may be better suited. For example, batch arrivals may better characterise the burstiness of networking traffic, and therefore provide a more accurate analysis.

When coding is employed the analysis relies on the order statistic to gain a wait time distribution. Although this provides a closed form expression this may be a limitation if more complex coding schemes are considered. For example the analysis only considers a simple $(3, 2)$-coding scheme. Although this provides an indication of the wait-time for a coding scheme, much more complicated coding schemes are used in practice. One such class of codes are fountain codes, or in particular Raptor codes. Raptor codes work by sending coded packets until 'enough' packets are received to decode the original message. In this case using order statistics may not be the best way to analyse the delay distributions and therefore provokes further work.

As the results presented in this thesis are generally focused on analytical results, there is a scope for further work in both emulation and simulation. Some work has already been done in this area for redundancy as introduced in [83] and [84]. However, a fully deployed system employing redundancy

seems to remain elusive, and simulation/emulation results in specific areas may help aid the progression in this area.

Appendix A

A.1 Proof of Lemma 5.3.1

Lemma A.1.1. *Consider an $M/PH/1$ queue with Poisson(λ) arrival process, and iid service times with a $PH(\boldsymbol{\alpha}, S)$ distribution of order m. The LST of the waiting time distribution is given by:*

$$W^*(s) = \frac{1 + \lambda \boldsymbol{\alpha} S^{-1} \mathbf{1}}{1 - \lambda \boldsymbol{\alpha}(sI - S)^{-1}\mathbf{1}}, \tag{A.1}$$

where I denotes the $m \times m$ identity matrix, and $\mathbf{1}$ a column vector of all ones of length m.

Proof. Observe that $S\mathbf{1} + \mathbf{s^0} = \mathbf{0}$, as $\mathbf{s^0}$ was chosen to make the rows of the rate matrix describing the phase type distribution sum to zero. Consequently,

$$\begin{aligned} (sI - S)\mathbf{1} &= s\mathbf{1} + \mathbf{s^0}, \\ (sI - S)^{-1}\mathbf{s^0} &= \mathbf{1} - s(sI - S)^{-1}\mathbf{1}. \end{aligned}$$

Also, $\alpha_0 + \boldsymbol{\alpha}\mathbf{1} = 1$, as $(\alpha_0, \boldsymbol{\alpha})$ is a probability vector. Substituting these into the LST of the service time distribution given in (5.5), after simplification the following expression is obtained

$$1 - g(s) = s\boldsymbol{\alpha}(sI + S)^{-1}\mathbf{1}.$$

Substituting this into the P-K formula (5.4), and noting that $\rho = \lambda \mathbb{E}[S]$, where the mean service time is given by $\mathbb{E}[S] = -\boldsymbol{\alpha} S^{-1}\mathbf{1}$, the claim of the

lemma is obtained after some straightforward manipulations. □

A.2 Hyper-Exponential Equations

The wait time with hyper-exponential service can be written as follows:

$$\begin{aligned} W^*(s) &= \frac{(1-\lambda(\frac{p}{\mu_1}+\frac{1-p}{\mu_2}))}{1-\lambda(\frac{p}{\mu_1+s}+\frac{1-p}{\mu_2+s})} \\ &= \frac{(1-\lambda(\frac{p}{\mu_1}+\frac{1-p}{\mu_2}))(s+\mu_1)(s+\mu_2)}{(s+\mu_1)(s+\mu_2)-\lambda p(s+\mu_2)-\lambda(1-p)(s+\mu_1)} \end{aligned} \tag{A.2}$$

When inverting the LST it is found that the PDF is a mixture of exponentials given by:

$$\begin{aligned} f_{W(t)} &= (1-\rho)\delta(t) + \frac{e^{-tx_1}}{s_1-s_2} - \frac{e^{-tx_2}}{s_1-s_2} \\ F_{W(t)} &= 1-\rho - \frac{1+e^{-tx_1}}{x_1(s_1-s_2)} + \frac{1+e^{-tx_2}}{x_2(s_1-s_2)} \\ &\textit{where:} \\ &\quad s_i \text{ are the roots of the denominator of } W^*(s) \\ &\quad x_i = -s_i(1-\rho)(s_i+\mu_1)(s_i+\mu_2) \quad i \in 1,2 \\ &\quad \rho = \lambda\mathbb{E}[S] = \lambda\left(\frac{p}{\mu_1}+\frac{1-p}{\mu_2}\right) \end{aligned} \tag{A.3}$$

A.3 Derivation for the Exceptional First Service Model

A.3.1 Derive $g(s)$ and $g_0(s)$

Deriving the distribution for the service time is this first hurdle when considering the vacations model due to the consideration of the availability of the server/channel and subsequently retransmissions. The path state model in Figure A.1 is a representation of the state (available or unavailable) of the server/channel, in this case the path being available and unavailable is

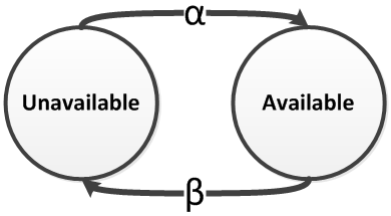


Figure A.1: The On-Off Channel for the exceptional first service model

defined by the exponential distribution with rate α and β respectively. In the proposed model the equations for $g(s)$ and consequently $g_0(s)$ are derived using this path state model and are given in Lemmas A.3.1 and A.3.2 respectively.

Lemma A.3.1. *The LST of a service distribution with retransmissions is given by:*

$$g(s) = \mathbb{E}(e^{-sT}) = \frac{\frac{\mu}{\mu+\beta}\frac{\mu}{\mu+s}}{1 - \frac{\beta}{\mu+\beta+s}\frac{\alpha}{\alpha+s}} \tag{A.4}$$

Proof. If the server was always available, the distribution would be easy as it would be an M/M/1 queue, with service rate μ.

