Software Defined Networking for Radio telescopes

A case study on the applicability of SDN for MeerKAT

Presented by:
Martin Slabber

Supervisors:
Neco Ventura and Dr Joyce Mwangama
Department of Electrical and Electronics Engineering
University of Cape Town

Submitted to the Department of Electrical Engineering at the University of Cape Town in fulfilment of the academic requirements for a Masters of Science degree in Electrical Engineering.

October 2021

Declaration

1. I know that plagiarism is wrong. Plagiarism is to use another's work and pretend that it is one's own.
2. I have used the IEEE convention for citation and referencing. Each contribution to, and quotation in, this report from the work(s) of other people has been attributed, and has been cited and referenced.
3. This report is my own work.
4. I have not allowed, and will not allow, anyone to copy my work with the intention of passing it off as their own work or part thereof.
5. I know the meaning of plagiarism and declare that all the work in the document, save for that which is properly acknowledged, is my own. This thesis/dissertation has been submitted to the Turnitin module (or equivalent similarity and originality checking software) and I confirm that my supervisor has seen my report and any concerns revealed by such have been resolved with my supervisor.

Signature ..... Signed by candidate ..... (Martin Slabber)

Date .....20 October 2021.....

Acknowledgements

I want to take this opportunity to thank my supervisors Neco Ventura and Dr Joyce Mwangama, for their help and allowing me to accomplish this work in the time frame available to me. The part-parttime was challenging for all of us. Your ability to describe different points of view and broad subject knowledge has been invaluable to this dissertation and me.

Much appreciation goes to Dr Jason Manley for his explanations, encouragement, and allowing me to build on from his research. Thank you to my employer SARAO for allowing this study and partial financial support. To Khutšo Ngoasheng, my line manager at SARAO, thank you for your backing and always asking with interest about my studies. To my colleague, Thomas Bennett thank you for proofreading and technical feedback.

To my wife and two young boys, thank you Anne, Max, and Joe, this would be nothing without your love and support.

And to my parent that has taught and exemplified curiosity and hard work. THIS is all your fault, thank you very much.

Abstract

Scientific instruments like radio telescopes depend on high-performance networks for internal data exchange. The high bandwidth data exchange between the components of a radio telescope makes use of multicast networking. Complex multicast networks are hard to maintain and grow, and specific installations require modified network switches. This study evaluates Software Defined Networking (SDN) for use in the MeerKAT radio telescope to alleviate the management complexity and allow for a vendor-neutral implementation. The purpose of this dissertation is to verify that an SDN multicast network can produce suitable paths for data flow through the network and to see if such an implementation is easier to maintain and grow.

There is little literature regarding SDN for radio telescope networks; however, there is considerable work where different aspects of SDN are discussed and demonstrated for video streaming. SDN with multicast for video streaming, although simpler, forms the background research. Considerable work was put into understanding and documenting the different aspects of a radio telescope affecting the data network.

The telescope network controller generates the OpenFlow rules required by the SDN controller and is a new concept introduced in this work. The telescope network controller is fitted with two placement algorithms to demonstrate its flexibility. Both algorithms are suitable for the expected workload, but they produce very different traffic patterns. The two algorithms are not compared to one another, they were created to demonstrate the ease of adding domain-specific knowledge to an SDN. The telescope network controller makes it easy to introduce and use new flow placement algorithms, thus making traffic engineering feasible for the radio telescope.

Complex multicast networks are easier to maintain and grow with SDN. SDN allows customised packet forwarding rules typically unattainable with standard routing and other standard network protocols and implementations. A radio telescope with a software-defined data network is resilient, easier to maintain, vendor-neutral, and possesses advanced traffic engineering mechanisms.

Contents

Abstract ii

List of Figures vii

List of Tables ix

List of Acronyms x

1 Introduction 1

1.1 Research motivation 3

1.2 Problem definition 5

1.3 Dissertation objectives 6

1.4 Research questions 7

1.5 Scope and limitations 8

1.6 Dissertation outline 8

2 Background 10

2.1 Literature Review 10

2.2 MeerKAT radio telescope 12

2.2.1 Packet-switched network 13
2.2.2 Receptor 15
2.2.3 Subarray 15
2.2.4 Baseline 16

2.3 Hard real-time signal processing 17
2.3.1 Digitiser 17
2.3.2 Correlator 21
2.3.3 SKARAB 27
2.3.4 Beamformer 27
2.4 Network 28
2.4.1 Data network 28
2.4.2 Clos network topology 31
2.5 Soft real-time signal processing 34
2.5.1 Science Data Processing 34
2.5.2 User Supplied Equipment 34
2.6 Software Defined Networking 35

3 System design **37**

3.1 Description of data distribution 37
3.1.1 Network topology 38
3.1.2 Spine switch as Rendezvous Point 38
3.1.3 Data processing 41
3.2 Problem description 41
3.2.1 Multicast networking 41
3.2.2 Network management 43
3.2.3 Summary 44
3.3 Design approach 45
3.3.1 Rationale 46
3.3.2 Methodology process 46
3.4 Solution design 46
3.4.1 System description 46
3.4.2 Functional system architecture 49
3.4.3 SDN application 52
3.4.4 Initialisation scripts 53
3.4.5 Model design 53
3.4.6 Database 56

3.4.7 Flow rule construction . . . 57
3.5 Simulation . . . 58
3.6 Summary . . . 59

4 Implementation **62**

4.1 Modelling . . . 62
4.1.1 Python . . . 63
4.1.2 PostgreSQL . . . 63
4.1.3 Model configuration . . . 64
4.1.4 Model set up . . . 66
4.1.5 Subarray . . . 67
4.1.6 Visualisation and reporting . . . 68
4.2 Network simulation . . . 68
4.2.1 ContainerNet . . . 69
4.2.2 Ryu . . . 70

5 Results and discussion **71**

5.1 Interpretation of plots . . . 72
5.2 Load balancing flow placement method . . . 73
5.3 Worst case scenario . . . 75
5.3.1 Bin packing flow placement method . . . 77
5.4 Number of streams per digitiser . . . 79
5.5 Increased number of receptors . . . 81
5.6 Summary . . . 84

6 Conclusion **85**

6.1 Discussion . . . 86
6.2 Future work . . . 87

A Placement methods **94**

A.1 Pseudo code functions 94

A.2 Balanced placement 95

A.3 Bin packing placement 96

A.4 Examples showing algorithms side by side 96

A.4.1 Small streams 97
A.4.2 Mixed streams 97
A.4.3 Sorted mixed streams 98
A.5 Implementation detail 99

B Units and notation **100**

C Publications **101**

List of Figures

1.1 Telescope data network. 3

2.1 Interactions of major subsystems via the data network. 14
2.2 Packet size between receptor switch and Leaf switch measured from Leaf switch. Subarray performing observation in L-band. . . 19
2.3 Receptor switch connects to each digitiser with four 10 Gb/s links and to a Leaf switch in the KAPB with one 40 Gb/s. 20
2.4 Link bandwidth utilisation between receptor switch and Leaf switch measured from Leaf switch. Subarray performing observation in L-band. 20
2.5 Composition of the different stages, hosts, and engines in the correlator. 22
2.6 Data flow through the correlator, the thickness of the edges are relative to the data rate. 26
2.7 Block diagram showing the high-level data flow through the data network. 28
2.8 Folded-Clos architecture of the data network, showing links between Spine and Leaf switches. 30
2.9 Mellanox Ethernet switches. 31
2.10 Clos network with ingress on the left and egress on the right, each node is a switch with a port number formula as an inscription. . . 32
2.11 Multicast streams in a Clos network. 33
2.12 Multicast streams in a folded-Clos network. 33
2.13 Network control architectures. 36

3.1 Data streams from sources to targets. 39
3.2 Interactions via the data network of Telescope network controller and SDN controller along with major subsystems. 47
3.3 Interactions with telescope network controller. 48
3.4 Abstract system architecture. 49
3.5 UML component diagram of TNC. Including CAM and Network switches. 51
3.6 OpenFlow command construction. 57

5.1 Comparing bandwidth utilisation of Spine switches for different sizes of the first subarray only. 73
5.2 Bandwidth utilisation for each link of experiment A64S18L36D04-64-00-00-00. 74
5.3 Comparing bandwidth utilisation of Spine switches for different subarray combination where subarray 1 was kept at $A_n = 32$. . . 75
5.4 Bandwidth utilisation for each link of experiment A64S18L36D04-33-17-09-05. The load balancing placement method was used on the worst case subarray combination. 76
5.5 Bandwidth utilisation for each link of experiment BA64S18L36D04-33-17-09-05. The bin packing placement method was used on the worst case subarray combination. 78
5.6 Comparing bandwidth utilisation of Spine switches in worst case subarray combination. Biggest difference between A_d and A_n. . . 79
5.7 Comparing bandwidth utilisation of Spine switches for different number of digitiser output streams. 80
5.8 Comparing bandwidth utilisation of Spine switches for subarray 1 combinations of $A_n \in \{64, 90\}$ and digitisers with either 4 or 16 streams. 82
5.9 Bandwidth utilisation for each link of experiment A90S18L36D16-90-00-00-00. 83

A.1 Placement of small flows. 97
A.2 Placement of flows of mixed sizes. 98
A.3 Placement of flows of mixed sizes sorted largest to smallest. . . 99

List of Tables

2.1 Digitiser bands. 18

4.1 Model configuration: Main and subarray sections. 64
4.2 Model configuration: Correlator section. 65
4.3 Model configuration: Network section. 65
4.4 Model configuration: Devices section. 66

B.1 Decimal and binary prefixes. 100

List of Acronyms

1-to-1	one-to-one
1-to-N	one-to-many
ACID	Atomicity, Consistency, Isolation, Durability
ADC	Analog to Digital Converters
API	Application Program Interface
BGP	Border Gateway Protocol
BIDIR-PIM	Bidirectional Protocol Independent Multicast
CAM	Control And Monitoring
CapEx	Capital Expenditure
CASPER	Collaboration for Astronomy Signal Processing and Electronics Research
CLI	Command Line Interface
CMC	Correlator Master Controller
COTS	Commercial Off The Shelf
CRUD	Create, Read, Update, Delete
CSP	Central Signal Processor
CTE	Command Table Expression
DCIM	Data Center Infrastructure Management
DHCP	Dynamic Host Configuration Protocol
DMC	Digitiser Master Controller
DNS	Domain Name System
FAST	Five hundred meter Aperture Spherical Telescope
FPGA	Field-Programmable Gate Arrays
GMRT	Giant Metrewave Radio Telescope
GPS	Global Positioning System
GPU	Graphical Processing Unit
HERA	Hydrogen Epoch of Reionisation Array

HPC	High Performance Computing
IEC	International Electrotechnical Commission
IGMP	Internet Group Multicast Protocol
IP	Internet Protocol
IPAM	Internet Protocol Address Management
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
KAPB	Karoo Array Processing Building
L3	Layer 3
LACP	Link Aggregation Control Protocol
MASER	Microwave Amplification by Stimulated Emission of Radiation
MPI	Message Passing Interface
MTU	Maximum Transmission Unit
NRF	National Research Foundation
N-to-1	many-to-one
N-to-N	many-to-many
OAM	Operations, Administration and Management
OEM	Original Equipment Manufacturer
OOB	Out-Of-Band
OOBM	Out-Of-Band Management
OpEx	Operating Expenditure
OSPF	Open Shortest Path First
PAPER	Precision Array for Probing the Epoch of Reionisation
PFB	Polyphase Filter Bank
PIM	Protocol Independent Multicast
PIM-SM	Protocol Independent Multicast Sparse Mode
PTP	Precision Time Protocol
QSFP+	Quad Small Form-factor Pluggable
RDBMS	Relational DataBase Management System
RDMA	Remote Direct Memory Access
RF	Radio Frequency
ROACH	Reconfigurable Open Architecture Computing Hardware

RP	Rendezvous Point
SARAO	South African Radio Astronomy Observatory
SDLC	Software Development Life Cycle
SDN	Software Defined Networking
SDP	Science Data Processing
SDS	Software Defined Storage
SETI	Search for Extraterrestrial Intelligence
SFP+	enhanced Small Form-factor Pluggable
SI	International System of Units
SKA	Square Kilometer Array
SKARAB	Square Kilometer Array Reconfigurable Application Board
SPEAD	Streaming Protocol for Exchanging Astronomical Data
SQL	Structured Query Language
SSH	Secure Shell
STP	Spanning Tree Protocol
SVC	Scalable Video Coding
TCAM	Ternary Content-Addressable Memory
TCP	Transmission Control Protocol
TNC	Telescope Network Controller
UDP	User Datagram Protocol
UHF	Ultra High Frequency
UML	Unified Modeling Language
USE	User Supplied Equipment
WAN	Wide Area Network

Chapter 1

Introduction

Progress in science [1][2] demands that bigger and more sophisticated instruments need to be built. These instruments require ever-increasing amounts of computational resources. The instruments' construction and running costs, Capital Expenditure (CapEx) and Operating Expenditure (OpEx) respectively, are minimised by using standard commodity servers and networking hardware where possible. This strategy is in line with current trends in large High Performance Computing (HPC) systems. Sophisticated high-performance networks are required to interconnect the computational resources, storage systems and data acquisition components to one another. Ostensibly these networks seem very similar and have common requirements: high bandwidth throughput, and low latency transfers at the lowest possible monetary cost. However, to accommodate for application-specific needs, the network design and more so the implementation differs between HPC and scientific instruments and also between the different instrument types. In many cases, scientific instruments require a network tuned to the application's processing pipeline, whereas HPC requires Message Passing Interface (MPI) and/or Remote Direct Memory Access (RDMA) operations between all pairs of compute nodes. Network architectures like folded-Clos are suited to both scenarios, but the differences come in with the configuration of these networks.

In astronomy, bigger instruments (telescopes) are continually being built to peer ever deeper into space and with increased resolution we are able to see objects in more detail and therefore form a better understanding of the universe. The important measure of the size of a telescope is the synthesised aperture,

which is the effective size of the telescope. Especially in radio astronomy, it is not feasible to build big single-dish telescopes that are bigger than those found at Effelsberg, Green Bank, Arecibo, or the recently completed Five hundred meter Aperture Spherical Telescope (FAST) telescope in China. It is generally easier to compensate for the mechanical difficulties of building an enormous single-dish telescope by building an interferometer telescope. With interferometry, the signal from many smaller dishes are combined into a product equivalent to that of a huge dish. Interferometry enables engineers to move past the mechanical limitations, but it requires significantly more computational capabilities and a high-performance interconnect to connect the sources, targets and compute nodes to one another. Here, a data distribution network plays a pivotal role in the instruments by facilitating the signal transfer and interlinking the compute components allowing these to function as one.

In radio telescopes, multicast networking has been successfully deployed on dedicated data networks that are used to transport huge volumes of real-time digitised-signal streams between processing units within the correlator. As the telescope grows (number of receptors), so does the complexity of configuring the data network, and even more so that of the multicast routing. One of the advantages of multicast is the effectiveness by which it can deliver the same content to multiple targets (destinations). Multicast has been around since the 1980s [3], but it has suffered from complex configuration and expensive (time and effort) management protocols.

With Software Defined Networking (SDN), the control and data planes of the network are separated, and the control decisions for all the switches in a network can be performed from a central location. This central controller (an SDN controller) has a holistic view of the network and so can make a more informed decision on the best route for each data flow. Custom applications can be integrated (or embedded) into SDN controllers to enhance the capabilities of the SDN controller by providing application-specific flow placement methods and other functions to allow the controller to manage the network to best serve the application. SDN can promote multicast by simplifying set up, enabling more complex forwarding schemes, improving resilience and facilitating dynamic routing.

MeerKAT is a 64 receptor radio telescope in the Karoo region of South

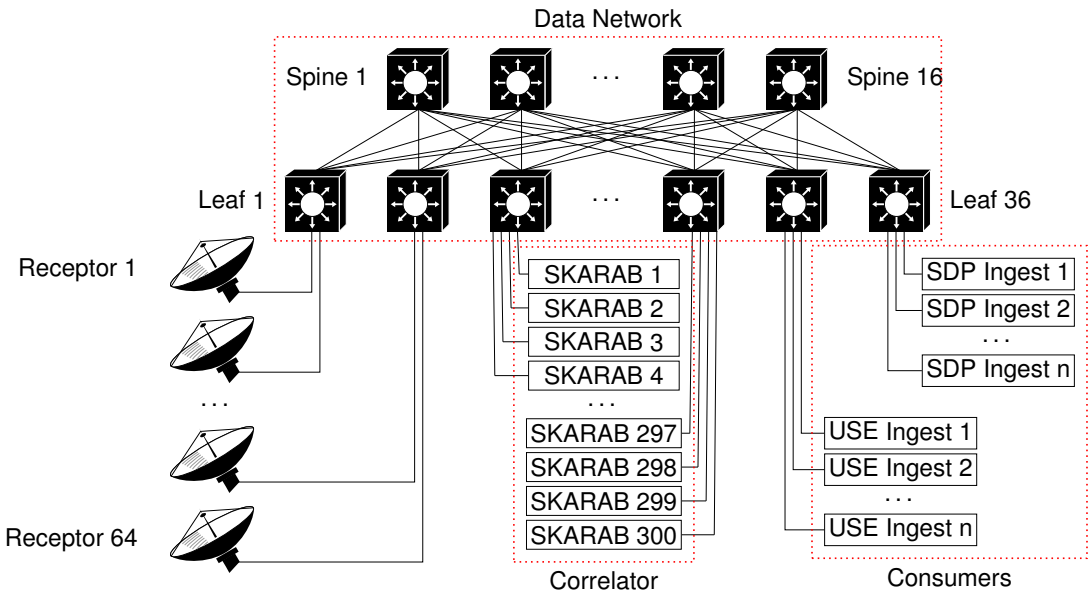


Figure 1.1: Telescope data network.

Africa. MeerKAT is an operational telescope built and run by the South African Radio Astronomy Observatory (SARAO). The very faint signals received by MeerKAT are digitised at the receptors and transported to a central processing facility. There, the signals are processed by a correlator that is built up out of many processing nodes. A data network (Figure 1.1) exists to link the receptors to the correlator and to link the components of the correlator. The consumers of the correlator are also connected to the data network to receive data streams. Multicast is heavily used in this network to facilitate data exchange. This dissertation studies the MeerKAT data network and presents an application that can calculate flow rules for the multicast streams. To configure the switches, an SDN controller application was developed that will apply the calculated flow rules to the switches in the network.

1.1 Research motivation

Multicast routing protocols are challenging to manage and are slow to update in large networks. Each node in the network must perform its own control plane calculations and then propagate its decision outcome to other nodes in the network. This may lead to nodes having to reconsider their previous decisions. The distributed nature of the multicast routing protocol causes a slow convergence on the final routing decision. Often the routing is non-optimal and re-

quires fine-tuning by network engineers to yield the best results[1]. SDN removes the control plane burden from the network nodes and moves this to a central (with high availability if needed) controller. An SDN controller is typically run on Commercial Off The Shelf (COTS) server equipment. It can have powerful processing capabilities at a modest cost. Such an SDN controller has a complete view of the network and can make better decisions, and do so faster than in the case of a distributed control plane. The SDN controller needs an effective strategy to decide the route each flow should take. Unlike typical multicast problems (*e.g.*, video streaming) where the flows are one-to-many (*1-to-N*) (a single source sends to multiple targets), an SDN controller in a radio telescope needs to cater for many-to-many (*N-to-N*) compositions.

Folded-Clos network topology has become a popular topology for building non-blocking networks out of COTS network switches. In such a network, all traffic between sources and targets go through the Spine switches, except in the special case when sources and targets are attached to the same Leaf switch. The controller needs to determine the best Spine switch to use for each multicast stream, taking into consideration the other streams that also occupy the network.

The current generation of radio telescopes is in effect very advanced software-defined radios. Nearly every aspect of the instrument is controlled by software systems that are interlinked to one another, *e.g.*, software defined observation planning and scheduling, software defined control and monitoring, software defined resource allocation and task scheduling, and Software Defined Storage (SDS). SDN provides the tools and opportunity to define the network capabilities from a governing software system. SDN will make it possible to interlink the network function with the other components of the telescope. SDN dramatically improves the possible configurations that can be scheduled (and queued) without network engineers having to change configurations on network switches. Complex network topologies can be more easily realised (or evaluated) when the switch configuration is orchestrated from an SDN controller. Scenarios where links of different speeds and different switch types (for example different port densities) could be considered. Network engineers can go as far as adding multiple links between devices and not have to be concerned with Link Aggregation Control Protocol (LACP) or Spanning Tree Protocol (STP). Vendor

[1] This requires significant skill and expertise, and in-depth understanding of network routing along with a good understanding of the workings of a radio telescope.

interoperability is also significantly easier with SDN[4].

With a software application at the helm of the network configuration, it would be possible for the radio telescope to dynamically handle failures in the network, *e.g.*, a link going down or the loss of a Spine switch.

An application attached or embedded in to an SDN controller can bring a new predictive form of operation to the system. Since the control logic is encapsulated in code, dry-runs of future configurations can be modelled without having to make any changes to the operational instrument. The effect of adding sources and targets can be investigated. This functionality will play an important role in the design of network upgrades, since different switch port density versus cost or port speed trade-off studies can be performed in an empirical manner.

1.2 Problem definition

Multicast routing is the process of propagating Layer 3 (L3) multicast streams from one or more sources to one or more targets. This should be done as efficiently as possible, avoiding duplicate packets as well as not overloading intermediate links and switch port buffers. Packet duplication may result due to multiple paths in a cyclic network [5].

At present, Protocol Independent Multicast (PIM) is the industry standard multicast routing protocol[6]. In Internet Protocol (IP) networks, PIM relies on IP routing protocols like Open Shortest Path First (OSPF) [7] or Border Gateway Protocol (BGP)[8] to find the shortest path and avoid loops.

Several flavours of PIM are available, but Protocol Independent Multicast Sparse Mode (PIM-SM) is arguably the most well known of the multicast routing protocols. PIM-SM [9] works on the basis that a predefined Rendezvous Point (RP) has been chosen for each multicast stream used in the network. Each RP is assigned to a router[2] in the network. The chosen router will depend on the specific device, network topology, and load on the device. A router can be the RP for multiple multicast streams in a network. For each multicast stream, routers in the network must know the address of the RP and be able to reach

[2]In the network under investigation, the switches have the capabilities to function as routers. Routers and switches are used interchangeably.

the RP via unicast networking. The RP switch will build up and maintain two trees (graphs). A target tree is built from the RP to the target devices. Target devices send Internet Group Multicast Protocol (IGMP) join messages to their gateway routers. The gateway router and other routers along the path to the RP will register with the RP by means of PIM join messages. In a similar fashion a source tree is built from the source to the RP; sources send IGMP register messages that get propagated to the RP in the same manner as the IGMP join messages.

Once the RP has constructed the source and target trees, the trees are merged. The resulting graph is optimised by adding shorter paths and pruning. Eventually, the graph converges into one with the shortest paths. With a converged graph, data flow is not necessarily through the RP. By deliberate placement of the RP, a network engineer can guide PIM-SM for faster convergence. PIM-SM is not ideal for traffic engineering since it only considers the shortest path and can not be trained to produce the least cost (Steiner tree) traffic flow. By deliberate insertion of routes and manually adjusting OSPF link cost values, PIM-SM can be tricked into producing the desired graph.

In practice, network engineers have to carefully assign each RP and allocate the associated multicast addresses to the appropriate resource controller in anticipation of the possible configurations of the telescope. Since the telescope can be configured in many modes and has subarrays that can operate at different modes, it has to be foreseen what all the possible combinations of modes are. This is naturally not always possible and leads to artificial limitations imposed on the telescope to reduce the possibility of operating in a mode (or combination of modes) that the network can not support. Since important parts of each switch configuration differ, the majority of switch configuration is done by hand with some assistance from deployment tools. Small mistakes can be hard to diagnose since the configuration of many devices has to be inspected to form a picture of the complete network.

1.3 Dissertation objectives

The first objective of this work is to detail aspects of a radio telescope data network and how the different components interact with one another, and to

describe the vital role the network plays in the system. It is required to identify the parameter space that the network operates in and how changes in the configuration of an observation can have a dramatic impact on the network requirements.

The second and main objective of this dissertation is to establish a framework for determining the placement of flows on the network in such a way that the best use is made of the network's capabilities hereinafter referred to as the model.

The third objective is to demonstrate that a SDN controller application can be implemented that will program flow rules for a *N-to-N* stream in a multicast environment. This is demonstrated in a scaled-down network simulation environment.

1.4 Research questions

The following is a summary of the primary research questions examined in this dissertation:

Q1.1 What is the parameter space that a radio-telescopes data-distribution network operates in with regards to telescope configuration?

Q1.2 Is it possible to programmatically allocate the data flows on the network without overloading any switch ports?

Q1.3 Can SDN be used to replace a multicast routing as used in a radio telescope?

With an environment implemented to answer the primary questions, several secondary questions are addressed:

Q2.1 What is the effect on the network if the digitiser were to use more streams to transmit data?

Q2.2 Can the SDN be expanded by adding more receptors without adding more switches or redevelopment of SDN application?

1.5 Scope and limitations

The research is limited to the data distribution network of a radio telescope. This data distribution network has three primary responsibilities:

1. Bringing data from the receptors into the correlator. With a peak combined bandwidth of 2.4Tb/s.
2. Interconnecting the components of the correlator.
3. Transferring data to one or more secondary processing systems (soft-real time).

This network has the highest bandwidth demand and tightest latency requirements of all networks within a radio telescope. This dissertation only considers changes to the network configuration, the network topology is preserved. Compatibility with the current implementation of digitisers, correlator and other systems are kept. The implementation should not be vendor specific so that it can be transferred to other radio telescopes, and the use of well established industry standards is preferred.

1.6 Dissertation outline

The rest of the dissertation has the following structure.

A literature review is presented in Chapter 2, followed by a detailed breakdown of the telescope data processing pipeline. The data processing components are individually discussed along with their impact on the network.

Chapter 3 describes the problem with multicast routing in respect to the telescope network in more detail. The proposed SDN solution is introduced and explained, along with the processing model and associated components.

The implementation details of the SDN solution and the verification by simulation setup is detailed in Chapter 4.

Chapter 5 provides the results of the SDN controlled data network by presenting several plots accompanied by an explanation of how the solution satis-

fies the research questions.

Chapter 6 concludes the work with a discussion.

Chapter 2

Background

This chapter lays down the ground work and describes the environment. The chapter starts with a literature review, followed by an explanation of the components of a radio telescope. To understand the niche use of Ethernet networking, the data generation properties and component interactions within the telescope are explained. In some cases, necessary formulas are supplied for determining the bandwidth usage for different operational condition and these are matched with observations from the network monitoring system. This is mostly to document it for further work but was necessary to determine the maximum bandwidth expected for the data streams within the current operational constrains. The chapter ends with short explanations of Clos network topology and SDN.

2.1 Literature Review

Broekema *et al*. [10] suggested the use of SDN for the Square Kilometer Array (SKA) radio telescope in 2015. They proposed to use an SDN network for the Science Data Processing (SDP) of the SKA telescope. In their brief mention of SDN, the authors summarise the workings of SDN succinctly.

> *We therefore propose to build a software-defined network (SDN) infrastructure, which will become an integral part of the SDP dataflow, and will fall under the direct control of the SDP monitoring and control system. This means that the network is no longer a static piece*

> *of infrastructure, but may dynamically change configurations to suit the workflow requirements. Such a software-defined network also allows an effective decoupling of sending and receiving nodes. In this model, the sending peers, the Central Signal Processor (CSP)[1] correlator nodes, effectively send to a virtual receiving node, which may or may not physically exist. Receiving nodes subscribe to data flows from the CSP, as directed by the data flow manager. A software network controller directs physical data flows by having switches modify Ethernet headers in transit to match receiving peers: a classic publish-subscribe model, implemented in a network. Support for these technologies is currently available in many newer Ethernet switches from a variety of vendors. However, this is a novel approach to building a sensor network, that needs to be prototyped. [10]*

Although the suggestion was only for the lower bandwidth[2] soft real-time subsystem of the radio telescope, the SDP, we agree and feel it is even more so necessary for the high bandwidth and hard real-time network. A follow-up paper [11] on the subject went into more depth with regards to SDN and several network devices were tested. This early work discussed SDN in a broad sense and stated that SDN could be used to replace multicast networking.

The principle reference with regards to the working of a radio telescope and networking in the radio telescope was the doctoral thesis [12] of Manley. In this work, Manley established and described the systems that became the MeerKAT correlator. This includes the Ethernet-based data network and the use of multicast for signal propagation. We aim to build on that work and provide an update by describing the interplay of systems with the network, specifically for MeerKAT.

Many good papers are available that discuss the use of SDN to improve or make multicast networking viable for video streaming. We will not mention all these but give a general overview. Although on the surface there are similarities between video streaming and signal propagation in a radio telescope, and both require constant latency (low jitter) and consume large amounts of bandwidth, the implementation of the one is not transferable to the other. We found a lot of

[1] CSP in SKA is equivalent to the MeerKAT correlator in the context of this dissertation.

[2] Lower bandwidth is relative here. SDP has lower bandwidth than the correlator but still very high in comparison to many other industries.

inspiration from the various techniques used for the distribution of video through a network with the use of SDN managed multicast. The best solutions tend to be application specific. For instance, the use of Scalable Video Coding (SVC) to transmit multiple streams where a client consumes the streams based on its capabilities; the more streams the client consumes and combines the better the video quality. These video streams are sent as multicast streams allowing clients to join as many as they need without incurring additional load on the sending server. Topology considerations is another divergent aspect between radio telescopes and video streaming. In a radio telescope, we know the topology and can optimise for it, whereas video streaming is typically for the wider internet or large corporate networks where the SDN controller responsible for video streaming cannot be optimised for the network topology.

An early (2012) demonstration of the use of SDN for multicast propagation, with video streaming as the use case, was the Multiflow system [13]. Although the problem of SDN routing was not addressed the authors managed to break ground and demonstrate the advantages of knowing the network topology when multicast paths are determined. Their research was only for a loop free network. Although published after our first few rounds of prototypes the work by Noghani *et al*. [14] follows the same architectural approach as we describe in this dissertation. Their work only focuses on video distribution in a loop-free network. The paper titled “A Survey on Multicasting in Software-Defined Networking“ [15], first describes the problem of multicast in a non-SDN network. The authors then argue the benefits of SDN for multicast propagation and further discusses several algorithms that have been implemented for this purpose.

2.2 MeerKAT radio telescope

In the sparsely populated Karoo region of South Africa, an ambitious engineering project is underway, namely the building of the MeerKAT radio telescope. But MeerKAT, with its 64 steerable dishes [16], is only a pathfinder telescope for the even more ambitious international SKA telescope with a target of 3000 dishes [17].

The design and construction of MeerKAT is done by SARAO, which is an organisation that forms part of the larger National Research Foundation (NRF)

of the South African government. MeerKAT is not the first telescope to be built by SARAO. The much smaller but very successful seven dish KAT-7 telescope has moved through the engineering phases and has been operational as a science instrument for several years.

MeerKAT is a multiband interferometer telescope capable of operating as one large telescope or as several smaller telescopes simultaneously. These smaller telescopes are called subarrays (§2.2.3) and the subarrays are capable of operating at different frequency bands by mechanically selecting a different receiver and digitiser (§2.3.1).

With interferometry, multiple receptors (§2.2.2) are used to synthesise a telescope with a much larger aperture. This simplifies the construction since many smaller dishes can be built compared to one large complex and costly dish. On MeerKAT the receptor is the dish, analogue components, digitisers, control system, and all other equipment that are located at the dish. The MeerKAT receptors are spread out in an area with an 8km diameter. Although interferometry simplifies the mechanical construction and significantly reduces the overall cost of the telescope, it requires significantly more computational resources. This computation is performed by the correlator (§2.3.2), which is a hard real-time signal processing (§2.3) unit responsible for combining the signals from the different receptors. Each subarray has its own correlator. The correlator is constructed out of many computation units and a communication network (§2.4.1) is required to connect these compute units together. The same network is also used to connect the correlator to the receptor digitisers and to downstream soft real-time (§2.5) systems.

2.2.1 Packet-switched network

MeerKAT uses an Ethernet-based packet-switched network (hereafter, the data network) for the distribution of astronomy data (voltage samples, frequency samples, correlation products etc.). This network is built using COTS networking equipment. This led to a reduction in cost and complexity, compared to building a custom specialised signal and data distribution system [12].

On MeerKAT there is no long-distance transfer of analogue signals. The digitiser performs the digitisation and timestamping along with the necessary

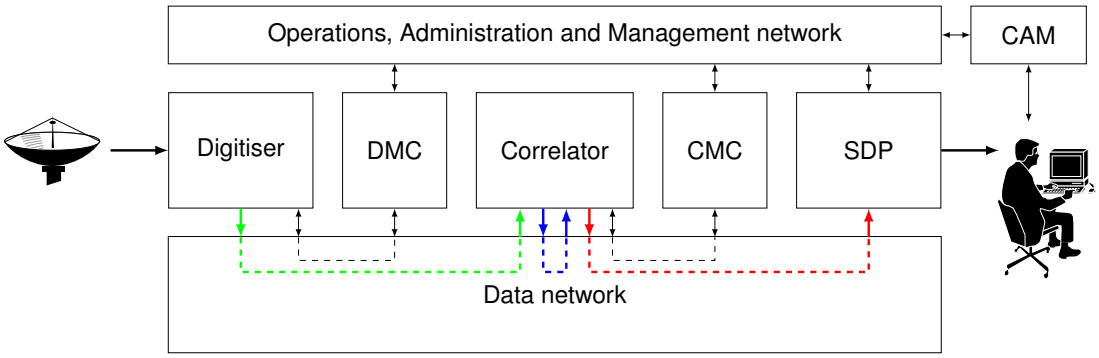


Figure 2.1: Interactions of major subsystems via the data network.

Radio Frequency (RF) functions at the receptor. An extensive underground optical fibre network [18] links the receptors to the central computing facility, the Karoo Array Processing Building (KAPB). At each of the receptors there is a small data switch that is connected, by the optical fibre network, to the data network in the KAPB at 40 Gb/s.

Besides linking the digitisers to the correlator the data network is also used to link the correlator components to one another.

The Precision Array for Probing the Epoch of Re-ionisation (PAPER) [19] and KAT-7 [20] telescopes were early adopters of using Ethernet networks to distribute telescope signal data. Other telescopes that also use Ethernet-based data networks are the Hydrogen Epoch of Reionisation Array (HERA)[21], Giant Metrewave Radio Telescope (GMRT) [22] and Medicina [23] telescopes.

Internet Protocol version 4 (IPv4) multicast is used to effectively distribute data between various components on the data network. This allows for multiple components to subscribe to the same data stream. It also simplifies the configuration since neither sender nor receiver has to know about one another, they only have to be aware of the predefined addresses used for the different data streams.

Along with the data network, there is also an Operations, Administration and Management (OAM) network used for management of devices. This network is used to send instructions to and monitor the behaviour of devices. The OAM network is a shared 1 Gb/s network.

Figure 2.1 is a block diagram that shows the basic interaction between sub-

systems and the data network. The Digitiser Master Controller (DMC) and the Correlator Master Controller (CMC) are gateway systems that provide interfaces to the digitisers and correlator respectively. The DMC and the CMC communicate with the components they manage via the data network but communicate with the Control And Monitoring (CAM) system over the OAM network.

To improve efficiency, Jumbo frames are used on the data network. The majority of packets on the network are larger than 1500 bytes in size and the network Maximum Transmission Unit (MTU) on all the equipment has to be set to a sufficiently large value to increase efficiency.

2.2.2 Receptor

A MeerKAT receptor is the dish, antennas, receivers, digitisers, and all the control mechanisms and electronics that are housed at the dish.

The dish is made up out of 40 aluminium panels and is $13.5\,\mathrm{m}$ in diameter. The dish is steerable in azimuth from $-185°$ to $275°$ at a speed of up to $2°/\mathrm{s}$ and in elevation from $15°$ to $92°$ at a speed of up to $1°/\mathrm{s}$ [24]. The receptor has an offset Gregorian configuration with a $3.8\,\mathrm{m}$ sub-reflector, and there is a mechanical band selector (indexer) housed on the sub-reflector arm. Four receiver-digitiser pairs can be mounted on the indexer. The structure is $19.5\,\mathrm{m}$ high and weighs $42\,000\,\mathrm{kg}$.

2.2.3 Subarray

Since MeerKAT consists out of multiple receptors, it is possible to fulfil the needs of various simultaneous scientific observations if these observations do not require all the receptors. It is also useful to run engineering tests on a small subset of receptors while the majority are active in scientific-observations. Some observation types do not yet need the high resolution made possible when all receptors are used *e.g.*, sky surveys and pulsar timing. Further, some researchers cannot handle the data volume associated with observations that used all the receptors and structure their research around multiple smaller observations. For these and other reasons, MeerKAT was designed to support multiple subar-

rays[3]. MeerKAT is configured to operate with four concurrent subarrays, which is a practical size but could be changed in configuration. Subarrays are operated independently from one another and cannot share resources like receptors and correlators. Subarrays share the same infrastructure this includes the data network, the pool of correlator compute units and other processing nodes. Subarrays are managed from the CAM system and have access to the same CMC, DMC, and SDP systems.

Scientists request observations via telescope-operations, when these observations have been planned and are scheduled the equipment assigned to the subarray are configured and activated. In this process, resources are configured based on the requested and available resources and the configuration of the subarray. Physical actions like moving the receptor indexer to the correct position to enable observations in the right band are performed. Firmware is flushed to the correlator compute units and all the systems taking part in the array are configured and activated. As part of the configuration multicast addresses for the various data-streams are issued. Each subarray uses different multicast address blocks. A subarray can be active from a few minutes to days.

2.2.4 Baseline

A baseline is the product (multiplication) of any two outputs from the digitisers. Since the baseline is calculated in the correlator we can also state that it is the product of any two correlator inputs. When A_n receptors take part in a subarray there will be $\frac{A_n(A_n-1)}{2}$ baselines in the array. On each receptor, there is only one active digitiser. Each digitiser has streams for the horizontal and vertical polarised signals, thus two outputs per digitisers. For each subarray, there will be twice as many inputs ($2A_n$) to the correlators as there are receptors.

The number of cross-correlation products in a subarray can be determined by $\frac{2A_n(2A_n-1)}{2}$. Besides the cross-correlation products the correlator also multiplies each stream with itself to create an auto-correlation product, thus $2A_n$ needs to be added when calculating the number of baselines.

[3] In MeerKAT nomenclature subarray is spelt without a hyphen.

$$B_l = \frac{2A_n(2A_n - 1)}{2} + 2A_n$$
$$= 2A_n^2 + A_n \tag{2.1}$$

Applying subarray sizes 4, 8, 16, 32, and 64 to Equation 2.1 we observe respective number of baselines as 36, 136, 528, 2080, and 8256. As the number of receptors in a subarray grows there is a dramatic increase in the processing requirements.

2.3 Hard real-time signal processing

MeerKAT uses an FX type correlator where digitised voltage signals pass through a Polyphase Filter Bank (PFB) stage, the frequency decomposition stage (F-stage), then the cross product of all the input pairs are calculated (X-stage). It is possible to use an XF type correlator, resulting in more traffic on the data network and thus more costly infrastructure.