However, in this model this is not the case. Therefore the first thing to consider is the probability that a packet is served before the server becomes unavailable. This is $\mathbb{P}$(Service time $<$ Time it takes the server to go offline) = $\mathbb{P}$(Service finishes|Server remains available) = $\frac{\mu}{\mu+\beta}$ since this is the same as the probability that the service time is the minimum of the service time and the time to go offline. Similarly, the probability that the transmission does not complete is therefore $\frac{\mu}{\mu+\beta}$.

This means that if the usual service time, call it T_0, is exponential with service rate μ then the actual service time is restricted by the fact that the message must complete service before the path becomes unavailable. Hence, T is not a straight forward distribution.

Consider the variable X_i where $X_i \sim exp(\mu + \beta) + \exp(\alpha)$, where the first term of X_i corresponds to the server going off before finishing , and the second term corresponds to the server being in the off state.

The distribution of the service time can now be derived by applying the

respective probabilities with the corresponding events as follows:

$$T = \begin{cases} exp(\mu) & \text{w.p. } \left(\frac{\beta}{\mu+\beta}\right)^0 \frac{\mu}{\mu+\beta} \\ exp(\mu) + X_1 & \text{w.p. } \left(\frac{\beta}{\mu+\beta}\right)^1 \frac{\mu}{\mu+\beta} \\ ... & \\ exp(\mu) + X_1 + ... + X_n & \text{w.p. } \left(\frac{\beta}{\mu+\beta}\right)^n \frac{\mu}{\mu+\beta} \end{cases} \tag{A.5}$$

In general the distribution can now be represented as:

$$T = exp(\mu) + \sum_{i=0}^{N} X_i \text{ where } X_0 = 0 \tag{A.6}$$

and N has the given distribution:

$$\mathbb{P}(N = n) = \frac{\mu}{\mu+\beta}\frac{\beta}{\mu+\beta}^N ; n = 0, 1, 2... \tag{A.7}$$

Since the distribution for T is effectively a sum of exponential distributions the LST of T can be calculated to give:

$$\mathbb{E}(e^{-sT}|N = n) = \frac{\mu}{\mu+s}(\frac{\mu+\beta}{\mu+\beta+s}\frac{\alpha}{\alpha+s})^n \tag{A.8}$$

Summing over all values of n then gives:

$$\mathbb{E}(e^{-sT}) = \frac{\frac{\mu}{\mu+\beta}\frac{\mu}{\mu+s}}{1 - \frac{\beta}{\mu+\beta+s}\frac{\alpha}{\alpha+s}} \tag{A.9}$$

■ □

Lemma A.3.2. *The LST of a exceptional first service distribution with retransmissions is given by:*

$$\begin{aligned} g_0(s) &= (1-q)g(s) + qg(s)\frac{\alpha}{\alpha+s} \\ q &= \frac{\beta}{\alpha+\beta+\lambda} \end{aligned} \tag{A.10}$$

Proof. Using the LST for the service distribution with retransmissions from Lemma A.3.1 the equation for the exceptional first service can be found. The difference is that there must be a consideration to whether or not service starts in an available or unavailable state. First, let

$$q = \mathbb{P}(\text{New arrival sees channel in the off state})$$

then:

$$g_0 = (1-q)g(s) + qg(s)\frac{\alpha}{\alpha+s} \tag{A.11}$$

Since when the server is in the off state there is an additional amount of time before the service starts, but it is known that this quantity is exponential distributed with rate α.

It is now left to derive q , but since the path state model is known the probability can be calculated by conditioning on the channel being available at time t = 0 and integrating over all values of t as follows:

$$\begin{aligned}
q &= \int_0^\infty \lambda e^{-\lambda t}\mathbb{P}(\text{Channel is off at time t}|\text{On at time 0})dt \\
&=^* \int_0^\infty \lambda e^{-\lambda t}\begin{pmatrix}0 & 1\end{pmatrix} e^{Qt}\begin{pmatrix}1\\0\end{pmatrix} \\
&= \int_0^\infty \frac{\lambda e^{-\lambda t}}{\alpha+\beta}\begin{pmatrix}0 & 1\end{pmatrix}\begin{pmatrix}1 & \alpha\\ 1 & -\beta\end{pmatrix}\begin{pmatrix}1 & 0\\ 0 & e^{-(\alpha+\beta)}\end{pmatrix}\begin{pmatrix}\beta & \alpha\\ 1 & -1\end{pmatrix}\begin{pmatrix}1\\0\end{pmatrix}dt \\
&= \frac{\lambda\beta}{\alpha+\beta}\int_0^\infty e^{-\lambda t}(1-e^{-(\alpha+\beta)t})dt \\
&= \frac{\lambda\beta}{\alpha+\beta}\left(\frac{1}{\lambda} - \frac{1}{\alpha+\beta+\lambda}\right) \\
&= \frac{\beta}{\alpha+\beta+\lambda}
\end{aligned} \tag{A.12}$$

*where Q is the rate matrix of the path state model

$$
\begin{aligned}
Q &= \begin{pmatrix} -\alpha & \alpha \\ \beta & -\beta \end{pmatrix} \\
&= \begin{pmatrix} 1 & \alpha \\ 1 & -\beta \end{pmatrix} \begin{pmatrix} 0 & 0 \\ 0 & -(\alpha+\beta) \end{pmatrix} \begin{pmatrix} \beta & \alpha \\ 1 & -1 \end{pmatrix} \frac{1}{\alpha+\beta}
\end{aligned} \tag{A.13}
$$

The diagonalisation of Q is used in the derivation of q to decompose the matrix exponential.

The distribution of g_0 is now completely characterised. □

Using the results from Lemmas A.3.1 and A.3.2, the equations to characterise the exceptional first service model can be found and are presented in equation A.14. The final LST for the wait time and the inverse are particularly complex, so the derivations are shown in a separate section: A.3.3.

$$
\begin{aligned}
g(s) &= \frac{\frac{\mu}{\mu+\beta}\frac{\mu}{\mu+s}}{1-\frac{\beta}{\mu+\beta+s}\frac{\alpha}{\alpha+s}} \\
G &= \frac{\alpha(\mu+\beta)+\beta(\mu+\beta+\alpha)}{\mu\alpha(\mu+\beta)} \\
g_0(s) &= (1-q)g(s) + qg(s)\frac{\alpha}{\alpha+s} \\
q &= \frac{\beta}{\alpha+\beta+\lambda} \\
G_0 &= G + \frac{q}{\alpha} \\
\rho &= \lambda G \\
W_s &= A\frac{s-\lambda g_0(s)+\lambda g(s)}{s-\lambda+\lambda g(s)} \\
A &= \frac{1-\rho}{1+(\lambda G_0)-\rho}
\end{aligned} \tag{A.14}
$$

A.3.2 The Expected Service time for the exceptional first service model

The expected service time can be calculated using the LST. First by taking the derivative of $g(s)$ and $g_0(s)$, and then evaluating at $s = 0$.

$$
\begin{aligned}
g(s) &= \frac{\frac{\mu}{\mu+\beta}\frac{\mu}{\mu+s}}{1-\frac{\beta}{\mu+\beta+s}\frac{\alpha}{\alpha+s}} \\
\mathbb{E}(W_s) &= \frac{dg(s)}{ds}|_{s=0}
\end{aligned}
$$

Let

$$
A = \frac{\alpha\beta}{(\alpha+s)(\beta+\mu+s)} - 1 \tag{A.15}
$$

Then

$$
\mathbb{E}(W_s) = \frac{\mu^2}{A(\beta+\mu)(\mu+s)^2} - \frac{\mu^2(\frac{\alpha\beta}{(\alpha+s)(\beta+\mu+s)^2} + \frac{\alpha\beta}{(\alpha+s)^2(\beta+\mu+s)})}{A^2(\beta+\mu)(\mu+s))}
$$

$$
\begin{aligned}
\mathbb{E}(W_{s_0}) &= \frac{dg_0(s)}{ds}|_{s=0} \\
\frac{dg_0(s)}{ds} &= \frac{\mu^2(\frac{\beta}{\alpha+\beta+\lambda}-1)}{A(\beta+\mu)(\mu+s)} - \frac{\alpha\beta\mu^2}{A(\beta+\mu)(\alpha+s)(\mu+s)(\alpha+\beta+\lambda)}
\end{aligned} \tag{A.16}
$$

$$
\begin{aligned}
\mathbb{E}(W_{s_0}) &= \frac{dg_0(s)}{ds}|_{s=0} \\
&= \frac{\mu(\frac{\beta}{\alpha+\beta+\lambda}-1)}{(\frac{\beta}{\beta+\mu}-1)(\beta+\mu)} - \frac{\beta\mu}{(\beta+\mu)(\frac{\beta}{\beta+\mu}-1)(\alpha+\beta+\lambda)}
\end{aligned} \tag{A.17}
$$

A.3.3 LST and PDF of the Wait time for exceptional service

By substituting all the equations given in A.14 in to W_s the LST for the wait time can be found. Since the exceptional first service model is used the equation is particularly complicated and is given in equation A.18.

LST for the wait time

$$
\begin{aligned}
W_s =& A\frac{W_s^{Num}}{W_s^{Den}} \\
W_s^{Num} =& s(\beta+\mu)(\mu+s)(s^2+(\alpha+\mu+\beta)s+\alpha\mu) \\
& -\lambda\mu^2((\alpha+s)(q+1)-\alpha(q-1))(\beta+\mu+s) \\
W_s^{Den} =& \mu^2(\alpha+s)(\beta+\mu+s)-(\beta+\mu)(\mu+s)(\lambda-s)(s^2+(\alpha+\mu+\beta)s+\alpha\mu)
\end{aligned}
\tag{A.18}
$$

Inverting this transform produced a horrendous equation, A.19. However, a key result was the fact that the equation is still a sum of exponentials. This meant it could be approximated by the slowest decaying exponential, as discussed in the main text.

Inverse LST

$$
\begin{aligned}
f_W(t) = \Sigma_{r\in\mathbf{R}}\frac{A}{A'}\{&\delta(t)+(\beta\lambda r^2+\beta^2\lambda r+\lambda\mu r^2+\lambda\mu^2 r+\alpha\lambda\mu^2+\beta+\lambda\mu^2+\beta^2\lambda\mu \\
&+\alpha\beta\lambda r+\lambda\mu^3 q+\alpha\lambda\mu r+3\beta\lambda\mu r+\lambda\mu^2 qr+2\alpha\beta\lambda\mu+\beta\lambda\mu^2 q)exp(rt)\}
\end{aligned}
\tag{A.19}
$$

$$
\begin{aligned}
A' =& 2\alpha\mu^2-\beta^2\lambda+2\beta\mu^2+\beta^2\mu+3\beta r^2+2\beta^2 r-\lambda\mu^2+3\mu r^2+4\mu^2 r+\mu^3 \\
&-\alpha\beta\lambda+2\alpha\beta\mu+2\alpha\beta r-\alpha\lambda\mu-3\beta\lambda\mu-2\beta\lambda r+2\alpha\mu r+6\beta\mu r-2\lambda\mu r
\end{aligned}
\tag{A.20}
$$

where $\mathbf{R}$ is the set of roots from:

$$
\begin{aligned}
&(\mu+\beta)s^3+(\alpha\beta+3\beta\mu+\alpha\mu+2\mu^2+\beta^2-\lambda\mu-\beta\lambda)s^2 \\
&-(3\beta\lambda\mu+2\alpha\beta\mu-\alpha\lambda\mu-\alpha\beta\lambda+2\beta\mu^2+\beta^2\mu+2\alpha\mu^2+\mu^3-\lambda\mu^2-\beta^2\lambda)s \\
&-2\alpha\beta\lambda\mu+\alpha\beta\mu^2-\beta^2\lambda\mu-\beta\lambda\mu^2-\alpha\lambda\mu^2+\alpha\mu^3
\end{aligned}
\tag{A.21}
$$