2.3.1 Digitiser

On MeerKAT each receptor can be fitted with four independent receiver-digitiser pairs. Only one digitiser per receptor is active at any time. The appropriate receiver-digitiser is selected at subarray configuration time. Setup of the receptor for subarray activation involves turning the indexer to the correct position and enabling the digitisers. This is done remotely by the CAM system allowing for fully automated reconfiguration of the telescope. The receiver includes the analogue components including receiver horn, antenna, filters and amplifiers. Each digitiser receives two conditioned analogue signals one for horizontal polarisation and one for vertical polarisation. These signals are sampled and the resulting voltage measurements are sent to the correlator. In its simplest form, the digitiser can be seen as two high-speed Analog to Digital Converters (ADC) units.

Digitisers are supplied with an extremely accurate time signal sourced from a combination of several on-site reference units (Microwave Amplification by

Stimulated Emission of Radiation (MASER) clock, Rubidium clock and Global Positioning System (GPS)). The on-site time is meticulously kept in sync with other national and international references.

Table 2.1: Digitiser bands.

Name	Frequency range	Sampling rate Samples per second (S_D)	Packet rate Packets per second (P_D)
UHF	0.58 – 1.015 GHz [24]	1088×10^6	5.32×10^5
L-band	0.9 - 1.67 GHz [24]	1712×10^6	8.35×10^5
X-band[4]	8 – 14.5 GHz [24]	-	-
S-band	2 - 4 GHz	1750×10^6	8.54×10^5

The receptors are fitted with receiver-digitiser pairs that are designed and built for different frequency bands. The receiver-digitisers for Ultra High Frequency (UHF), L-band, and S-band have been installed on MeerKAT and X-band will be done at a later stage. Table 2.1 lists the operating frequency ranges of the receivers for each band along with the sampling rate and the number of network packets emitted by each digitiser per second. The digitisers for the mentioned bands differ in several ways but from a network perspective the most noticeable difference is the sampling rate (S_D), this indicates how many times a second the digitiser will sample the incoming analogue signals. The sampling rate ranges from 1088 million samples every second (for UHF) to 1750 million samples every second (for S-band).

The digitiser has four $10\,\mathrm{Gb/s}$ interfaces, two interfaces for sending out horizontal polarisation data and two for vertical polarisation. The digitisers are managed by the DMC, and one of the four interfaces on the digitiser also receives the OAM traffic. The bandwidth consumed by OAM traffic is negligible compared to the observation data, approximately $10\,\mathrm{kb/s}$ per digitisers. OAM traffic should be prioritised over the captured signal data.

Each acquisition (sample) by the digitiser ADC is a $10\,\mathrm{bit}$ value. 4096 samples along with a 40bit timestamp make up an Ethernet packet. Each packet contains $5125\mathrm{B} = \frac{4096\times10\mathrm{bit}+40\mathrm{bit}}{8}$ of raw data. The data is encapsulated and transmitted using the Streaming Protocol for Exchanging Astronomical Data (SPEAD) [25] protocol. Packet size measurements performed on the link between a receptor and the data network can be seen in Figure 2.2. Visually one

[4] X-band digitiser has not yet entered the design stage.

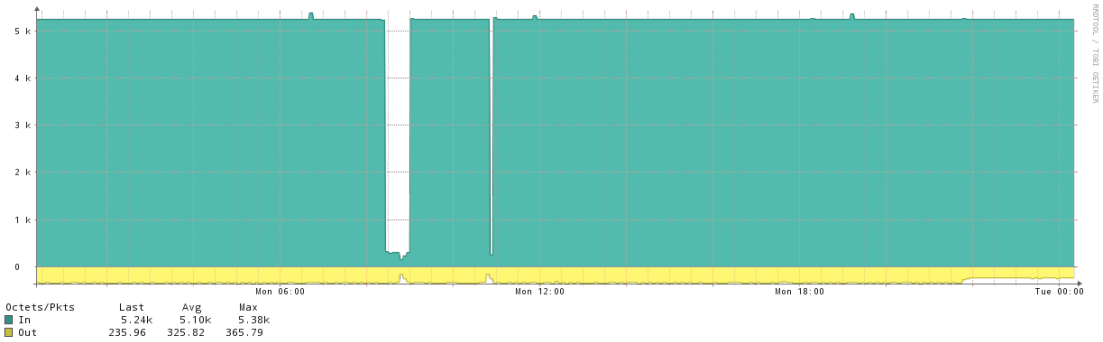


Figure 2.2: Packet size between receptor switch and Leaf switch measured from Leaf switch. Subarray performing observation in L-band.

can see that the measured packet size in Figure 2.2 matches the calculated packet size well. Every 4096 samples the digitiser send two packet onto the network, one for each polarisation. The packets are sent out on alternate interfaces to accommodate the data rate in excess of $10\,\mathrm{Gb/s}$. Equation 2.2 shows how to calculate the number of packets (P_D) sent out onto the network every second by a digitiser.

Each polarisation on the receptor is given a distinct set of multicast addresses on subarray configuration.

$$P_D = \frac{2S_D}{4096} \tag{2.2}$$

Figure 2.3 shows how at each receptor all four digitisers are connected to the same network switch, this switch has a $40\,\mathrm{Gb/s}$ uplink to the data network in the KAPB. Digitisers on a receptor share the switch and uplink since it is only practical to operate one of the digitisers at a time.

The Mellanox SX1012 Ethernet (Figure 2.9a) switch is used as the data switch at the receptor. This switch has 12 Quad Small Form-factor Pluggable (QSFP+) $40\,\mathrm{Gb/s}$ interfaces. The switch supports QSFP+ to four enhanced Small Form-factor Pluggable (SFP+) splitter cables. When using the splitter cable on the MX1012 there is no port loss. The Mellanox SX1012 switch is SDN capable, has good power efficiency, low latency and cost around US$6000 each.

Given the number of packets emitted per second (P_D from Equation 2.2)

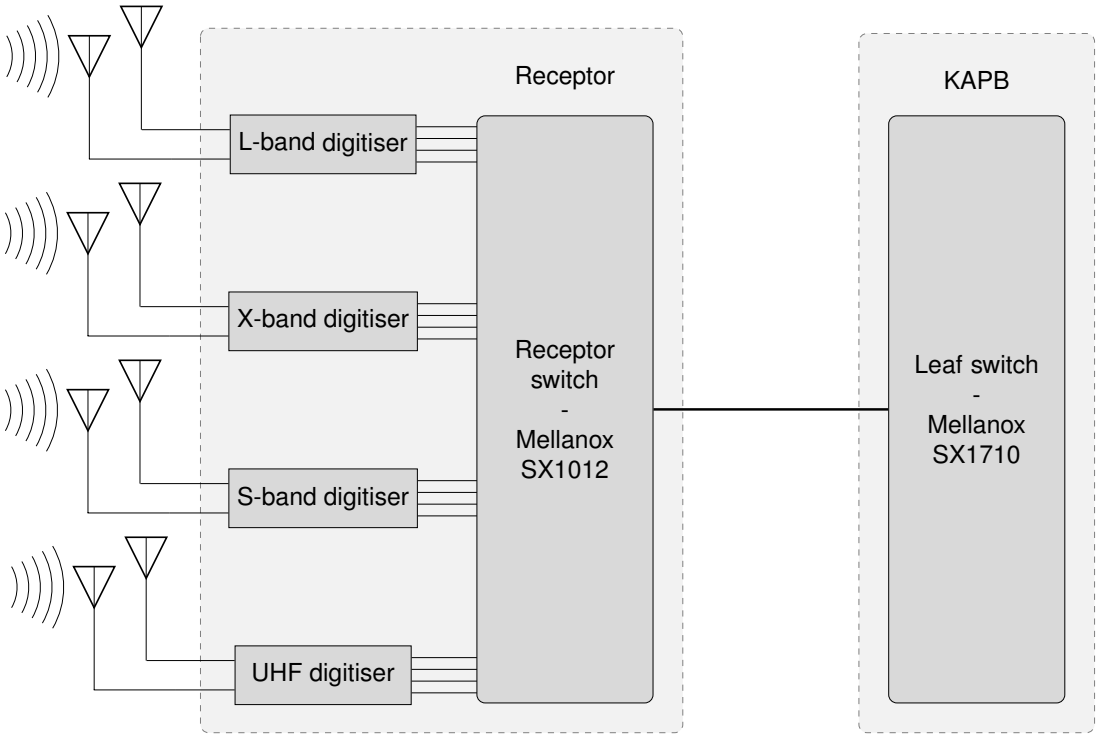


Figure 2.3: Receptor switch connects to each digitiser with four $10\,\mathrm{Gb/s}$ links and to a Leaf switch in the KAPB with one $40\,\mathrm{Gb/s}$.

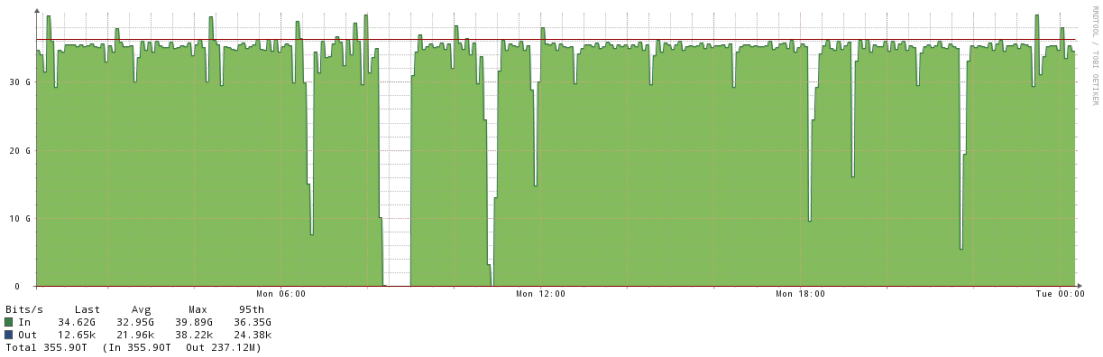


Figure 2.4: Link bandwidth utilisation between receptor switch and Leaf switch measured from Leaf switch. Subarray performing observation in L-band.

and the size of the packet payload (5125 B) we calculate the average bandwidth use of 34.3 Gb/s for L-band and 35.1 Gb/s for S-band. Comparing these values to the measurements (Figure 2.4) of the link between the receptor switch and the data network while and L-band digitiser was active, we see a minuscule protocol overhead. For the remainder of this work, a conservative (inflated) value of 37 Gb/s as the bandwidth used by the digitiser will be used regardless of the selected band. This is to ensure that the network can handle the highest load conditions.

The digitiser is also often referred to as the D-engine or D-host in technical diagrams and discussions, in this dissertation a short form notation D_n is used to reference a digitiser *e.g.*, D_1, D_{27}, and D_{64}.

2.3.2 Correlator

The correlator used in MeerKAT is an FX correlator. The incoming digitised voltages in the time domain are first converted into the frequency domain and channelised; the F-stage. Thereafter the cross multiplication takes place where each frequency channel from all the digitisers are multiplied by one another, this product is averaged over a period (accumulation); the X-stage. Along with the correlation product the correlator can also deliver steerable beams. For MeerKAT the X-stage can output one polarised beam, and if additional beams are required an additional B-stage has to be instantiated.

The correlator is constructed in such a way that it is always necessary for the number of receptors (A_n) to be a power of two [12] and greater than three. Thus 4, 8, 16, 32 or 64 receptors are needed by each correlator.

This is not always possible and when the number or real receptors (A_d) is not a power of two the CAM system will assign data streams of active receptors to dummy receptors to bring the number of receptors (A_n) in a subarray up to the next power of two. Thus if only two receptors ($A_d = 2$) were added to the subarray the correlator will still expect four receptors ($A_n = 4$) so two dummy receptors have to be added and the streams of the existing receptors will be reused. Given A_d, the value of A_n can be calculated using equation 2.3. The added dummy receptor products are removed or ignored when processed by

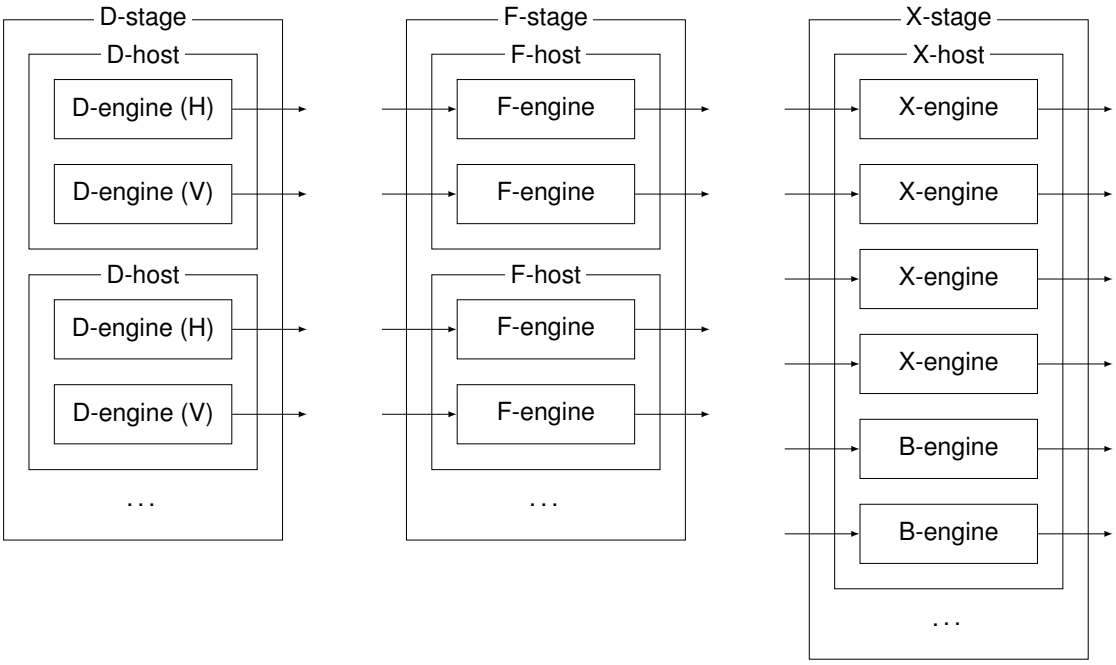


Figure 2.5: Composition of the different stages, hosts, and engines in the correlator.

the soft real-time systems like SDP.

$$A_n = \begin{cases} 4 & \text{if } A_d < 4 \\ 2^{\lceil \log_2 A_d \rceil} & \text{otherwise} \end{cases} \tag{2.3}$$

The correlator stages (F, X, and B) are all constructed from the same hardware components. For this the Square Kilometer Array Reconfigurable Application Board (SKARAB) (§2.3.3) is used. A pool of 300 units are available and on subarray configuration, the CMC will configure SKARAB devices to operate as ether hosts in the F, X, or B stages of the correlator. Each host is capable of performing multiple functions, the functional units are called engines. Hosts in the F-stage will have two F-engines, one for each polarisation. Hosts in the X-stage will have four X-engines and two B-engine. Hosts in the B-stage will have several B-engines, the exact number has not yet been determined. Figure 2.5 illustrates the composition of stages and hosts.

F-stage

The primary function of the F-stage is to perform channelisation on the digitised voltage data (time-domain) and outputs a representation of this data as power levels per channel, where each channel represents a small range of frequency spectrum (frequency domain). The MeerKAT correlator supports 3 channelisation modes (1Ki, 4Ki, and 32Ki), the modes will break up the band into 1024, 4096, or 32768 channels. Where 32Ki mode results in many more channels and each channel is very narrow (small bandwidth).

An F-engine core can perform the processing for only one input. Thus there need to be as many F-engines as there are inputs to the correlator. Since each receptor outputs a horizontal and a vertically polarised signal there is a need for $2A_n$ F-engines in a correlator. Two F-engine can be hosted on one SKARAB processing node. Typically the same SKARAB node processes the horizontal and vertical polarisation's from the same receptor. There are as many SKARAB nodes (F-hosts) in the F-stage as there are receptors configured (A_n) for the correlator.

The F-engine has to receive a sufficient number of samples for the PFB to be performed. It has to gather twice the number of channels of samples before proceeding with the PFB. *e.g.*, In 4Ki mode, 4096 channels will be produced from 8192 samples but in 1Ki mode, only 2048 samples are needed to channelise the signal into the 1024 channels.

The data rate out of the F-engine is independent of the channelisation mode. As the channelisation increase, the F-engine generates more samples but less frequently. The number of samples (S_F) out of the F-engine per second can be determined with Equation 2.4.

$$S_F = \frac{S_D}{2C_h} C_h = \frac{S_D}{2} \tag{2.4}$$

For Equation 2.4 let C_h be the number of channels and S_D the digitiser sampling rate. To determine the frequency by which the F-engine emits channelised data we need to divide the digitiser sampling rate by the number of samples needed to do the channelisation (twice the number of channels, $2C_h$). To get

the number of samples per second we will have to multiply this by the number of channels emitted each time. The number of channels then cancels out and we see that S_F is half S_D.

When the samples are sent to the X-engines adjacent channels are grouped. There will be as many groups as there are X-engines and there are four times more X-engines per correlator than receptors. Each grouping has a multicast address assigned to it. All the F-engines will send to all the multicast addresses. Each X-engine will only listen (or subscribe) to one multicast address. In this setup the network is used to transpose the data. The number of packets (P_F) emitted by each F-engine every second is a function of the number of X-engines in the subarray ($4A_n$), the cadence of the PFB is performed at ($\frac{S_D}{2C_h}$), the number of samples in a grouping ($\frac{C_h}{4A_n}$) and the number of samples in a packet (S_{fp}). The majority of the factors cancel out to produce Equation 2.5. This matches the expected result of having $\frac{S_F}{S_{fp}}$ packets per second.

$$P_F = 4A_n \frac{C_h}{4A_n} \frac{S_D}{2C_h} \frac{1}{S_{fp}} = \frac{S_D}{2S_{fp}} \tag{2.5}$$

The samples out of the F-engine are 8+8bit complex numbers[5] and each packet has a 40bit timestamp similar to the digitiser output. The size of S_{fp} is chosen so as to produce a data frame in the Ethernet packet that is smaller than the 9000bytes allowed for a Jumbo Frame. A satisfactory value for S_{fp} is 4096. As mentioned previously there are two F-engines per F-host. We calculate raw data rates out an F-host of $28\,\mathrm{Gb/s}$, $27.4\,\mathrm{Gb/s}$, and $17.4\,\mathrm{Gb/s}$ for S-band, L-band, and UHF-band respective. As with the D-engines we will use a single inflated value of $29\,\mathrm{Gb/s}$ as the output bandwidth of an F-host.

X-stage

The X-stage is the final stage of hard real-time signal processing and produces the primary output of the correlator. The X-engine performs a cross multiplication and accumulation of all the baselines (B_l). Each such baseline calculation has to be performed for each of the correlator channels (C_h), the F-stage will channelise the input streams into 1024, 4096, or 32768 channels.

[5] 8bits used for the real and 8bits used for the imaginary

The computational complexity of the X-stage increase dramatically with the increase of receptors in a subarray, the baselines grow exponentially with the increased of receptors. The computational complexity of the X-stage is $\mathcal{O}(C_h B_l)$.

The products of the baselines are summed (accumulated) over a period. This period is typically 4 to 8 seconds but could be as short as 0.5s. Although the number of data points generated by the cross multiplication is huge these values are accumulated, this results in a more manageable data rate out of the correlator. Accumulation is either expressed in seconds (T_{acc}) or as a frequency (f_{acc}), where $f_{acc} = \frac{1}{T_{acc}}$.

The number of samples out of the X-stage is a product of channelisation (C_h), accumulation frequency (f_{acc}) and the number of baselines(B_l). Equation 2.6 gives the number of samples per second (S_X) for an X-engine. Unexpectedly the digitiser sampling (S_D) rate plays no role in the correlator output, because of the accumulation.

$$S_X = \frac{C_h B_l f_{acc}}{4 A_n} \tag{2.6}$$

Although the aggregation stage hugely depreciates the data rate coming out of the correlator it introduces a new problem. The SKARAB compute nodes are all-time synchronised to a high degree, a few ns, they emit packets onto the network at exactly the same time. For example, when $T_{acc} = 4$, then every four seconds, all the X-engines will emit a series of packets. This can cause a buffer overflow on the switch ports. An artificial delay has been added to the output of each X-engine; but when the per port buffer-memory is under strain, packet drops do occur[6].

The samples emitted by the X-engine are 32+32bit complex numbers. Each X-engine is responsible for a section of the band and sends results to a multicast address. The X-engines output less than 40 Gb/s onto the network. Figure 2.6 shows the relative volume of data out of the D-hosts ($D_{1..8}$), F-hosts ($F_{1..8}$), and X-hosts ($X_{1..8}$) along with the ingest ($I_{1..4}$) nodes of the SDP.

[6]Some newer switches claim to solve this problem with a shared memory buffer architecture.

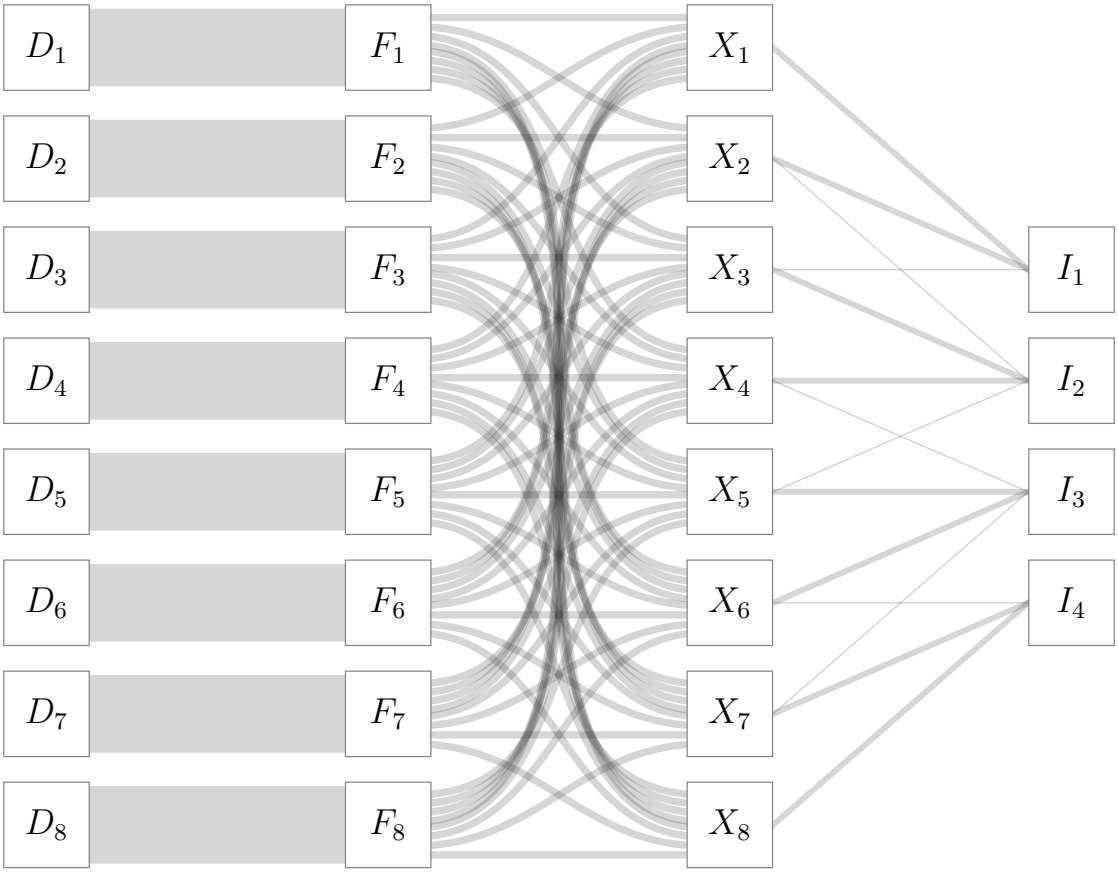


Figure 2.6: Data flow through the correlator, the thickness of the edges are relative to the data rate.

2.3.3 SKARAB

The SKARAB devices are the building blocks of the correlator. The same pool of SKARAB devices is used as F-hosts , X-hosts, and B-hosts. On initialisation of a subarray, some SKARAB devices are assigned to be F-hosts and some X-hosts for the subarray's correlator instance. For every receptor (A_n) in the subarray, there needs to be an F-host. Equal amounts of F-hosts and X-hosts are used per subarray. Thus every subarray uses $2A_n$ SKARAB devices.

Initially, the MeerKAT correlator was constructed with the second generation of the Reconfigurable Open Architecture Computing Hardware (ROACH) based processing devices but all of these were replaced with the more capable SKARAB devices.

The SKARAB board has a single Xilinx Virtex 7 Field-Programmable Gate Arrays (FPGA) and four $40\,\mathrm{Gb/s}$ interfaces, it is capable of running these at full-duplex. Only one of the $40\,\mathrm{Gb/s}$ interfaces will be used on the SKARAB.

The SKARAB devices are being developed in conjunction with the Collaboration for Astronomy Signal Processing and Electronics Research (CASPER) project.

2.3.4 Beamformer

MeerKAT will be equipped with several digitally steerable beams per subarray. One beam is always created in the X-stage since each X-host contains four X-engines and two B-engines. Two B-engines are required for one beam, one B-engine for the x-axis and one for the y-axis. At present MeerKAT only implements the single beam per subarray but when additional beams are produced a B-stage that is constructed out of B-hosts similarly to the X-stage will be started. The B-stage will also consume the same streams as the X-stage do. For this work, we are only concerned with the single beam produced from the X-stage. The bandwidth per beam is equivalent to the F-stage output.

2.4 Network

The MeerKAT network is comprised of several distinctly different areas that can be viewed as mostly independent networks. An OAM network that resembles a typical IT network that facilitates internet access, interoffice Wide Area Network (WAN) and other standard network services like Dynamic Host Configuration Protocol (DHCP) and Domain Name System (DNS). An Out-Of-Band (OOB) network that connects to the management interface of all the switches and Out-Of-Band Management (OOBM) controllers in the servers. As well as the data networks. For this study, we are only concerned with the biggest (by data volume, speed, and the number of nodes) of these data network, the correlator data network.

2.4.1 Data network

The correlator data network carries the data from the digitisers to the correlator and from the correlator to the soft real-time systems. The same data network links the correlator components to one another.

Because the digitisers, correlator and soft real-time systems are all on the data network many different data streams can be sent between the various systems, to a limited degree this is depicted in Figure 2.7.

The following data streams are common on the data network, but not limited to:

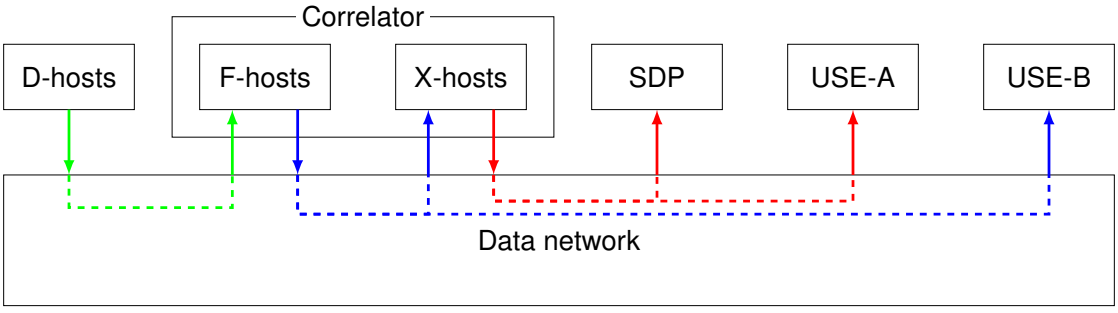


Figure 2.7: Block diagram showing the high-level data flow through the data network.

- Digitisers (D-engines) to the correlator (F-engines in).
- Correlator (X-engines and B-engines out) to soft real-time systems (*e.g.*, SDP).
- Between the correlator components (F-engines out to X-engines and B-engines in).
- Intermediate correlator products (F-engines out) to soft real-time systems.
- Engineering servers with large high-speed storage are connected to the data network and is capable of doing captures on any of the data streams. This is essential for debugging and verification.

The data network has a folded-Clos topology, sometimes also referred to as a Fat tree [26]. The network is made up of Leaf and Spine switches. The Spine switches are called middle stage switches in a traditional Clos network [27], these Spine switches only connect to Leaf switches. The Leaf switch performs either an egress or an ingress role in the traditional Clos network. But when the same Leaf switch fulfills both the egress and ingress roles we refer to the topology as folded-Clos. This folded adaptation to the traditional Clos topology is well suited to Ethernet-based networking where a link can asynchronously transmit and receive at the full data rate.

There is no direct link between Leaf switches and there is no interconnection between Spine switches. Figure 2.8 shows a small folded-Clos network with 8 Spine switches and 16 Leaf switches.

The MeerKAT data network is made up of 36 Leaf and 18 Spine switches, with $40\,\mathrm{Gb/s}$ connection between each Leaf and Spine switch. The same type of switch is used both for the Leaf and Spine nodes. Both Spine and Leaf switches are Mellanox SX1710 switches, the switch has 36 QSFP+ ports. Half the ports on the Leaf switches connect to equipment and the other half to the Spine switches. All 36 ports on the Spine switches are used to connect to leaf switches.

Leaf switches 1 to 4 connect to receptor switches. Each of these Leaf nodes has 16 receptors connected to it. Leaf switches 6 to 21 connect to SKARAB nodes. The remainder of the Leaf switches has the nodes from the soft-real time systems connected to them.

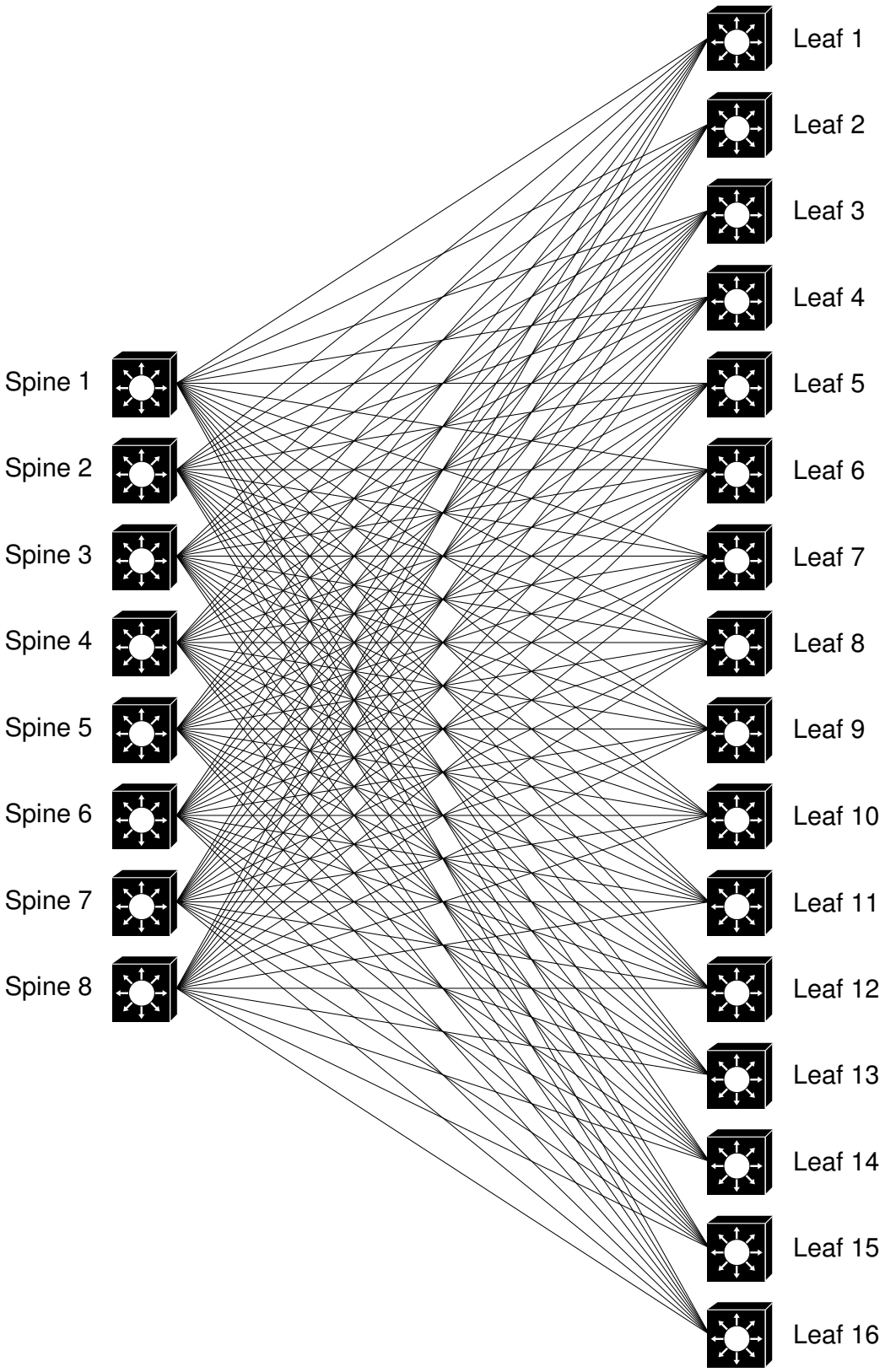


Figure 2.8: Folded-Clos architecture of the data network, showing links between Spine and Leaf switches.

(a) SX1012 (b) SX1710

Figure 2.9: Mellanox Ethernet switches.

2.4.2 Clos network topology

Manley [12] described the necessity of a full-crossbar non-blocking Ethernet switch for the MeerKAT correlator network. In his work he described the scaling considerations that led to choosing the Clos network topology [27]. This dissertation is looking at the feasibility of using SDN on the MeerKAT network and we will not evaluate other network topology or even evaluate the fitness of the Clos network for the application. To be able to correctly develop the SDN application a good understanding of the Clos network topology is necessary and how this topology is used to build the MeerKAT data network.

In 1953 Charles Clos published a seminal paper [27] on non-blocking switching networks. Although his paper was geared towards circuit-switched networks as used by the telephone industry of the time, the concepts of his work is transferable to packet switched networks. Typically in packet switched network the term folded-Clos is used to identify the architectural change from the original. In the original work and in typical circuit switched networks there are ingress switches and egress switches but in packet switched networks the ingress and egress layers are amalgamated, folded onto one another, with the resulting characteristic two layer design. A folded-Clos network has a layer of Spine switches and a layer of Leaf switches. All Leaf switches connect to all Spine switches. There are no Spine-to-Spine or Leaf-to-Leaf links in such a network. Access into the network is only via the Leaf switches.

Clos networks are described by three variables r, n, and m. Where r is the number of ingress and egress switches, there are r ingress switches and r egress switches. The number on ingress ports on the ingress switch is n and like wise, the number of egress ports on the egress switch will be n. Then m denote the number of middle stage switches. Since all edge switches (ingress and egress) connect to all m middle stage switches m ports are used on each

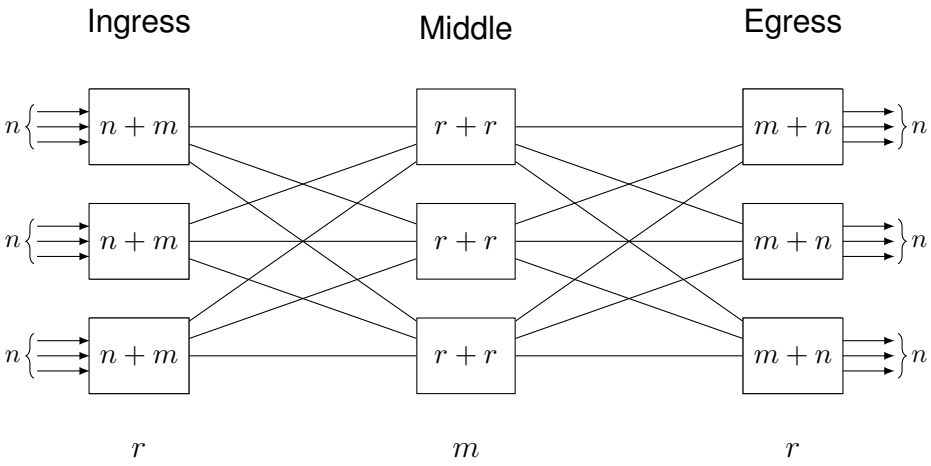


Figure 2.10: Clos network with ingress on the left and egress on the right, each node is a switch with a port number formula as an inscription.

edge switch for creating the network fabric, $2r$ number of ports are needed on each of the middle stage switches for this purpose.

Clos explained that when the values of $m >= n$ the network will be non blocking. To make the best use of resources and keep the equipment cost as low as possible we will make $m = n$, no unnecessary Spine-to-Leaf links. Such a network is drawn in Figure 2.10.

When the folded-Clos network is considered we see that there will be $2r$ Leaf switches each with $n + m$ ports. There have to be m Spine switches each with at least $2r$ ports. We can take this further with $m = n$ and insist that Spine and Leaf switches are of the same type (make and model) to simplify the network hardware procurement and management. Thus $2r$ Leaf switches with $2m$ ports each and r Spine switches with $2r$ ports each. By making $2m = 2r$ to force the same hardware for Spine and Leaf switches we get that there are m Spine switches with $2m$ ports and $2m$ Leaf switches with $2m$ ports needed to create an optimal non-blocking fabric. This results in a simplified statement, when the same switch model is used for both Spine and Leaf to produce a non-blocking fabric: Have as many Leaf switches as ports on a switch and half that many Spine switches.

In the MeerKAT data network, the Mellanox SN1710 switch with 36 ports is being used as both the Leaf and Spine switch. The network is comprised out of 54 ($2r + m = 36 + 18$) switches (Spine switch 1 to 18 and Leaf switch

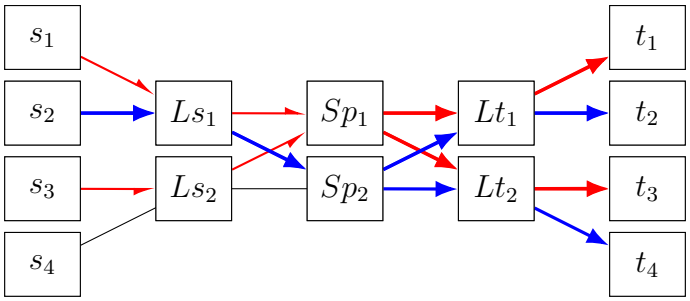


Figure 2.11: Multicast streams in a Clos network.

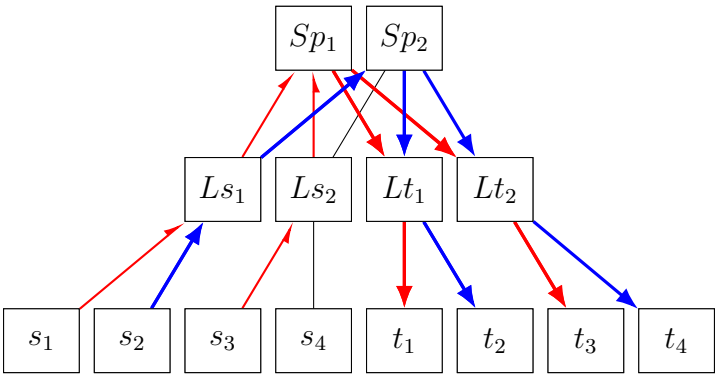


Figure 2.12: Multicast streams in a folded-Clos network.

1 to 36) and the resulting access port count is 648 ($2r \times n = 36 \times 18$) ports. Since each link is capable of 40 Gb/s bidirectional transfer the network has a 51.84 Tb/s throughput. The network is rearrangeably non-blocking [28] and full line rate transfers between any two nodes are *theoretically* possible at all times.