A benefit to using the LST is that the mean can be found by differentiating

it and evaluating at $s = 0$. In particular: $\mathbb{E}(W_t) = \frac{dW_s}{ds}|_{s=0}$.

$$
\begin{aligned}
\mathbb{E}(W_t) =& \frac{dW_s}{ds}|_{s=0} \\
=& - \frac{(((\lambda(\beta(\alpha+\beta+\mu)+\alpha(\beta+\mu)))}{(\alpha\mu(\beta+\mu))-1)\left(\frac{\lambda\beta}{((\beta+\mu)(\frac{\beta}{(\beta+\mu)}-1)(\alpha+\beta+\lambda))} - \frac{(\beta/(\alpha+\beta+\lambda)-1)}{((\beta+\mu)(\frac{\beta}{(\beta+\mu)}-1))}\right)} \\
&+ \frac{(\mu(\beta/(\alpha+\beta+\lambda)-1)(\beta/(\beta+\mu)^2+\beta/(\alpha(\beta+\mu))))}{((\beta+\mu)(\beta/(\beta+\mu)-1)^2)} \\
&+ (\beta\mu)/(\alpha(\beta+\mu)(\beta/(\beta+\mu)-1)(\alpha+\beta+\lambda)) \\
&- \frac{(\beta\mu(\beta/(\beta+\mu)^2+\beta/(\alpha(\beta+\mu))))}{((\beta+\mu)(\beta/(\beta+\mu)-1)^2(\alpha+\beta+\lambda)))} \\
&- \lambda/((\beta+\mu)(\beta/(\beta+\mu)-1)) \\
&+ \frac{(\lambda\mu(\beta/(\beta+\mu)^2+\beta/(\alpha(\beta+\mu))))}{\frac{((\beta+\mu)(\beta/(\beta+\mu)-1)^2)-1))}{\lambda+\frac{(\lambda\mu)}{((\beta+\mu)(\beta/(\beta+\mu)-1))})\frac{\lambda\beta}{(\alpha(\alpha+\beta+\lambda))}}} \\
&+ (\beta(\alpha+\beta+\mu)+\alpha(\beta+\mu))/(\alpha\mu(\beta+\mu))) \\
&- (\lambda(\beta(\alpha+\beta+\mu)+\alpha(\beta+\mu)))/(\alpha\mu(\beta+\mu))+1)) \\
&- ((\lambda((\mu(\beta/(\alpha+\beta+\lambda)-1))/((\beta+\mu)(\beta/(\beta+\mu)-1)) \\
&- (\beta\mu)/((\beta+\mu)(\beta/(\beta+\mu)-1)(\alpha+\beta+\lambda))) \\
&+ \frac{(\lambda\mu)}{\frac{((\beta+\mu)(\beta/(\beta+\mu)-1)))((\lambda(\beta(\alpha+\beta+\mu)+\alpha(\beta+\mu)))}{(\alpha\mu(\beta+\mu))-1)(\lambda/((\beta+\mu)(\beta/(\beta+\mu)-1))}} \\
&- \frac{(\lambda\mu(\beta/(\beta+\mu)^2+\beta/(\alpha(\beta+\mu))))}{\frac{((\beta+\mu)(\frac{\beta}{(\beta+\mu)}-1)^2)+1))}{((\lambda+(\lambda\mu)/((\beta+\mu)(\frac{\beta}{(\beta+\mu)}-1)))^2\frac{\lambda\beta}{\alpha(\alpha+\beta+\lambda)}}} \\
&+ (\beta(\alpha+\beta+\mu)+\alpha(\beta+\mu))/(\alpha\mu(\beta+\mu))) \\
&- (\lambda(\beta(\alpha+\beta+\mu)+\alpha(\beta+\mu)))/(\alpha\mu(\beta+\mu))+1))
\end{aligned}
\tag{A.22}
$$

Bibliography

[1] "ONF: Open Networking Foundation," https://www.opennetworking.org, accessed: 2014-09-30.

[2] T. D. Nadeau and K. Gray, *SDN: Software Defined Networks: An Authoritative Review of Network Programmability Technologies.* " O'Reilly Media, Inc.", 2013.

[3] B. Galloway and G. P. Hancke, "Introduction to industrial control networks," *IEEE Communications surveys & tutorials*, vol. 15, no. 2, pp. 860–880, 2013.

[4] B. Heller, R. Sherwood, and N. McKeown, "The controller placement problem," in *Proceedings of the first workshop on Hot topics in software defined networks.* ACM, 2012, pp. 7–12.

[5] J. Åkerberg, M. Gidlund, and M. Björkman, "Future research challenges in wireless sensor and actuator networks targeting industrial automation," in *Industrial Informatics (INDIN), 2011 9th IEEE International Conference on.* IEEE, 2011, pp. 410–415.

[6] V. C. Gungor and G. P. Hancke, "Industrial wireless sensor networks: Challenges, design principles, and technical approaches," *IEEE Transactions on industrial electronics*, vol. 56, no. 10, pp. 4258–4265, 2009.

[7] M. R. Hung Nguyen, Nickolas Falkner. Internet topology zoo. [Online]. Available: http://www.topology-zoo.org/

[8] S. H. Yeganeh, A. Tootoonchian, and Y. Ganjali, "On scalability of software-defined networking," *IEEE Communications Magazine*, vol. 51, no. 2, pp. 136–141, 2013.

[9] P. Xiao, W. Qu, H. Qi, Z. Li, and Y. Xu, "The sdn controller placement problem for wan," in *Communications in China (ICCC), 2014 IEEE/CIC International Conference on*. IEEE, 2014, pp. 220–224.