Multicast on Clos topology network example

A data stream from one device (source) to another (target) propagate through the Leaf switch the source is connected to, then through a Spine switch, then via the Leaf switch the target is connected to and finally to the target. Figure 2.11 show two example data streams (red and blue) through a simple network. The red streams from sources s_1 and s_3 are combined in Spine switch Sp_1 and sent to targets t_1 and t_3. The blue stream from s_2 is sent to two target, t_2 and t_4. Figure 2.12 show the same network and data streams but depicted as a folded-Clos topology; this is an improvement on the Clos topology since a device on the network can be a target for one stream but it is the source for another. In such a network, the selection of the Spine switch the stream traverses is our main concern.

2.5 Soft real-time signal processing

2.5.1 Science Data Processing

SDP is the final stage of the data processing in the telescope. SDP consumes the data streams out of the X-stage and performs a last round of processing. Within SDP the observation is stored and made available to scientists from around the world. The SDP is comprised out of many of its own components. A scheduler is used to start worker processes on different nodes depending on the processing requirements.

The ingest processes (I_1 to I_4 in Figure 2.6) are responsible for ingesting the correlator data and making it available to internal processes of the SDP. Up to four ingest nodes can be used per subarray, for this work we assume that the ingest processes are always on separate nodes. Each ingest process consume a quarter of the X-stage output but overlap is required to ensure accuracy. Thus each Ingest consumes a little more than a quarter of the data out of the X-stage. A 64 receptor ($A_n = 64$) subarray has 256 X-engines thus ingest I_1 and I_4 will consume $\frac{64+1}{256}$ of the total X-stage output each. I_2 and I_3 will each consume $\frac{64+2}{256}$ of the X-stage output.

Besides the X-engine output the SDP also ingests the B-engine output, with only one beam per subarray (generated as part of the X-stage) only one beamformer ingest node is required per subarray.

2.5.2 User Supplied Equipment

User Supplied Equipment (USE) is a very general term used for the equipment that will be supplied and operated by other science organisation, but hosted on-site in the KAPB. Typically these instruments will be several racks of high performance compute nodes that use data from the correlator and SDP and perform further specific calculations on the data.

It is possible that USE systems could consume data from the D-engines, F-engines, X-engines, or B-engines. But it is expected that the majority of USE systems will consume F-engine output. In Figure 2.7 two USE systems are

shown as an example of how they could consume data streams.

To incorporate the effect of USE systems on the network three USE systems were added to the models.

1. A 4 node pulsar timing system that consumes the B-engines output.
2. A system with 90 nodes that create its own beams from F-engine outputs.
3. A 128 node system that resembles what a Search for Extraterrestrial Intelligence (SETI) processing system would need. This system has F-stage output as input.

2.6 Software Defined Networking

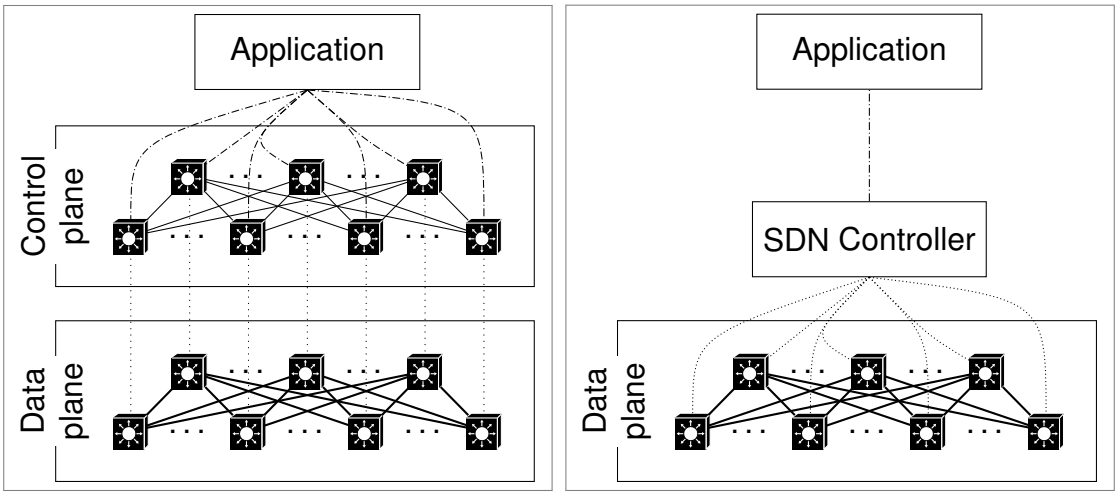
SDN is a concept where a software system controls and configures network devices that make up a specific network; unlike a traditional non-SDN network where each device is configured individually and the devices sometimes exchange information via cooperative protocols like OSPF or BGP.

With SDN the control function of the network devices is separated from the forwarding function. This increases the agility and flexibility of the network.

An SDN network has three layers to it [29]:

1. Application plane
2. Control plane
3. Data plane

From the bottom up, the data plane or sometimes called the infrastructure is responsible for the forwarding of the packets. This layer is the physical network devices. The control plane is the central control authority of the network, it is a software system that keeps track of the devices on the network, receives the network objectives from the application layer and communicate with the network devices to realise the network objectives. We refer to this software system as the SDN controller. The SDN controller exposes a northbound Application Program Interface (API) for the application and uses a southbound API to communicate with the network devices, OpenFlow is a popular Southbound API. The application plane is a software stack that understands the business require-



(a) Traditional network (b) SDN

Figure 2.13: Network control architectures.

ments and is able to convert these into instructions for the SDN controller. In many businesses, the application is custom developed to manage the network to best suit the business requirements. Traditional network control where each switch in the network participates in the control and data plane is represented in Figure 2.13a. The SDN approach is placed right alongside, as Figure 2.13b, to show that the data plane and application stay intact but the control logic is moved to a central controller, the SDN controller.

SDN should not be confused with configuration management and deployment tool that exists for network devices. These software tools merely help to assist network engineers to create (using templates) and deploy the configuration to network devices. SDN takes a more active role and the SDN controller continuously communicates with the network devices. SDN controllers could be consulted by a network device for every new (where no existing rule has been defined) packet that the network device has to deal with, the SDN controller then instructs the network device to forward, respond to, or drop the packet.

Chapter 3

System design

This chapter discusses the second and the third objectives of the dissertation. Namely, establish a flow placement framework (model) as a source of flow rules for an software-defined network and to demonstrate SDN as a multicast routing replacement (simulation) using the model results in a virtual network.

A brief description of the telescope system is given to bring attention to the network and multicast. Then the problem with the management of multicast routing is expressed. Followed by an explanation of the design approach and solution. We conclude with a summary of this chapter to highlight how an SDN based solution will simplify the network management and increase the versatility of the telescope.

3.1 Description of data distribution

A radio telescope is a distributed sensor network [11] that has to deal with large and many data streams. These data streams have to go through several signal processing stages to produce a product that can be stored and shared with scientists. Many compute nodes work together to perform the necessary processing. The data distribution and communication between the compute nodes calls for a high bandwidth network.

Multicast networking facilitates data distribution between source nodes and target nodes. It allows for efficient transmission from one source to many targets

and simplifies the scenario of transmission from many sources to a single target, as well as the combination of both.

3.1.1 Network topology

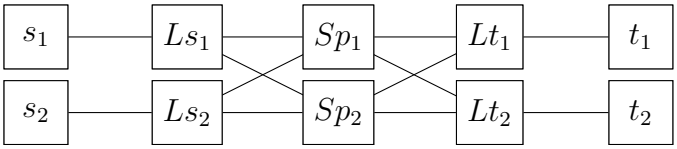
The MeerKAT data network has a folded-Clos topology. A non-blocking network is constructed using COTS network switches on this topology. Manley [12] explained the necessity of a full-crossbar non-blocking Ethernet switch for the MeerKAT correlator network. He described the scaling considerations that led to choosing the Clos network topology [27]. This dissertation looks at the attainability of using SDN on the MeerKAT network and do not consider other network topologies or even the tenability of folded-Clos as the network topology.

To correctly develop the SDN application, a good understanding of the Clos network topology is necessary, chiefly how the MeerKAT data network is built based on it. The data network and Clos network topology were described in §2.4.1 and §2.4.2 respectively. Each data stream (multicast stream) will traverse at least three switches, refer to the example in §2.4.2. A data stream will go through one Spine switch only; this Spine switch is also the RP for the stream. The RP Spine switch is responsible for the data plane stream delivery, the aggregation of source flows and the distribution of target flows.

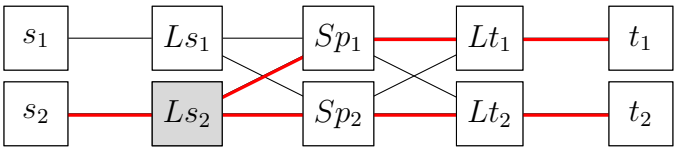
The selection of the Spine switch a stream traverses is vital to ensure that no links in the network become overloaded. The next section (§3.1.2) justifies the use of only Spine switches as RP. The selection of Spine switches as RP for multicast streams, to avoid overloading of network links, is the aim of the rest of the chapter.

3.1.2 Spine switch as Rendezvous Point

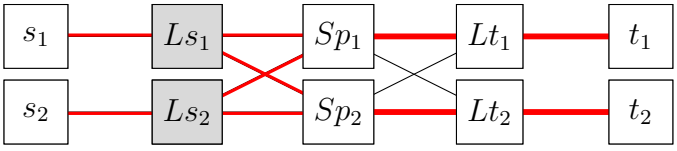
Considering both the *1-to-N* and the many-to-one (*N-to-1*) scenarios, it is clear that the easiest option is to have Spine switches (Sp_n) as the RP. In *1-to-N* the stream can be duplicated on either the source Leaf switch (Ls_n) or a Spine switch. In *N-to-1* the streams can be aggregated on either the target Leaf switch (Lt_n) or a Spine switch.



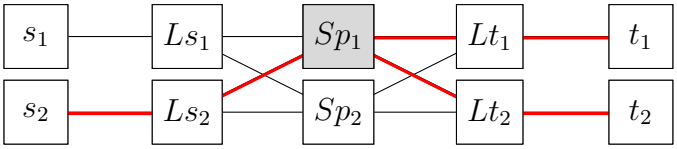
(a) Network topology, no active source.



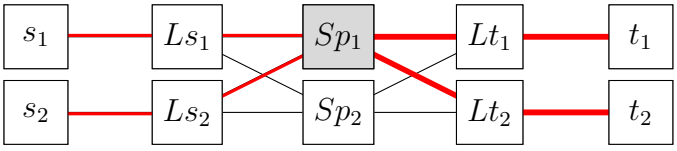
(b) One active source. Stream duplication on Leaf switch.



(c) Two active sources. Stream duplication on Leaf switches.



(d) One active source. Stream duplication on Spine switch.



(e) Two active sources. Stream duplication on Spine switch.

Figure 3.1: Data streams from sources to targets.

Under further investigation, it holds that the Spine switch is indeed the best place for the RP. Spine switches as RP yields the flow graph with the least number of edges. In the network under investigation all links are the same (40 Gb/s), thus we can take the number of edges in the flow graph as a cost factor, note that there is a different flow graph for each multicast stream in the network. The flow graph for a multicast stream can be given as $G_m = (V, E)$ with a set of sources $S \subset V$ and a set of targets $T \subset V$.

Figure 3.1a is a simplified network with two sources ($s_1 \in S$ and $s_2 \in S$) connected to different source Leaf switches (Ls_1 and Ls_2). This network has two targets ($t_1 \in T$ and $t_2 \in T$) connected to target Leaf switches (Lt_1 and Lt_2). Leaf switches are connected to Spine switches (Sp_1 and Sp_2).

Figure 3.1b and 3.1c show the network with traffic (thick red lines) from one and both sources respectively. In these two networks the RP was placed on the source Leaf switches. Figure 3.1d and 3.1e show the network with traffic from one and both sources respectively. In these two networks the RP was placed on a Spine switch (Sp_1).

With a single source, seven links were occupied when RP was placed on source Leaf switches (Figure 3.1b), and only six links were occupied when RP was placed on a Spine switch (Figure 3.1d). With two sources and two targets having RP on the source Leaf switches (Figure 3.1c) the cost (number of edges/links used) is 10 versus a cost of 8 when the RP was placed on a Spine switch (Figure 3.1e).

Equation 3.1 presents a way to calculate the cost factor of the flow graph G_m with RP on Leaf switches and equation 3.2 to calculate the cost factor when having the RP on a Spine switch.

$$k_L(G_m) = n(L_s) + n(L_s)n(L_t) + 2n(L_t) \tag{3.1}$$

$$k_S(G_m) = 2(n(L_s)n(L_t)) \tag{3.2}$$

The diagrams in Figure 3.1 and Equations 3.1 and 3.2 show that fewer links

are used in the network when the RP are located on the Spine switches. This also result in less and simpler flow rules. Thus we only consider Spine switches as RP in the current and SDN networks.

3.1.3 Data processing

In Chapter 2 a great deal of information was given regarding the interaction between components and the flow of data in a radio telescope, specifically MeerKAT, with a well-defined flow of data from D-engines to F-engines to X-engines to SDP ingest nodes. It is also important to again mention that Ethernet-based multicast allows for the consumption of data streams by several different systems simultaneously and redistribution of the resulting data stream *e.g.*, B-engines. This dissertation aims to make no changes to the data processing system, only to improve on the network environment. The proposed (improved) network shall stay compatible with the current data processing systems on all aspects.

The control of these data processing systems is through the CAM system. A scientist would request an observation via CAM. The CAM system is then responsible for instantiating and configuring the processing systems to perform the requested observation. CAM is also responsible for the deconfiguration of systems when the observation is complete.

3.2 Problem description

3.2.1 Multicast networking

Multicast networking is used for data distribution, which results in a very effective data distribution system. Each source device only sends out one copy of a data stream even if multiple targets require the data stream. With multicast, when multiple targets subscribe to a data stream, network switches will send the stream to these targets. It is even feasible to have multiple sources and targets for the same stream if the data exchange protocol supports this. This is the case with SPEAD [25] a User Datagram Protocol (UDP) data exchange protocol used in astronomy.

Multicast streams are convenient to work with since each stream has an IP address. By giving the multicast stream an address, the data stream is addressable, and there is no need to distribute a register of source and target addresses. CAM issues preconfigure multicast IP addresses to the processing systems; this happens without any knowledge of the active network. RP placement is dependant on the value of the multicast streams IP address. A multicast IP address is not only a convenient reference to the data stream of a topic; they influence the runtime network configuration and can have a dramatic impact on the distribution of data streams on the network. If the stride between addresses is too big or too small, there is a risk of one or more Spine switches having to forward more traffic than what it can handle, leading to packet loss.

In a multicast network sources are unaware of targets, and targets are unaware of sources, they are loosely coupled. Although convenient, this does create a possibility for a target to miss the initial portion of a data stream. Since there is no synchronisation between source and targets, the source can send data even though the target is not ready, it could be that the target is ready but that the network has not fully converged on a route for the particular data stream. In a stream processing system like a radio telescope, this is inevitable, and components are designed to handle this situation to a certain extent. Even though the components of the telescope system can handle missing data, we do want to ensure that the network is ready as soon as possible and before data transmission is started. Currently, there is no way for the telescope control system, the CAM system, to influence or inspect the data network. The control system has no indication of the network state and cannot inform the network of the telescope's intent.

Multicast routing protocols depend on the L3 routing protocols (*e.g.*, OSPF) for network topology information [30, 31]. Multicast routing and L3 routing are set up and maintained by the network engineers. These protocols are complex to master and even a small change in the network requires a good deal of planning and often even more effort to debug since the behaviour is difficult to predict. Folded-Clos network topology is a very dense overlapping cyclic graph. Multicast routing protocols need to determine which RP will be responsible for a multicast stream. The multicast routing protocols keep the source tree and target tree in memory to be able to forward traffic. In a network with many data streams, there may not be sufficient amounts of memory on the

network switches to accommodate all the multicast streams required by the different operating modes of the telescope. For memory utilisation and better RP placement, the switch Original Equipment Manufacturer (OEM) of the MeerKAT data network switches implemented a unique hashing scheme[1]. Thus all the switches in the network have to run custom firmware, provided by the OEM. Even though the OEM provides excellent equipment and service, it is a risk for MeerKAT since there is currently only one switch OEM with the specialised implementation.

3.2.2 Network management

Network engineers adjust switch settings like the internal loopback IP address of each switch to influence the placement of RP and have several cost variables to play with to influence OSPF and PIM.

Multicast stream IP addresses have to be carefully assigned at subarray creation. This is to ensure that the streams are spread over the Spine switches and do not cause any of the switch links to be oversubscribed. Unfortunately, the allocation of IP addresses to multicast streams are unaware of the state of the network and has no knowledge of the current network utilisation. Thus regardless of the network condition or load, the same address combination is always used for a subarray with a certain size (A_n). This can lead to situations, based on the configuration of the other subarrays, where some Spine switches are overloaded although the network as a whole is underutilised. In these cases, it can result in data loss.

Artificial constraints on the subarray combinations that are allowed have been introduced to restrain the possible combinations, which limits the full potential of the telescope. Over time, network engineers have tuned the switch configuration, multicast IP address allocation process and the artificial constraints to a point where the stream placement for the common modes of operation yields a well load-balanced network. However, any change to the network, adding devices, removing devices, or a change to the key science observations will require manual intervention to ensure the network operates correctly. Manual creation of acceptable load distribution is a very time-consuming and

[1]The hashing changed is done in the implementation of Bidirectional Protocol Independent Multicast (BIDIR-PIM) [32].

complex task. To reiterate the placement of data streams on the network in the current system is good, well balanced across all the Spine switches. However, the problem is that this is very difficult to do, is time-consuming, is limited to the most common subarray combinations and is not transferable in whole if the network is expanded or upgraded.

If a Spine switch were to fail, it is crucial to promptly replace the device even though there is still enough total capacity over the remainder of the Spine switches. OSPF and PIM can dynamically handle some failure conditions but do not always yield a favourable result without manual intervention. It is worth noting that the failure of a Leaf switch is still critical regardless of the control plane improvement.

3.2.3 Summary

In summary, the main drawbacks of the current data network that this dissertation aims to resolve are:

- Complex configuration
- Static behaviour, unable to automatically react to physical change *e.g.*, failures, upgrade
- Vendor-specific implementation
- Require high-end COTS switches that are capable of L3 routing and multicast routing
- Poor or unbalanced utilisation of Spine switches, if not manually adjusted
- No control system (CAM) integration
- Network behaviour is influenced by the multicast address choice

These drawbacks result in the following negative consequences on the telescope:

- Complexity - difficult to document, manage and change
- Slow to change and adapt to growing system needs
- Costly
- Limited and slow skills transfer between network engineers
- Poor redundancy
- Data loss as a result of link saturation

- Vendor lock-in
- Constraints on the combinations of simultaneous observations
- Lack of observability

3.3 Design approach

After studying (Chapter 2) the MeerKAT data network and the processing components, it became clear that a better method of performing traffic engineering on the data network is required. For this, the routing of the multicast streams needs to be improved. Since a multicast stream consists of multiple flows, the placement of flows by writing flow rules to all switches involved can replace multicast routing. SDN and in particular OpenFlow is well suited to the task of writing flow rules to network switches. Once an SDN controller has been set up correctly, it is very easy to distribute custom flow rules through the network.

There are many discussions [33, 14, 34] and tutorials on the set up of multicast networking with SDN for high volume video streaming, it is most often for a single source multi-target scenario with, understandably, fan-out being pushed as far out to the network edge as possible. The other widespread use of multicast in data centres is the use of it for timing, *i.e.*, Precision Time Protocol (PTP) and for metrics propagation. In both of these cases, the usage is low volume and straightforward *1-to-N* and *N-to-1*, respectively. Radio telescopes are unique in their signal processing requirements at scale. In nearly every case the radio telescope is a once-off build, with new or different technologies compared to previous telescopes. Therefore there is not much existing reference material to go by for solving the mentioned problems. But with high-speed Ethernet becoming a very affordable medium for high volume low latency applications, we believe future radio telescopes will use Ethernet packet switching networks as the primary data distribution medium, and if we can overcome the complexity of multicast network routing then this too would be the predominant method.