[10] M. Kobayashi, S. Seetharaman, G. Parulkar, G. Appenzeller, J. Little, J. Van Reijendam, P. Weissmann, and N. McKeown, "Maturing of openflow and software-defined networking through deployments," *Computer Networks*, vol. 61, pp. 151–175, 2014.

[11] M. P. Fernandez, "Comparing openflow controller paradigms scalability: Reactive and proactive," in *Advanced Information Networking and Applications (AINA), 2013 IEEE 27th International Conference on*. IEEE, 2013, pp. 1009–1016.

[12] S. Sharma, D. Staessens, D. Colle, M. Pickavet, and P. Demeester, "Enabling fast failure recovery in openflow networks," in *Design of Reliable Communication Networks (DRCN), 2011 8th International Workshop on the*. IEEE, 2011, pp. 164–171.

[13] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, "Nox: towards an operating system for networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 3, pp. 105–110, 2008.

[14] A. Tootoonchian, M. Ghobadi, and Y. Ganjali, "Opentm: traffic matrix estimator for openflow networks," in *International Conference on Passive and Active Network Measurement*. Springer, 2010, pp. 201–210.

[15] D. Y. Huang, K. Yocum, and A. C. Snoeren, "High-fidelity switch models for software-defined network emulation," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*. ACM, 2013, pp. 43–48.

[16] S. Scott-Hayward, G. O'Callaghan, and S. Sezer, "Sdn security: A survey," in *Future Networks and Services (SDN4FNS), 2013 IEEE SDN For.* IEEE, 2013, pp. 1–7.

[17] S. Sezer, S. Scott-Hayward, P. K. Chouhan, B. Fraser, D. Lake, J. Finnegan, N. Viljoen, M. Miller, and N. Rao, "Are we ready for sdn? implementation challenges for software-defined networks," *IEEE Communications Magazine*, vol. 51, no. 7, pp. 36–43, 2013.

[18] P. Porras, S. Shin, V. Yegneswaran, M. Fong, M. Tyson, and G. Gu, "A security enforcement kernel for openflow networks," in *Proceedings of the first workshop on Hot topics in software defined networks.* ACM, 2012, pp. 121–126.

[19] L. F. Müller, R. R. Oliveira, M. C. Luizelli, L. P. Gaspary, and M. P. Barcellos, "Survivor: an enhanced controller placement strategy for improving sdn survivability," in *Global Communications Conference (GLOBECOM), 2014 IEEE.* IEEE, 2014, pp. 1909–1915.

[20] A. Manzalini, R. Saracco, C. Buyukkoc, P. Chemouil, S. Kuklinski, A. Gladisch, M. Fukui, E. Dekel, D. Soldani, M. Ulema *et al.*, "Software-defined networks for future networks and services," in *White Paper based on the IEEE Workshop SDN4FNS*, 2013.

[21] "Software-defined networking: The new norm for networks," ONF White Paper, 2012.

[22] K. Jeong, J. Kim, and Y.-T. Kim, "Qos-aware network operating system for software defined networking with generalized openflows," in *Network Operations and Management Symposium (NOMS), 2012 IEEE.* IEEE, 2012, pp. 1167–1174.

[23] B. Brungard, M. Betts, and N. Spreche, "Mpls-tp requirements section 2.5 58a," http://tools.ietf.org/html/draft-ietf-mpls-tp-requirements-10#section-2.5.2, 2009.

[24] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker, "Ethane: Taking control of the enterprise," in *ACM SIGCOMM Computer Communication Review*, vol. 37, no. 4. ACM, 2007, pp. 1–12.

[25] P. Fonseca, R. Bennesby, E. Mota, and A. Passito, "A replication component for resilient openflow-based networking," in *Network Operations and Management Symposium (NOMS), 2012 IEEE*. IEEE, 2012, pp. 933–939.

[26] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "Devoflow: Scaling flow management for high-performance networks," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4, pp. 254–265, 2011.

[27] M. Yu, J. Rexford, M. J. Freedman, and J. Wang, "Scalable flow-based networking with difane," *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 4, pp. 351–362, 2010.

[28] D. Levin, A. Wundsam, B. Heller, N. Handigol, and A. Feldmann, "Logically centralized?: state distribution trade-offs in software defined networks," in *Proceedings of the first workshop on Hot topics in software defined networks*. ACM, 2012, pp. 1–6.

[29] ——, "Sdn control simulator," github.com/cryptobanana/sdnctrlsim.

[30] A. Tootoonchian and Y. Ganjali, "Hyperflow: A distributed control plane for openflow," in *Proceedings of the 2010 internet network management conference on Research on enterprise networking*, 2010, pp. 3–3.

[31] J. Stribling, Y. Sovran, I. Zhang, X. Pretzer, J. Li, M. F. Kaashoek, and R. Morris, "Flexible, wide-area storage for distributed systems with wheelfs." in *NSDI*, vol. 9, 2009, pp. 43–58.

[32] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar, "Flowvisor: A network virtualization layer," *OpenFlow Switch Consortium, Tech. Rep*, pp. 1–13, 2009.

[33] Y. Hu, W. Wang, X. Gong, X. Que, and S. Cheng, "Balanceflow: controller load balancing for openflow networks," in *Cloud Computing and Intelligent Systems (CCIS), 2012 IEEE 2nd International Conference on*, vol. 2. IEEE, 2012, pp. 780–785.

[34] S. Hassas Yeganeh and Y. Ganjali, "Kandoo: a framework for efficient and scalable offloading of control applications," in *Proceedings of the first workshop on Hot topics in software defined networks*. ACM, 2012, pp. 19–24.

[35] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama *et al.*, "Onix: A distributed control platform for large-scale production networks." in *OSDI*, vol. 10, 2010, pp. 1–6.