We know we can represent a multicast stream as several flow rules on switches and that programming these rules on switches can be taken care of by an SDN controller. We can focus our attention on unpacking these multicast streams into flows and the calculation of flow placement. Each multicast stream is a directed acyclic graph; each edge has a bandwidth requirement as-

sociated with it. Single source multicast streams will have the same bandwidth requirement for all edges. We overlay the multicast stream graphs over the network topology graph. The bandwidth requirement of each overlapping edge is summed and compared to the bandwidth made available by the network for that edge. The objective is to ensure that the sum is never more than the available bandwidth.

3.3.1 Rationale

The most complex and constraining component of the data distribution network is the multicast routing. By replacing the multicast routing implementation with a flexible and manageable system, it would simplify the administration of the data network. It enables the removal of artificial constraints and opens up network upgrade possibilities.

Multicast compatibility needs to be maintained to avoid redevelopment of processing systems and tools already widely used. Multicast itself is a good fit, but the associated routing protocols are not sufficient.

3.3.2 Methodology process

The research work loosely followed the spiral model [35] of Software Development Life Cycle (SDLC). With the following key stages Analysis, Planning, Design, Development, Deployment, and Testing. With every iteration the understanding of the system improved and also the solution.

3.4 Solution design

3.4.1 System description

SDN is based on a central system that manages switches in a network. This central system, the SDN controller, has a complete view of the network and can be fed additional information to allow it to configure network devices to best suit the objective of the overall system. Moving from distributed logic to a single process significantly simplifies the implementation of the control logic, making

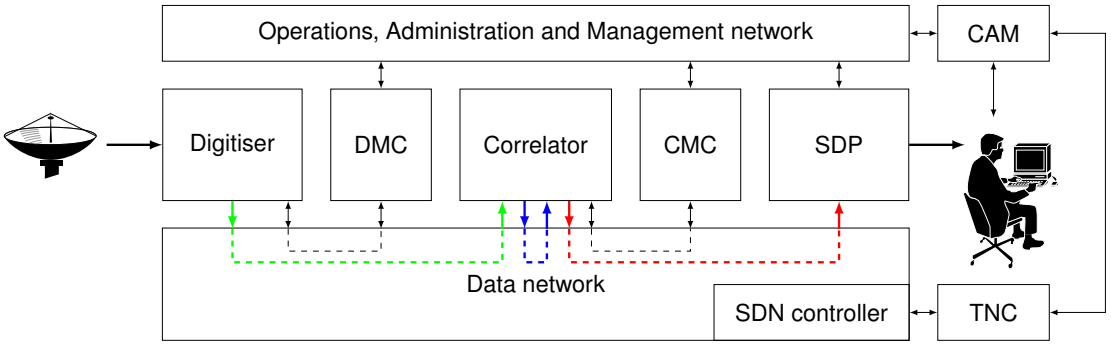


Figure 3.2: Interactions via the data network of Telescope network controller and SDN controller along with major subsystems.

the development of alternative control logic for a network more accessible. The move to a single control process speeds up the decision-making of the control logic. Firstly, by eliminating the communication between the components. Secondly, the SDN controller can run on a standard server that has significantly better processing capabilities at a lower cost compared to a network switch. The solution will use an SDN controller and use OpenFlow when managing the switches in the network.

When we look at multicast routing, we observe it as a complex distributed computing and coordination problem. The result of multicast routing is a graph with traffic flowing from sources to targets. The complete data flow of the data network is a big acyclic directed graph. However, it makes more sense to think of it as many individual acyclic directed graphs where each such graph is one of the multicast streams. The graph or graphs are all constrained by a large ($n = 18$, $m = 36$) complete-bipartite graph[2] representing the physical network. It would be a daunting task solving route placement with these graphs, for any meaning of optimal. When breaking down the graph by subarray and accepting to serialise the route solving per stream, it becomes much less challenging and requires very little graph algebra. For each multicast stream, one can determine the effect on the network for each possible RP on Spine switch combination.

An ideal solution would involve the telescope control system (CAM) informing the network about a newly requested observation. For this, the CAM system will have to provide all the flow rules necessary to the SDN controller. Since CAM is a telescope control system it has no concept of networks, bandwidth,

[2]The bipartite graph $k_{n,m}$ is the graph $G = (U, V, E)$ where $U \in \{S1 \ldots S18\}$ is all the Spine switches and $V \in \{L1 \ldots L36\}$ is the Leaf switches.

and for that matter SDN flow rules. It is necessary to have an intermediary system to translate the telescope intent into instructions for the SDN controller. We will call this intermediary system the Telescope Network Controller (TNC). Figure 3.3 is a block diagram showing the interaction with TNC, following instructions from CAM to network switches. Figure 3.2 is an updated version of Figure 2.1 from Chapter 2 with the SDN controller and TNC added to show how these fit within the telescope.

CAM will send a request to TNC for a new network configuration on set-up and tear down of a subarray. CAM will provide it with the necessary information *e.g.*, reference name, the receptors involved, target soft-real time nodes, facts necessary to calculate bandwidth requirements. Brokering facts between subsystems is what the CAM system do. In the case the request could not be fulfilled, TNC will inform CAM of this failure and provide diagnostics information. This type of feedback will allow the operations team of the telescope to find faults early, not hours into an observation. This is not to replace network monitoring systems.

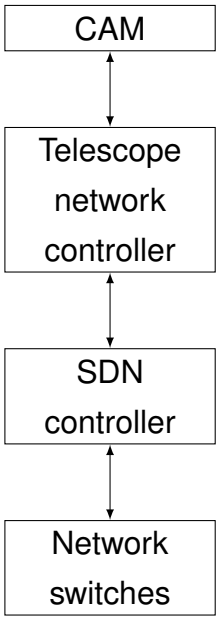


Figure 3.3: Interactions with telescope network controller.

The TNC is the outcome of this dissertation; we will not make many references to the TNC as a whole but mostly focus on the components of the TNC. The following components, as shown in Figure 3.4, make up the TNC: model, database, SDN application, and initialisation tools.

SDN controllers by themselves are only suitable for handling communications between applications (northbound API) and network equipment (southbound API). An SDN controler requires that the intended network configuration is provided via the northbound API or the SDN controller can be extended by running application logic within the controller, this is called an SDN application. From the information provided and knowledge of the network, the SDN controller can manage flow rules on all the switches in the network.

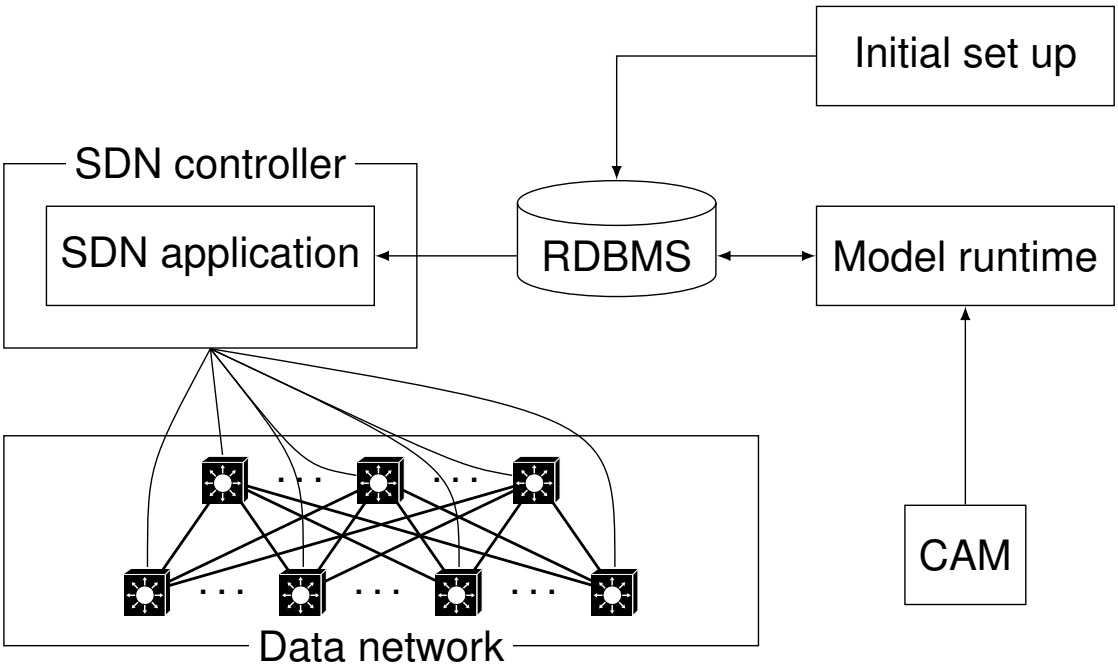


Figure 3.4: Abstract system architecture.

3.4.2 Functional system architecture

In a network without an SDN controller all the network switches are configured individually, either by an administrator that Secure Shell (SSH) into the device and typing commands or by an automation system. An automation system creates a new configuration from template files and predefined variables; thereafter the automation system writes the configuration to each switch.

An SDN controller takes a more active role, and it will maintain the flow tables in a switch with a handful of Create, Read, Update, Delete (CRUD) like methods. The SDN controller can continuously update the flow tables, and there is typically no need to have a maintenance window for switch configuration. The SDN controller only manipulates flow tables. It has access to a different selection of actions than what one has through a standard switch Command Line Interface (CLI). Typically SDN controllers don't have access to high-level functions, for example, OSPF configuration commands.

Since the TNC is the go-between between CAM and the SDN controller, it is responsible for the translation of CAM requests to flow rules. To perform this translation, the TNC must have some telescope specific business logic. It has to be aware of the dataflows in the telescope and how the scientific require-

ments on the telescope relate to bandwidth requirements on the network. The TNC has to understand the network as a whole and how the network topology and processing components influence one another. It is also the responsibility of the TNC to produce flow rules that will not exceed the capabilities of the network equipment. To be able to fulfil theses functions, the TNC require initial information about the telescope, network, and processing components. To store the initial information and for internal use, there has to be a persistent data store within the TNC. Lastly, the TNC needs to communicate with the SDN Controller.

The model creates the flow rules required by the SDN controller based on the telescope configuration as provided by CAM. The model contains the logic that manages the telescope to network interactions. A Relational DataBase Management System (RDBMS) provides persistent storage; all the TNC components interact with this database. Initialisation scripts set up the database and load site-specific parameters in the database. The SDN application extends the SDN controller with new functionality and gives it the capability to connect to the database and read flow rules from there.

Figure 3.5 is a Unified Modeling Language (UML) representation of the solution described. The solution uses an embedded application to translate the rules in the database for the SDN controller. A standalone software component, the model, inserted the flow rules into the database based on information already in the database (network topology, existing flows, available devices) and the telescope intent provided by CAM.

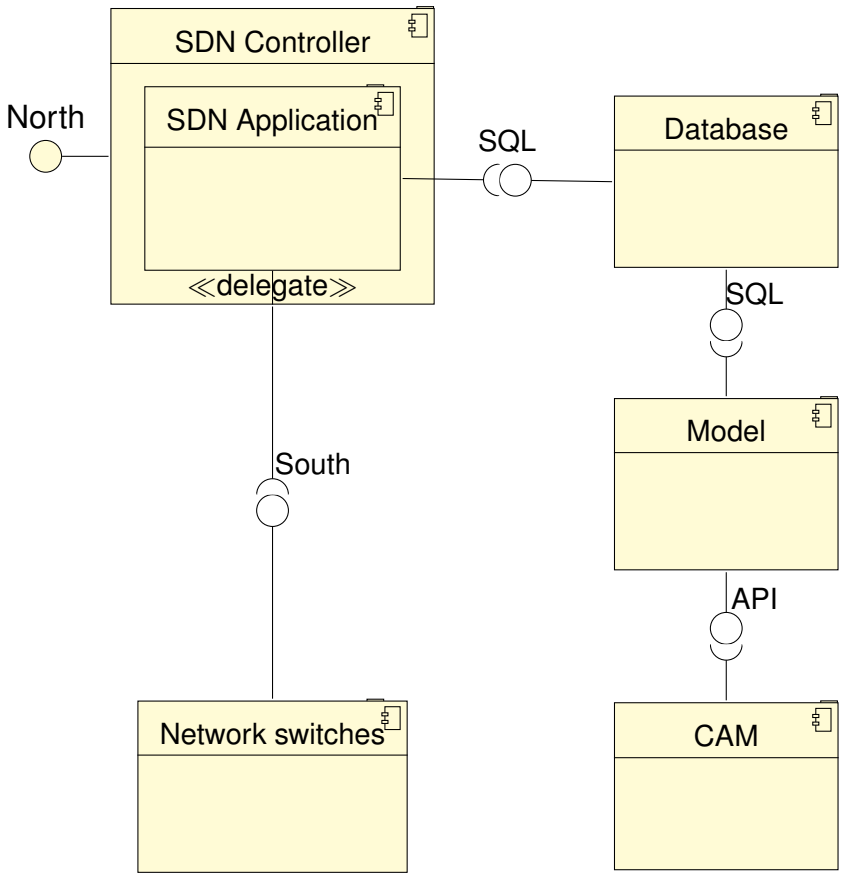


Figure 3.5: UML component diagram of TNC. Including CAM and Network switches.

3.4.3 SDN application

In our implementation, the SDN controller was extended by embedding an application into the SDN controller. Embedding is preferred, but using the northbound API would have worked. In brief, the advantage of embedding was that it reduced the number of interfaces that needed to be developed and made debugging much simpler since application and controller were in the same environment. The embedded SDN application built for the proof of concept is straightforward, its function is only to synchronise flow rules in the database with the controller. Production SDN application will need to manage more than only the multicast routing; the data network also carries traffic for monitoring, debugging and control of the processing components.

The model notifies, via database, the SDN application when a subarray has changed. Both when the subarray is created and torn down. The SDN application and the SDN controller programs the flow rules to the network switches.

The SDN application uses a Python library provided by the SDN controller for developing applications and the SDN application runs within the SDN controller. The SDN application connects to the database on startup and stays connected. The database is used for asynchronous signalling between SDN application and the model *e.g.*, notifications of rule changes. A new notification triggers the SDN application to read flow rules from the database and apply these to the applicable switches. The SDN application runs in a limited environment. Writing highly parallel or asynchronous code in this environment would be difficult. For this reason and in keeping with software engineering best practices of separation of concern, the creation of flow rules happens in the model.

The sequence by which the switches are programmed is important. Each stream involves at least three switches: one or more source Leaf switches, a Spine switch and one or more target Leaf switches. Target Leaf switches and the Spine switch should have their rules programmed before the source Leaf switches are programmed. Ideally, devices should only start sending after the network is set up. Setting the order that switches are programmed is very easy and helps in preventing a flood of packet-routing-requests to the SDN controller. Disabling the network learning capabilities on the switches is also advised in a static high throughput network.

3.4.4 Initialisation scripts

The network topology in the database should contain all the devices (nodes and switches) in the network and all active network interfaces on these devices, along with the links between devices. It is essential to have an accurate map of the connected devices. Furthermore, the bandwidth the links are capable of need to be in the database.

The SDN controller can learn the network topology, or an external system can provide it. For smaller networks or networks that change frequently, learning is an ideal solution, but on a bigger network and especially networks with high data volumes, learning can create a scaling challenge since packets arriving at a switch that has no predefined route is forwarded to the SDN controller. A radio telescope has a static network topology; the addition and removal of devices are infrequent and under tight control. The initialisation-scripts maintain the network topology and device information in the database. An improvement on this would be to synchronise the database with a Data Center Infrastructure Management (DCIM) system that also performs Internet Protocol Address Management (IPAM).

3.4.5 Model design

To determine the best placement for each of the data streams, a system was designed to model the data streams within the network. This model is configuration driven and changes can be made in the network topology, device functions, bandwidth allocation per function etc.

Modelling components use the database to exchange information about the telescope objective and network. The individual links of the data stream (edges of stream graph) are stored in the database as distinct entries. Each of these entries represents a flow rule to be programmed to a switch. The SDN application reads desired flow rules from the database when switches need to be programmed.

To create the flow rules for a subarray of a specific configuration the following is necessary in the database:

- General network information
  - Network topology - Leaf switch to Spine switch connections
  - Node connection info - device to Leaf switch connections
  - Link bandwidth
  - Current link utilisation
- Per stream information
  - Source nodes
  - Target nodes
  - Link bandwidth required
  - Multicast address

CAM will inform the model when a subarray is created or torn down. Given the subarray configuration, it is possible to determine all the multicast streams that are necessary for the subarray. The subarray configuration provided by CAM will contain the receptors (source) and the target soft real-time systems. The model assigns the correlator compute-components to functions[3]. The model establishes the sources and targets of all the multicast streams in the subarray. The multicast streams with sources and targets are stored in the database. Next, the bandwidth usage of the multicast stream has to be determined, the source egress bandwidth, and the target ingress bandwidth. In the proof of concept implementation, only the worst-case data rate per stream is used. Using the worst-case rates per stream is adequate for the current way MeerKAT operates and suitable for a proof of concept. Using exact data rates is possible to implement, and all necessary formulas with explanations are given in Chapter 2.

The critical step is to determine the path the multicast stream will follow from sources to targets through the network. The placement algorithm performs this step. The placement algorithm works out the Spine switch that will be the RP for the stream and attempts not to overload any link in the network. To showcase the possibilities, we present two placement algorithms. The first placement algorithm can attempt to balance the total bandwidth usage of each Spine switch (load balancing). The second placement algorithm uses a packing technique that will attempt to maximise the utilisation of the first Spine switch to a preconfigured peak value before attempting to utilise the next switch (bin packing). The load balancing method naturally comes to mind and is an obvious

[3] The assignment of compute nodes to functions is probably best done by a system dedicated to such a task. With the current homogeneous pool of compute nodes, it is still reasonable for the model to perform this task.

first choice. The load balancing method is favoured under most circumstances, since all the Spine switches work equally hard, but has a hard time solving when the required throughput is very close to the theoretical maximum throughput of the network. In such a case, the bin packing method is more likely to solve. The placement algorithms are described in more detail in Appendix A. The placement algorithm should be seen as part of the characteristics of the telescope and will evolve with the telescope.

The placement algorithm starts with the multicast stream with the highest bandwidth requirement and works its way down to the stream with the smallest bandwidth requirement. The multicast identifier (IP address) is used as the arbitrator. To demonstrate the advantage of sorting the streams by bandwidth see the examples in Section §A.4.3.

Both the algorithms starts by selecting the first Spine switch as the RP and updates the database. Much like PIM, it builds up a graph from sources to RP and from targets to RP, the graphs are combined. The network topology constrains this graph. The bandwidth usage is calculated for every edge of the graph and written to the database. The utilisation is calculated for all the links/interface on the chosen Spine switch to establish if the new flows have caused a link on the Spine switch to exceed the maximum data rate. If a link on the switch exceeds the configured peak data rate, then all the entries made into the database for this multicast stream is reverted, and the process will attempt to use the next Spine switch. If the interface utilisation of the switch is satisfactory, the process moves onto the next multicast stream. The key difference between the load balancing and bin packing happens at the point. With the next multicast stream, the bin packing method will start with the first Spine switch, whereas the load balancing method will move to the next Spine switch similar to when the stream could not be placed.

Once all the Spine switches have been considered. The load balancing process will try all the Spine switches once more and will raise an error if no route is found. The bin packing process will raise an error after the last Spine switch were evaluated. If an error state is reached, the model will stop and inform CAM that the requested subarray is not possible.

For demonstration purposes, the proof of concept model continues even under error conditions. The model uses the last tried switch for the multicast

stream and carry on with the other streams even though this resulted in overloaded links in the system. This was done to create plots[4] that showed the required aggregated bandwidth along with the overloaded links. This technique helped to highlight coding errors and the boundaries of the algorithm capabilities.

3.4.6 Database

The database is the only component in the SDN solution that holds a permanent state. All the other components can be stopped and started as needed, naturally, if these components were busy with a calculation or update these would be interrupted. The database is also the bridge between the SDN application and the model. The model will work through all the flows for a new subarray and only once complete will it signal the SDN application that it can apply the new flow rules.