[36] S. Asadullah, J. D. (Editor), B. Khasnabish, G. Reffet, E. R. (Editor), M. S. (Editor), and P. Smacchia, "Migration tools and metrics," https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/migration-tools-and-metrics.pdf, accessed: 2017-06-01.

[37] Cacti, "Cacti network logging and graphing too," http://www.cacti.net/, accessed: 2014-08-06.

[38] R. Sherwood, "Cbench: An openflow controller benchmarker," http://archive.openflow.org/wk/index.php/Oflops, accessed: 2014-09-12.

[39] UBIQUBE, "Msactivator," http://www.ubiqubesolutions.com/?page_id=25, accessed: 2014-07-10.

[40] "Openflow ofpeck," http://archive.openflow.org/wk/index.php/Ofpeck, note = Accessed: 2014-09-11.

[41] "Openflow openseer," http://archive.openflow.org/wk/index.php/OpenSeer, note = Accessed: 2014-09-11.

[42] HP, "Hp openview performance manager," https://h20392.www2.hp.com/portal/swdepot/displayProductInfo.do?productNumber=PERFMINFO, accessed: 2014-05-16.

[43] Y. Hu, T. Luo, N. C. Beaulieu, and C. Deng, "The energy-aware controller placement problem in software defined networks," *IEEE Communications Letters*, vol. 21, no. 4, pp. 741–744, 2017.

[44] Y. Hu, W. Wang, X. Gong, X. Que, and S. Cheng, "On reliability-optimized controller placement for software-defined networks," *Communications, China*, vol. 11, no. 2, pp. 38–54, 2014.

[45] N. Kumar, S. N. Swain, and C. S. R. Murthy, "Convex hull inspired distributed controller placement for assisting d2d transfers in lte-a networks," in *Wireless Communications and Networking Conference (WCNC), 2017 IEEE*. IEEE, 2017, pp. 1–6.

[46] S. Lange, S. Gebert, T. Zinner, P. Tran-Gia, D. Hock, M. Jarschel, and M. Hoffmann, "Heuristic approaches to the controller placement problem in large scale sdn networks," *IEEE Transactions on Network and Service Management*, vol. 12, no. 1, pp. 4–17, 2015.

[47] I. CPLEX, "Ilog cplex homepage 2017," *Available at [online]: https://www-01.ibm.com/software/info/ilog/*, 2017.

[48] M. Tanha, D. Sajjadi, and J. Pan, "Enduring node failures through resilient controller placement for software defined networks," in *Global Communications Conference (GLOBECOM), 2016 IEEE*. IEEE, 2016, pp. 1–7.

[49] Y. Hu, T. Luo, W. Wang, and C. Deng, "On the load balanced controller placement problem in software defined networks," in *Computer and Communications (ICCC), 2016 2nd IEEE International Conference on*. IEEE, 2016, pp. 2430–2434.

[50] A. Sallahi and M. St-Hilaire, "Optimal model for the controller placement problem in software defined networks," *IEEE Communications Letters*, vol. 19, no. 1, pp. 30–33, 2015.

[51] A. Farshin and S. Sharifian, "A chaotic grey wolf controller allocator for software defined mobile network (sdmn) for 5th generation of cloud-based cellular systems (5g)," *Computer Communications*, 2017.

[52] G. Wang, Y. Zhao, J. Huang, Q. Duan, and J. Li, "A k-means-based network partition algorithm for controller placement in software defined network," in *Communications (ICC), 2016 IEEE International Conference on*. IEEE, 2016, pp. 1–6.

[53] Y. Hu, W. Wendong, X. Gong, X. Que, and C. Shiduan, "Reliability-aware controller placement for software-defined networks," in *Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on*. IEEE, 2013, pp. 672–675.

[54] M. Shindler, "Approximation algorithms for the metric k-median problem," *Written Qualifying Exam Paper, University of California, Los Angeles. Cited on*, p. 44, 2008.

[55] A. Tamir, "An o (pn2) algorithm for the p-median and related problems on tree graphs," *Operations Research Letters*, vol. 19, no. 2, pp. 59–64, 1996.

[56] Y. Bartal, "Probabilistic approximation of metric spaces and its algorithmic applications," in *Foundations of Computer Science, 1996. Proceedings., 37th Annual Symposium on*. IEEE, 1996, pp. 184–193.

[57] ——, "On approximating arbitrary metrices by tree metrics," in *Proceedings of the thirtieth annual ACM symposium on Theory of computing*. ACM, 1998, pp. 161–168.

[58] J. Fakcharoenphol, S. Rao, and K. Talwar, "A tight bound on approximating arbitrary metrics by tree metrics," in *Proceedings of the*

thirty-fifth annual ACM symposium on Theory of computing. ACM, 2003, pp. 448–455.

[59] M. Charikar, C. Chekuri, A. Goel, and S. Guha, "Rounding via trees: deterministic approximation algorithms for group steiner trees and k-median," in *Proceedings of the thirtieth annual ACM symposium on Theory of computing.* ACM, 1998, pp. 114–123.

[60] M. Charikar, S. Guha, É. Tardos, and D. B. Shmoys, "A constant-factor approximation algorithm for the k-median problem," in *Proceedings of the thirty-first annual ACM symposium on Theory of computing.* ACM, 1999, pp. 1–10.

[61] K. Jain and V. V. Vazirani, "Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and lagrangian relaxation," *Journal of the ACM (JACM)*, vol. 48, no. 2, pp. 274–296, 2001.

[62] M. Charikar and S. Guha, "Improved combinatorial algorithms for the facility location and k-median problems," in *Foundations of Computer Science, 1999. 40th Annual Symposium on.* IEEE, 1999, pp. 378–388.

[63] V. Arya, N. Garg, R. Khandekar, A. Meyerson, K. Munagala, and V. Pandit, "Local search heuristics for k-median and facility location problems," *SIAM Journal on computing*, vol. 33, no. 3, pp. 544–562, 2004.