An RDBMS was chosen for several reasons.

Transactions By grouping operations into Atomicity, Consistency, Isolation, Durability (ACID) transactions, we can evaluate multiple subarrays simultaneously. The models will not interfere with one another, but they will be aware of each other's updates.

Query language Structured Query Language (SQL) is a powerful query language and it allows for large blocks of the model logic to be written in a simpler more concise manner and then to be executed within the database.

Relational Tabular structure with foreign keys is a well-understood method for modelling directed graphs. Although not as elegant as a dedicated graph database it is sufficient for this application.

Integration With a database, there is a plethora of tools and libraries to integrate the data it holds. If the route planning was performed in application memory it would have been necessary to develop exporters and a new set of tools.

[4] This can be seen in the line and heatmap plots in Chapter 5.

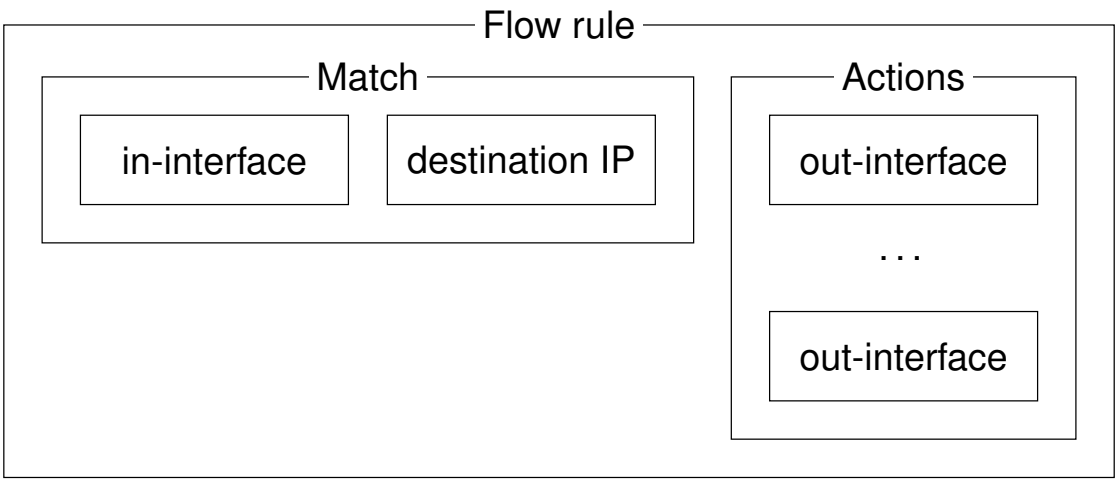


Figure 3.6: OpenFlow command construction.

3.4.7 Flow rule construction

The flow rule will be described from the perspective of the packet. The rules engine needs only the switch identifier and the destination address for multicast streams. This information is used to lookup which interfaces on the specific switch the packet should be sent out on to. This list is filtered to ensure the source interface and interfaces disabled by STP are not on the list. From this list, a list of OpenFlow output actions are created for every unique interface that packets would be sent out for this flow, this is the actions list. An OpenFlow match condition is created with the source interface and the destination IP address (the multicast stream address) as parameters to be matched. The new flow rule (Figure 3.6) is written to the switch with this match condition and action list. Thus when a packet reaches the switch, it searches for a rule with the same source port and destination IP address. If such a rule is found, all the actions in the action list are performed. The actions are to send the packet out on multiple interfaces. For aggregation, the rules are duplicated for the multiple source ports. The previous mention structure is followed since the number of stream aggregations are constant (F-engine to X-engine) but duplication is more variable since any number of soft real-time systems can join the subarray.

The rule is extremely simple and does not rely upon groups or pipe-lining. For MeerKAT there will be no more than 35000 rules per Spine switch. Neither groups nor pipe-lining was heavily investigated since there is little pressure on the switches Ternary Content-Addressable Memory (TCAM) memory with such

few and simple rules.

3.5 Simulation

The controller functionality is verified by simulating a representative network. This simulated environment allows for the testing of the modelling results and served as validation of the model. The simulation makes use of software-switches and virtual nodes running on the same computer.

In preparation for a simulation, the database gets updated with the desired network topology; using the init scripts. The simulation reads the topology from the database, in the same manner as the model, using this information soft-switches and virtual nodes are created and instantiated. The switches and nodes are connected, these links appear as network interfaces on the host operating system. The simulation environment makes it possible to login to the nodes and run standard Linux commands or install additional software. The links can be monitored and inspected with standard Linux networking commands.

Once the simulation environment is running the workflow is as follows:

1. Run the model with the subarray configuration of interest
2. Trigger the SDN application to read from the database and update switches via SDN controller
3. Select a multicast stream to test
4. Enter into the source nodes and start a sending script with the multicast address
5. Enter into the target nodes, join the stream and verify that the data is being received, and check the correctness
6. Enter into the remaining nodes, join the stream and confirm that no data is being received
7. Observe the network interfaces, from the host, to ensure traffic pattern is as expected
8. Stop sending processes on the source nodes
9. Select another multicast stream and continue with testing

Several multicast streams can be tested at the same time bearing in mind

the resource utilisation of the host computer. By lowering the number of packets per second and to a lesser extent, the packet size more streams can be tested at the same time. Testing too many[5] streams simultaneously results in noticeable jitter and packet drop.

Although a simulated network is not capable of testing at telescope data rates, it is a useful mechanism to verify the programming of the network switches. The topology of the simulated network is a smaller folded-Clos topology. Attached to each Leaf switch is at least two nodes.

Tests were combinations of the following scenarios:

- One of the nodes connected to a Leaf switch is sending
- One of the nodes connected to a Leaf switch is receiving
- Sending and receiving nodes on a Leaf switch (same and different streams)
- Nodes connected to a Leaf switch is receiving (same and different streams)
- Nodes connected to a Leaf switch is sending (same and different streams)

While evaluating combinations of the scenarios, the simulated network was interrogated to ensure correct operation. Some of the checks performed were:

- Multicast streams successfully flow through the simulated network and reach only the intended target nodes
- Packets do not propagate past the first Leaf switch when sending from an incorrect source node
- The bandwidth reported by the sending processes matched that of the receivers and is correctly reported by the network devices

3.6 Summary

With the proposed SDN approach the data processing components are kept intact. The data processing systems do not require any changes since the network has the same interfaces and capabilities. By replacing L3 and multicast routing protocols on the network switches with an SDN controller, the data flow in the network is controlled in a more precise manner. Allowing for better observ-

[5] Too many is very subjective and depends on the computer the simulation runs on. Under stress conditions, other applications can also influence the tests.

ability, fine control and data propagation structures not possible with currently available protocols.

For the SDN controller to be aware of the telescope intent the CAM system will have to instruct the TNC when a subarray is created and destroyed. On subarray creation CAM will send the subarray configuration to TNC and request the network to be configured to best match the configuration. Such a request will result in the network switches being programmed with an updated set of flow tables.

Within the TNC a model is run to determine the most suitable path for the multicast streams. The model utilises a database for persistent storage and data sharing with the SDN controller. The network topology and compute node information is made available to the model via the database. Compared to a non-SDN network an SDN telescope data network has the following advantages:

- Simplified configuration, the configuration is managed from a single point
- Switch configuration is dynamically updated by the SDN controller
- In the case of physical failures, the model will generate a new set of flow rules
- OpenFlow allows for the implementation to be vendor agnostic
- COTS switches do not have to be high-end L3 switches with multicast routing, switches still have to be high performance switches
- Utilisation of the Spine switches can be managed
- CAM system influence the network configuration and receive network status from the SDN controller
- Multicast addresses have no significance on data flow, they only have to be unique with in the network to identify the streams

To illustrate the possibilities of an SDN managed network two different placement algorithms is presented, Appendix A. This was not done to select a superior algorithm, but serve to showcase the flexibility and power of an SDN controlled network. Prior to SDN traffic engineering was limited to the behaviour of the routing and switching capabilities of the network hardware. It was not easy and in many cases, not possible to make fundamental changes to how traffic flow through an extensive network. Allowing an SDN controller to centrally manage the data flow in a network, while being orchestrated from the

application, the network can better support the application. Resulting in better application performance or a more cost-effective solution for an organisation, and in many cases both.

Chapter 4

Implementation

The previous chapters explained the problems and problem space. In this chapter the implementation of a system to evaluate the suitability of SDN in a radio telescope data network is explored.

To determine if SDN is suitable for use as the control mechanism for a radio telescopes data network (§2.4), it is necessary to determine the optimal placement of data flows through the network. A model was developed from the ground up to achieve this within the constraints of the MeerKAT radio telescope. The model takes in network topology and telescope configuration and determine a suitable path through the network for each multicast stream to make the best use of the available network bandwidth.

A small scale verification-simulations were performed to vet the capabilities of SDN flow-table rules in the network under test.

Modelling and simulation are two separate systems and each described separately. They are loosely coupled, simulation requires the model results, and exchange information via a RDBMS.

4.1 Modelling

Modelling performed in this research aims to represent the full data network topology with all the multicast streams (each with multiple flows). Along with

accuracy, the modelling system aims to be scalable, flexible and easily extendable.

4.1.1 Python

The modelling system was written in the Python programming language. Python is a versatile programming language and has a large community and an excellent selection of modules. Python is an interpreted language and programs do not require compilation before execution, this saves a considerable amount of time when doing exploratory work with a small run time penalty.

Python version information

Python version 3.6.8 were used for all applications in this research except for the Ryu (§4.2.2) controller where Python 2 were used. All Python code were executed from within an Ubuntu 18.04 Docker image.

4.1.2 PostgreSQL

PostgreSQL, a powerful open-source RDBMS, was used as the database. The database stored the network topology and the flow rules created by the model. The model and the SDN application store all their intermediate and state information in the database. Many of the processing steps were performed within the database using SQL queries.

Sharing information via the database allow the different components to exist as independent applications; each application would read from the database only what it requires and write back the updated or inserts new entries. This abstraction enabled fast prototyping and allowed for the separation of concern within the modelling framework.

The majority of SQL queries used are standard and easily transferable to other RDBMS. Four features used in the implementation is either specific to PostgreSQL or differs to other RDBMS:

- IP addresses were stored as the *inet* data type, this is a PostgreSQL native

datatype for IPv4 and Internet Protocol version 6 (IPv6) addresses.

- Extensive use was made of Command Table Expression (CTE), the `WITH` statement in SQL, for adding and updating records.
- The use of nested `SELECT` statements avoids unnecessary data transfer between the model and the database. Also known as subqueries or inner select.
- Some `INSERT` statements had `ON CONFLICT` defined this allowed for an alternative action if the record already exists *e.g.*, perform an `UPDATE`. The use of an `INSERT` statement with `ON CONFLICT DO UPDATE` is similar to `UPSERT` in other databases.

4.1.3 Model configuration

The modelling system is configuration driven and the setup of the telescope is captured in a configuration file. The core behaviour of the correlator, the source-target relationships, and device selection is built into the model. These could be broken out into the configuration system but would require a complex configuration language akin to a programming language like Python. Hence, when the need arises to separate these from the model (*e.g.*, testing different device selection schemes) it would be easy to adapt the model to have a plugin system for this purpose. Tables 4.1, 4.2, 4.3, and 4.4 are example sections of a configuration.

Table 4.1: Model configuration: Main and subarray sections.

```
[main]
name = A64S18L36D04.32-16-04-04
topology = A64S18L36D04

[subarrays]
arr1 = 32
arr2 = 16
arr3 = 4
arr4 = 4
```

The **main** section of the configuration (Table 4.1) provides the unique `name` used to identify each model run, sometimes referred to as an experiment. The `name` is made up of the `topology` field and the **subarrays** section. The `topology` field is a short description of the system and summarises the most frequently changed variables. It is made up of the following: If the first letter is a `B` then bin

packing scheme is used to place the flows otherwise load balancing is used to place the flows. The letter `A` is followed by the number of receptors or digitisers in the subarray, this is captured as `dhost_number_of` in the **devices** section as shown in Table 4.4. `S` followed by the number of Spine switches and `L` followed by the number of Leaf switches. These values are respectively `no_spine_sw` and `no_leaf_sw` from the **network** section in Table 4.3. Lastly `D` followed by the number of separate data streams out of the digitiser. This value is twice `no_dhost_flows_per_pol` from section **correlator** (Table 4.2) since each digitiser has two polarisation's.

Table 4.2: Model configuration: Correlator section.

```
[correlator]
no_xeng_per_xhost = 4
no_dhost_flows_per_pol = 2
bw_dhost_out_gbps = 37
bw_fhost_out_gbps = 28
bw_xhost_out_gbps = 0.5
bw_beam_out_gbps = 28
```

The **correlator** section primarily specifies the output data rate in Gb/s for the D-host, F-host, X-host, and the data rate of a polarised beam.

The **network** section describes the network topology, number of Spine and Leaf switches. Switch characteristics such as the number of ports (interfaces) and per port bandwidth capability are also defined in this section.

The **devices** section control on which Leaf switch devices connect to. The `type` field lists the device types that will be connected to the modelled network and the order in which they should be attached. For each device type, there is a `leaf_start`, `leaf_end`, and a `number_of` field. The `leaf_start` and `leaf_end` fields define the Leaf switches the devices of this type will connect to. The setup script will try to evenly spread the devices over the Leaf switches from

Table 4.3: Model configuration: Network section.

```
[network]
no_switch_ports = 36
no_spine_sw = 18
no_leaf_sw = 36
bw_sw_port = 40
```

Table 4.4: Model configuration: Devices section.

```
[devices]
types = dhost,skarab,ingest,bfi,ptuse,fbfuse,seti
dhost_number_of = 64
dhost_leaf_start = 1
dhost_leaf_end = 4
skarab_number_of = 306
skarab_leaf_start = 6
skarab_leaf_end = 22
ingest_number_of = 8
ingest_leaf_start = 23
ingest_leaf_end = 23
bfi_number_of = 4
bfi_leaf_start = 23
bfi_leaf_end = 23
ptuse_number_of = 4
ptuse_leaf_start = 23
ptuse_leaf_end = 23
fbfuse_number_of = 90
fbfuse_leaf_start = 24
fbfuse_leaf_end = 28
seti_number_of = 128
seti_leaf_start = 29
seti_leaf_end = 36
```

`leaf_start` to `leaf_end`. The `number_of` field defines how many devices of a specific device type should be attached to the network. The setup script takes into consideration the number of connected devices and available ports on each Leaf switch and will not over allocate a Leaf switch.

The **network** and **devices** sections are used by the setup script to write the desired network topology into the database before the model is run.

4.1.4 Model set up

The modelling environment setup is done through a script that adds switches and devices defined in the configuration file to the database. The script then inserts all entries into the database to represent the network links (connections from devices to Leaf switches and connections from Leaf switches to Spine switches).

As previously mentioned, the design of the system was to keep it very modular and have distinct components that each fulfil a specific purpose, in software engineering this is referred to as separation of concern. In a production environment the setup script will take on a very different form. A production system setup script will ensure that the real physical network is correctly described in the database. Information will be synchronised from a DCIM system or a set of carefully curated configurations files.

The setup script configures the network bandwidth for all device-to-switch and switch-to-switch connections. For the majority of the modelling performed the links were set to $40\,\mathrm{Gb/s}$.

4.1.5 Subarray

The primary component of the modelling system is the subarray script. This script is written in Python and takes as arguments the subarray number, the number of receptors (A_d) assigned to the subarray, and the path to the configuration file.

This script is run for each subarray independently; there is no relationship between the subarrays. All the resources are available to all the subarrays, but once a resource has been assigned to a subarray it cannot also be assigned to another subarray. The subarray script could be run concurrently for multiple subarrays but in all the examples used in this dissertation they were run sequentially. This was to ensure reproducible results, having the exact same placement on a rerun. Subarray 1 is modelled first then subarray 2, then subarray 3 and finally subarray 4 was modelled, a database copy was made after each subarray was modelled.

The subarray script assigns resources to the subarray being modelled by updating the database entry for each of these resources. Once the resources have been allocated the script will determine all the data streams within the subarray. At this point only the source and destination devices for the streams are considered. Streams can be one-to-one (*1-to-1*), *1-to-N*, *N-to-1*, and *N-to-N*. All streams are treated as *N-to-N* where source and target are equal to one or more devices. A multicast address is assigned to each stream. These addresses are assigned sequentially and are not associated with stream types

or subarrays. A database entry is made for each flow in the stream, where a flow is the device to device data exchange. Each database entry contains the source and target devices, the multicast address, the data type, and the bandwidth required for the flow. For modelling the worst case bandwidth was always used.

4.1.6 Visualisation and reporting

The modelling of a subarray, running the subarray script, is dubbed and experiment. The experiment name or identifier consists out of a topology identifier and the number of receptors in each subarray (§4.1.3).

A set of target experiments was defined and the subarray script was executed for each subarray defined in the target. Take for example a target of **A64S18L36.32-16-08-04**, here the first subarray has 32 receptors ($a_d^1 = 32$), the second subarray has 16 receptors ($a_d^2 = 16$), the third subarray has 8 receptors ($a_d^3 = 8$), and the fourth subarray has 4 receptors ($a_d^4 = 4$). Because the subarrays are added one after the other we can save and report on the state after each subarray has been modelled. This allow a progressive view into the build up of the target setup. Thus while modelling for experiment **A64S18L36.32-16-08-04** the following experiments are created **A64S18L36.32-00-00-00**, **A64S18L36.32-16-00-00**, **A64S18L36.32-16-08-00**. This can be seen in Figure 5.3 where the different stages are graphically compared to one another.

The state of the network being modelled is stored after every experiment. A graphical representation of the bandwidth utilisation[1] on the Spine switches are created along with other artefacts that graphically represent the experiment.

4.2 Network simulation

The modelling aims to determine the best[2] route for multicast streams through the network. On completion, the modelling script will write a set of flow rules

[1] Examples of the Spine switch bandwidth utilisation plots (heatmaps) can be seen in Chapter 5 along with related discussions.

[2] Best is dependant on and a function of the placement algorithm selected for the specific array

to the database. The SDN controller, with the developed embedded SDN application, reads flow rules from the database and applies (program) these to the switches in the network.

All data streams used in the telescope are multicast. There are some Transmission Control Protocol (TCP) connections made between devices for CAM, since these are of insignificant bandwidth and are the typical use case for SDN they were not implemented in the simulation framework. The simulation support IGMP messages from the devices on the network and ensures that all who join or registered to the same multicast have the appropriate flows assigned so that all packets from source devices reach all the target devices. IGMP join and register requests for a particular multicast stream will only work from nodes that was associated with the multicast stream by the model.

4.2.1 ContainerNet

The simulation environment was setup using ContainerNet [36]. ContainerNet is a derivative of Mininet a popular network simulation system. ContainerNet uses Docker to run the host environments in the network simulation. Running the hosts as Docker containers allow for a more functional host environment but still keeping it lightweight.

ContainerNet is deployed as a Docker container with elevated privileges, allowing it to talk to the Docker daemon, this is necessary to start and stop containers. Switches in ContainerNet are created using the Open vSwitch software. The simulated switches run inside the ContainerNet container, and all Mininet commands can be used from the familiar Mininet terminal inside the container. Like Mininet, the complete test environment can be created from a Python script. The Mininet example scripts work unchanged in ContainerNet. In its simplest form, it can be seen that ContainerNet adds the `addDocker` method and all the underlying changes to Mininet to be able to start and stop Docker containers as hosts in the simulated environment. The `addDocker` method allow one to add Docker containers as hosts when programming the simulation environment.

A Python scripts is used to create the network environment. ContainerNet is instructed by the script to create switches, hosts, and links based on the topology in the database. It is expected that at least the modelling setup has

been performed and stored in the database.