[64] S. Arora, P. Raghavan, and S. Rao, "Approximation schemes for euclidean k-medians and related problems," in *Proceedings of the thirtieth annual ACM symposium on Theory of computing.* ACM, 1998, pp. 106–113.

[65] J.-H. Lin and J. S. Vitter, "Approximation algorithms for geometric median problems," *Information Processing Letters*, vol. 44, no. 5, pp. 245–249, 1992.

[66] ——, "e-approximations with minimum packing constraint violation," in *Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*. ACM, 1992, pp. 771–782.

[67] M. R. Korupolu, C. G. Plaxton, and R. Rajaraman, "Analysis of a local search heuristic for facility location problems," *Journal of algorithms*, vol. 37, no. 1, pp. 146–188, 2000.

[68] P. Indyk, "Sublinear time algorithms for metric space problems," in *Proceedings of the thirty-first annual ACM symposium on Theory of computing*. ACM, 1999, pp. 428–434.

[69] D. Arthur and S. Vassilvitskii, "k-means++: The advantages of careful seeding," in *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, 2007, pp. 1027–1035.

[70] Y. Zhang, "Solving large-scale linear programs by interior-point methods under the matlab environment," *Optimization Methods and Software*, vol. 10, no. 1, pp. 1–31, 1998.

[71] K. M. Anstreicher, "Linear programming in o ([n3/ln n] l) operations," *SIAM Journal on Optimization*, vol. 9, no. 4, pp. 803–812, 1999.

[72] D. Arthur and S. Vassilvitskii, "How slow is the k-means method?" in *Proceedings of the Twenty-second Annual Symposium on Computational Geometry*, ser. SCG '06. New York, NY, USA: ACM, 2006, pp. 144–153. [Online]. Available: http://doi.acm.org/10.1145/1137856.1137880

[73] D. Hock, M. Hartmann, S. Gebert, M. Jarschel, T. Zinner, and P. Tran-Gia, "Pareto-optimal resilient controller placement in sdn-based core networks," in *Teletraffic Congress (ITC), 2013 25th International*. IEEE, 2013, pp. 1–9.

[74] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood, "On controller performance in software-defined networks." *Hot-ICE*, vol. 12, pp. 1–6, 2012.

[75] M. F. Bari, A. R. Roy, S. R. Chowdhury, Q. Zhang, M. F. Zhani, R. Ahmed, and R. Boutaba, "Dynamic controller provisioning in software defined networks," in *Network and Service Management (CNSM), 2013 9th International Conference on*. IEEE, 2013, pp. 18–25.

[76] A. Chilwan, K. Mahmood, O. N. Østerbø, and M. Jarschel, "On modeling controller-switch interaction in openflow based sdns," *International Journal of Computer Networks & Communications*, vol. 6, no. 6, p. 135, 2014.

[77] L. Zhang, Z. Zhao, Y. Shu, L. Wang, and O. W. Yang, "Load balancing of multipath source routing in ad hoc networks," in *Communications, 2002. ICC 2002. IEEE International Conference on*, vol. 5. IEEE, 2002, pp. 3197–3201.

[78] H. Charara, J.-L. Scharbarg, J. Ermont, and C. Fraboul, "Methods for bounding end-to-end delays on an afdx network," in *Real-Time Systems, 2006. 18th Euromicro Conference on*. IEEE, 2006, pp. 10–pp.

[79] J.-Y. Le Boudec and P. Thiran, *Network calculus: a theory of deterministic queuing systems for the internet*. Springer Science & Business Media, 2001, vol. 2050.

[80] F. Ciucu and J. Schmitt, "Perspectives on network calculus: no free lunch, but still good value," in *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication*. ACM, 2012, pp. 311–322.

[81] F. Ciucu, A. Burchard, and J. Liebeherr, "Scaling properties of statistical end-to-end bounds in the network calculus," *IEEE Transactions on Information Theory*, vol. 52, no. 6, pp. 2300–2312, 2006.

[82] S. Azodolmolky, R. Nejabati, M. Pazouki, P. Wieder, R. Yahyapour, and D. Simeonidou, "An analytical model for software defined networking: A network calculus-based approach," in *Global Communications Conference (GLOBECOM), 2013 IEEE*. IEEE, 2013, pp. 1397–1402.

[83] A. Vulimiri, P. B. Godfrey, R. Mittal, J. Sherry, S. Ratnasamy, and S. Shenker, "Low latency via redundancy," in *Proc. of the 9th ACM conf. on Emerging networking experiments and technologies*. ACM, 2013, pp. 283–294.

[84] A. Vulimiri, O. Michel, P. Godfrey, and S. Shenker, "More is less: reducing latency via redundancy," in *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*. ACM, 2012, pp. 13–18.

[85] S.-J. Lee and M. Gerla, "Split multipath routing with maximally disjoint paths in ad hoc networks," in *Communications, 2001. ICC 2001. IEEE International Conference on*, vol. 10. IEEE, 2001, pp. 3201–3205.

[86] D. G. Andersen, A. C. Snoeren, and H. Balakrishnan, "Best-path vs. multi-path overlay routing," in *Proceedings of the 3rd ACM SIGCOMM Conference on Internet Measurement*, ser. IMC '03. New York, NY, USA: ACM, 2003, pp. 91–100. [Online]. Available: http://doi.acm.org/10.1145/948205.948218

[87] N. F. Maxemchuk, "Dispersity routing," in *Proc. of ICC*, vol. 75, 1975, pp. 41–10.

[88] N. Maxemchuk, "Dispersity routing: Past and present," in *Military Communications Conference, 2007. MILCOM 2007. IEEE*. IEEE, 2007, pp. 1–7.

[89] K. Gardner, S. Zbarsky, S. Doroudi, M. Harchol-Balter, E. Hyytiä, and A. Scheller-Wolf, "Queueing with redundant requests: First exact analysis," 2014.

[90] A. Kumar, R. Tandon, and T. Clancy, "On the latency and energy efficiency of distributed storage systems."

[91] Y. Cui, S. Xiao, C. Liao, I. Stojmenovic, and M. Li, "Data centers as software defined networks: Traffic redundancy elimination with wireless

cards at routers," *Selected Areas in Communications, IEEE Journal on*, vol. 31, no. 12, pp. 2658–2672, 2013.

[92] N. B. Shah, K. Lee, and K. Ramchandran, "The mds queue: Analysing the latency performance of erasure codes," in *2014 IEEE International Symposium on Information Theory*. IEEE, 2014, pp. 861–865.

[93] ——, "When do redundant requests reduce latency?" in *Communication, Control, and Computing (Allerton), 2013 51st Annual Allerton Conference on*. IEEE, 2013, pp. 731–738.

[94] ——, "The mds queue: Analysing latency performance of codes and redundant requests," Technical Report, Tech. Rep., 2012.

[95] A. Aissani, "A retrial queue with redundancy and unreliable server," *Queueing systems*, vol. 17, no. 3-4, pp. 431–449, 1994.

[96] J. Y. Yen, "Finding the k shortest loopless paths in a network," *management Science*, vol. 17, no. 11, pp. 712–716, 1971.

[97] D. Eppstein, "Finding the k shortest paths," in *Foundations of Computer Science, 1994 Proceedings., 35th Annual Symposium on*. IEEE, 1994, pp. 154–165.

[98] D. Sidhu, R. Nair, and S. Abdallah, "Finding disjoint paths in networks," *SIGCOMM Comput. Commun. Rev.*, vol. 21, no. 4, pp. 43–51, Aug. 1991. [Online]. Available: http://doi.acm.org/10.1145/115994.115998

[99] W. Willinger and V. Paxson, "Where mathematics meets the internet," *Notices of the AMS*, vol. 45, no. 8, pp. 961–970, 1998.

[100] T. Karagiannis, M. Molle, M. Faloutsos, and A. Broido, "A nonstationary poisson view of internet traffic," in *INFOCOM 2004. Twenty third Annual Joint Conference of the IEEE Computer and Communications Societies*, vol. 3. IEEE, 2004, pp. 1558–1569.

[101] J. Hollinghurst, A. Ganesh, and T. Baugé, "Latency reduction in communication networks using redundant messages," in *Teletraffic Congress (ITC 29), 2017 29th International*, vol. 1. IEEE, 2017, pp. 241–249.

[102] R. Kanagavelu, B. S. Lee, R. F. Miguel, L. N. Mingjie *et al.*, "Software defined network based adaptive routing for data replication in data centers," in *Networks (ICON), 2013 19th IEEE International Conference on*. IEEE, 2013, pp. 1–6.

[103] S. Satapathy, "System and method for software defined network aware data replication," May 3 2016, uS Patent 9,330,156.

[104] G. Xu, J. Yang, and B. Dai, "Challenges and opportunities on network resource management in dcn with sdn," in *Big Data (Big Data), 2015 IEEE International Conference on*. IEEE, 2015, pp. 1785–1790.

[105] N. F. Maxemchuk, "Dispersity routing in high-speed networks," *computer networks and ISDN systems*, vol. 25, no. 6, pp. 645–661, 1993.

[106] N. B. Shah, K. Lee, and K. Ramchandran, "When do redundant requests reduce latency?" *IEEE Trans. on Comms*, vol. 64, no. 2, pp. 715–722, 2016.

[107] K. Gardner, S. Zbarsky, S. Doroudi, M. Harchol-Balter, and E. Hyytia, "Reducing latency via redundant requests: Exact analysis," *ACM SIGMETRICS Perf. Eval. Review*, vol. 43, no. 1, pp. 347–360, 2015.

[108] J. N. Daigle, "The basic m/g/1 queueing system," *Queueing Theory with Applications to Packet Telecommunication*, pp. 159–223, 2005.

[109] A. J. Ganesh, N. O'Connell, and D. J. Wischik, *Big queues*. Springer, 2004.

[110] H. Takagi, *Queueing analysis: a foundation of performance evaluation, vol. 1 : vacation and priority systems*, ser. Queuing Analysis.

[111] K. Gardner, M. Harchol-Balter, A. Scheller-Wolf, and B. Van Houdt, "A better model for job redundancy: Decoupling server slowdown and job size," *MASCOTS*, 2016.

[112] D. Szabó, F. Németh, B. Sonkoly, A. Gulyás, and F. H. Fitzek, "Towards the 5g revolution: A software defined network architecture exploiting network coding as a service," *SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, pp. 105–106, Aug. 2015. [Online]. Available: http://doi.acm.org/10.1145/2829988.2790025

[113] S. Liu and B. Hua, "Ncos: A framework for realizing network coding over software-defined network," in *Local Computer Networks (LCN), 2014 IEEE 39th Conference on*. IEEE, 2014, pp. 474–477.

[114] D. Szabo, A. Gulyas, F. H. Fitzek, and D. E. Lucani, "Towards the tactile internet: Decreasing communication latency with network coding and software defined networking," in *European Wireless 2015; 21th European Wireless Conference; Proceedings of*. VDE, 2015, pp. 1–6.

[115] W.-T. Tan and A. Zakhor, "Real-time internet video using error resilient scalable compression and tcp-friendly transport protocol," *IEEE Transactions on Multimedia*, vol. 1, no. 2, pp. 172–186, 1999.

[116] A. Boukerche, R. W. N. Pazzi, and R. B. Araujo, "A fast and reliable protocol for wireless sensor networks in critical conditions monitoring applications," in *Proceedings of the 7th ACM international symposium on Modeling, analysis and simulation of wireless and mobile systems*. ACM, 2004, pp. 157–164.