4.2.2 Ryu

Ryu is a popular open sourced SDN controller. Ryu provided a framework for writing a SDN application in, whereas most other SDN controllers only provide a northbound API (which Ryu also offers). Having the application embedded in Ryu allowed for a more straightforward development since no connection or state management is required. The network devices and packets are available as method calls on instances passed into the application. Inserting a breakpoint into the application code while troubleshooting, you can perform deep inspection of packets and directly send control packets to devices.

Ryu is modular and supports OpenFlow standards 1.0 through to the current version 1.5, as well as the Nicira extensions to OpenFlow. Along with OpenFlow, Ryu also support several other network configuration management and metering protocols. Ryu has full support for multicast and properly decodes IGMP messages. Ryu is a lightweight application and is very easy to install since it is available as a Python module. The documentation could be better but is adequate to get started with, the examples are comprehensive, and the core code is well structured and easy to follow.

Chapter 5

Results and discussion

To answer the research questions (§1.4) posed in Chapter 1. The model generated the data flow plan through the network for most of the subarray permutations *e.g.*, first subarray with all 64 receptors ($a_n^1 = 64$, $a_n^2 = 0$, $a_n^3 = 0$, $a_n^4 = 0$), first and second subarray each with 32 receptors ($a_n^1 = 32$, $a_n^2 = 32$, $a_n^3 = 0$, $a_n^4 = 0$), subarrays with 16 receptors ($a_n^1 = 16$, $a_n^2 = 16$, $a_n^3 = 16$, $a_n^4 = 16$).

In this chapter, we present the results of some of the models runs, dubbed experiments. Line plots and heatmap plots show the experiment results. The line plots are good to compare experiments against one another regarding traffic load on the Spine switches. The heatmap plot gives insight into the utilisation of switch ports for both ingress and egress traffic of an experiment. The first section explains more about the plots and how to interpret them. The plots communicate (summarise) the data flow placement produced by the model.

Related to the research questions **Q1.2** and **Q1.3**, the load balancing placement algorithm is demonstrated. Then the worst-case receptor allocation per subarray is introduced, followed by a demonstration of the bin packing placement algorithm.

The chapter concludes with showing results related to research questions **Q2.1** (number of digitiser output streams) and **Q2.2** (connecting more than 64 receptors to the data network).

5.1 Interpretation of plots

The results of the model runs (experiments) are visualised by means of two different plots.

First, a line-plot (*e.g.*, Figure 5.1) show the aggregated data rate into and out of each Spine switch. The experiment names, line colour, and markings are shown in the legend on the right of the plot. The Spine switches on the x-axis (horizontal) are shown against the bandwidth on the y-axis (vertical). Ingress rate is a positive value, and the egress rate is a negative value per Spine switch. Thus for each experiment, there will be two lines with the same colour and markings; one above zero showing the ingress rate and one below zero showing the egress rates. The line plots have experiment A64S18L36D04.32-00-00-00, in black with no markings, as a reference. The purpose of these line plots is to give an easily digestible representation of the relative load on the Spine switches. On these line plots, experiments that use load balancing will show two[1] nearly straight horizontal lines. Experiments using bin packing will have values further away from zero on the left and slopes towards zero going right. These line plots allow one to see the utilisation over all the Spine switches, thus giving a view into the overall placement and remaining capacity.

Second, a heat map (*e.g.*, Figure 5.2) shows the per link data rate of all the Leaf to Spine switch links. These plots should be viewed from the perspective of the Spine switches. Each figure is two heat maps the left shows the data rate into the Spine switch ports and the right the data rate out of the Spine switch ports. The Spine switches are arranged along the x-axis of each plot and the Leaf switched along the y-axis. Each cell represents a link and is filled with a shade of green that matches the data rate, where darkest green is $40\,\mathrm{Gb/s}$. The data rate, in Gb/s, is also printed in each cell. Where the data rate exceeded $38\,\mathrm{Gb/s}$, the target maximum data rate per link, the value is printed in red. There is no significance when the data rate is printed blue or white, those colours are only adjusted for readability based on the shade of green in the cell. The purpose of these heat maps is to show the Leaf to Spine link utilisation and to highlight overloaded links and hot spots. The heat maps are the most useful report of each experiment run.

[1] Ingress and egress, above and below 0 respectively.

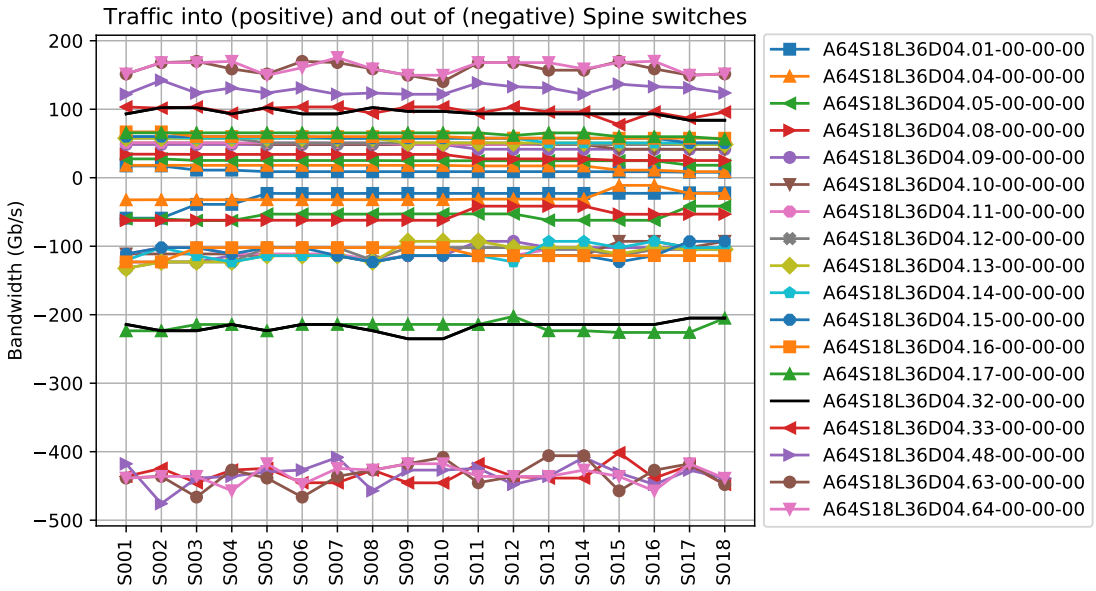


Figure 5.1: Comparing bandwidth utilisation of Spine switches for different sizes of the first subarray only.

5.2 Load balancing flow placement method

In Figure 5.1 it is demonstrated how the load balancing model distributes the bandwidth utilisation over the Spine switches. Each of the experiments in the figure had only one subarray active, and only streams from one subarray was modelled onto the network at a time. It is worth noting how the egress streams (below 0) are clustered, this is a result of the stream duplication function performed by the Spine switches. This is as a result of the correlator needing a number of inputs that is a power of two (§2.3.2). This figure could also be read as steps in the placement process of a single 64 receptor subarray with an experiment identifier of A64S18L36D04-64-00-00-00. The per link utilisation can be seen in Figure 5.2.

Figure 5.3 show experiments with more than one subarray active and several of the experiments demonstrate all 64 receptors in use. There is no data duplication in any of the experiments, number of receptors in each arrays were chosen to be a power of two. Worth noting is the impact of the number of baselines on the total egress data rate. Comparing experiment A64S18L36D04.64-00-00-00 in Figure 5.1 with experiment A64S18L36D04.32-32-00-00 in Figure 5.3, both with 64 active receptors. However, where all 64 receptors are in one subarray the total egress bandwidth per Spine switch in the network is

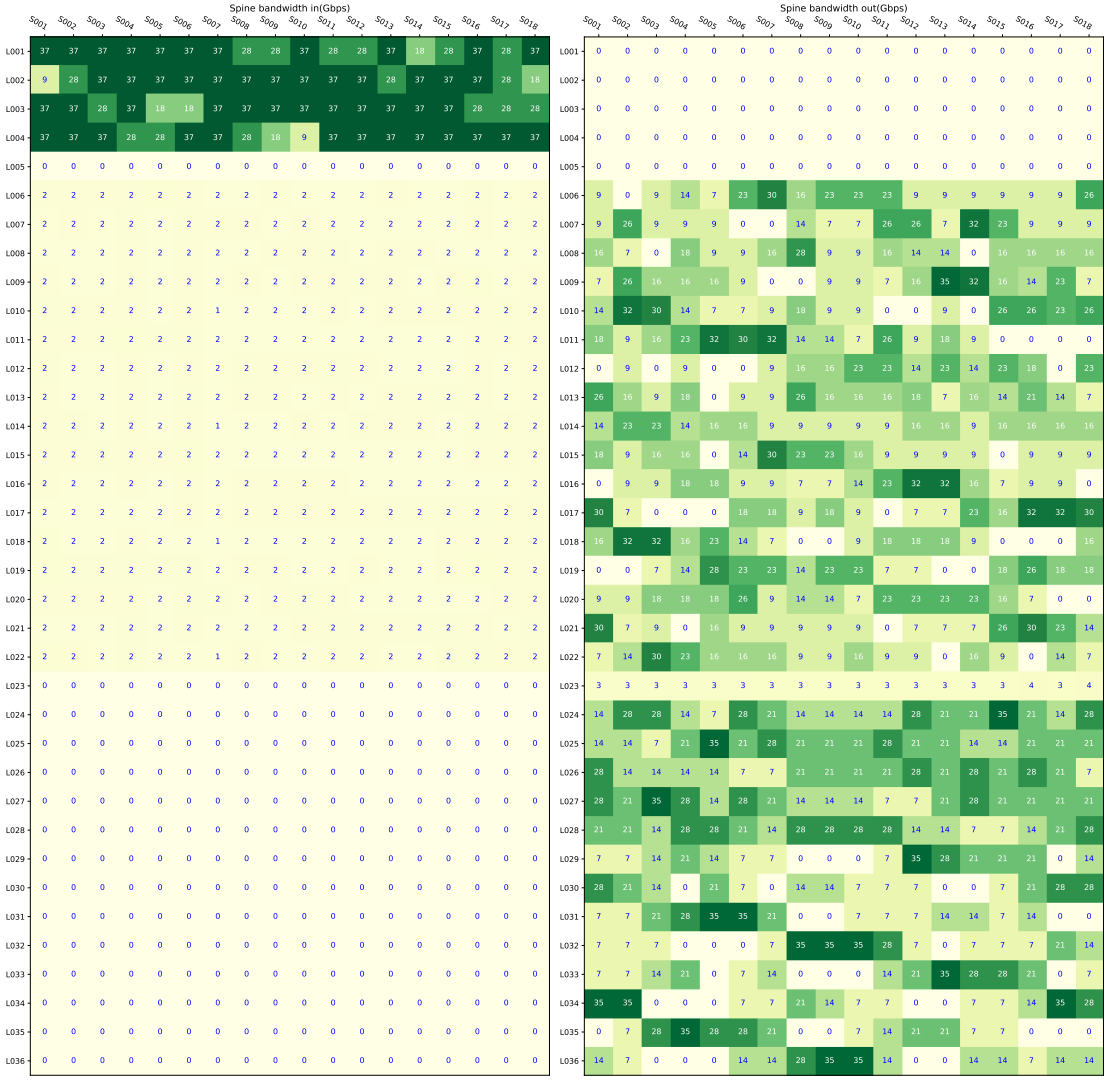


Figure 5.2: Bandwidth utilisation for each link of experiment A64S18L36D04-64-00-00-00.

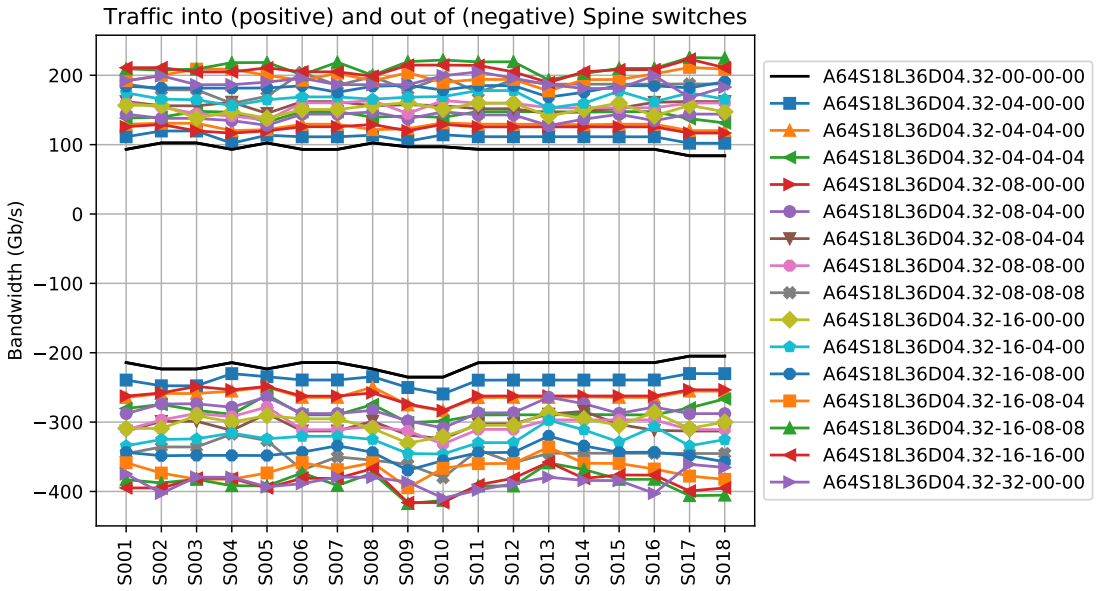


Figure 5.3: Comparing bandwidth utilisation of Spine switches for different subarray combination where subarray 1 was kept at $A_n = 32$.

more, in excess of $400\,\mathrm{Gb/s}$.

These are good initial indicators that show a software system can spread the bandwidth utilisation over the Spine switches.

5.3 Worst case scenario

To stress-test the flow allocation process, the worst possible workload has to be determined. Such a workload must use all the subarrays and should cause the maximum duplication of data flows. This set up should have the largest, by bandwidth, flows since these are more difficult to place than many smaller flows.

A stress test must work within the system boundaries, *e.g.*, no more than 64 receptors and use no more than 306 SKARAB nodes. We define the number of antennas for each of the subarrays as a_d^1, a_d^2, a_d^3, and a_d^4: thus $a_d^1+a_d^2+a_d^3+a_d^4 \leq 64$. Using the number of receptors in a subarray as input to Formula 2.3 the subarray input size (a_n^1, a_n^2, a_n^3, and a_n^4) are calculated. For each input in the subarray 2 SKARAB nodes are required: thus $2(a_n^1 + a_n^2 + a_n^3 + a_n^4) \leq 306$. Digitisers are set to output two streams per polarisation; thus all the data out of the digitisers are split into only four streams.

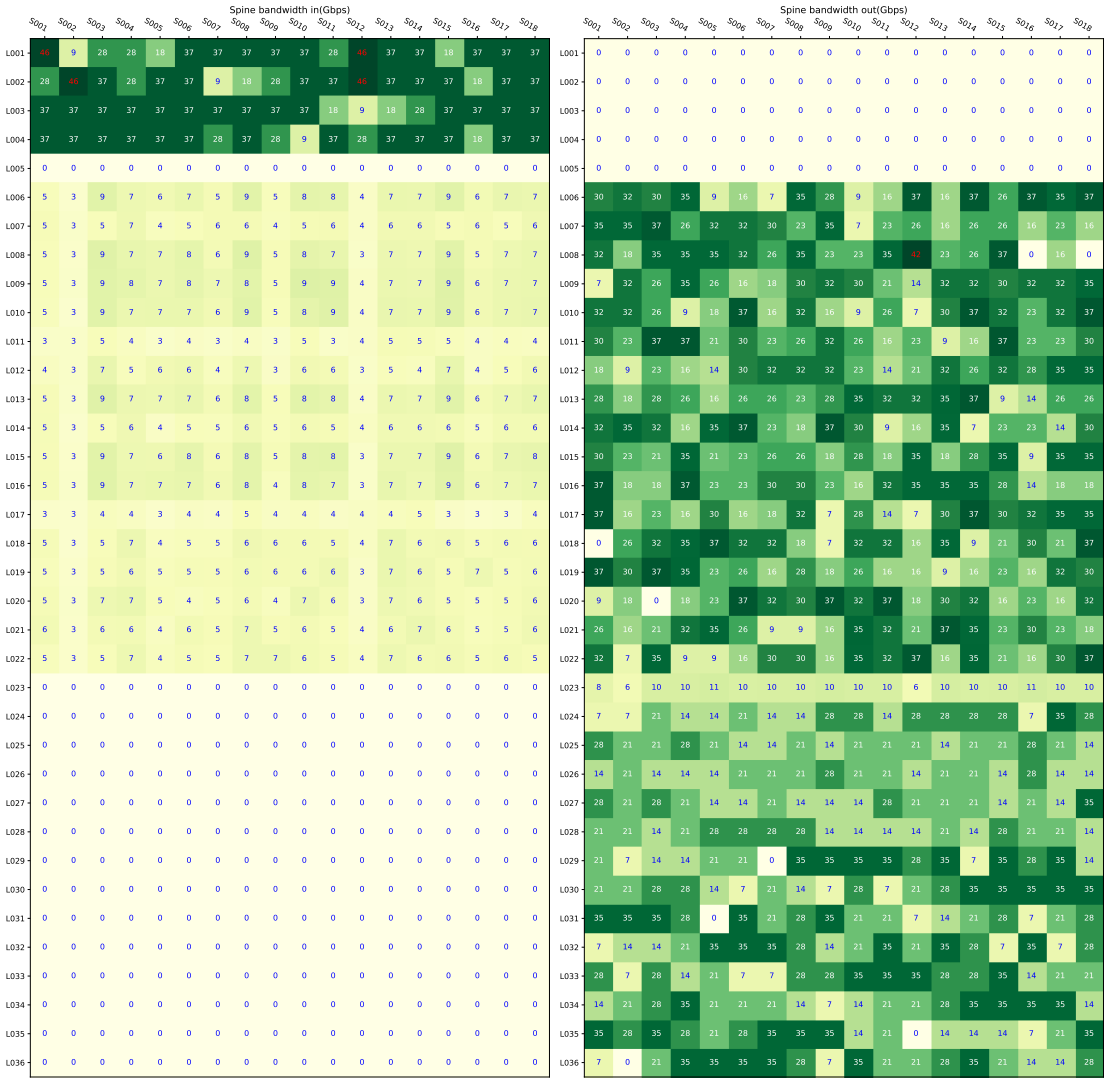


Figure 5.4: Bandwidth utilisation for each link of experiment A64S18L36D04-33-17-09-05. The load balancing placement method was used on the worst case subarray combination.

Digitiser streams are duplicated until the number of receptors (A_n) in a subarray is a power of two. Maximum duplication happens when the real receptors (A_d) are one more than a power of two, forcing the receptors (A_n) up to the next power of two. A subarray with $A_d = 33$ is configured as a 64 receptor array, resulting in 31 receptors that need to be synthesised by duplicating streams from other receptors.

Evaluating the constraints as mentioned above, we find that $a_d^1 = 33$, $a_d^2 = 17$, $a_d^3 = 9$, and $a_d^4 = 5$ used all 64 receptors and results in a subarrays with inputs of $a_n^1 = 64$, $a_n^2 = 32$, $a_n^3 = 16$, and $a_n^4 = 8$. This set up has 120 inputs and uses 240 SKARAB nodes, this subarray arrangement was executed as experiment A64S18L36D04-33-17-09-05 and the results are shown in Figure 5.4. It is worth noting that such an arrangement is improbable and has no known scientific or engineering utility.

When the flow placement system attempts to load balance the workload over the Spine switches, it is unable to place flows after the first two subarrays have been created. The load balancing placement algorithm is unable to cope with the subarray setup of $a_d^1 = 33, a_d^2 = 17, a_d^3 = 9, a_d^4 = 0$ but works for $a_d^1 = 33, a_d^2 = 17, a_d^3 = 0, a_d^4 = 0$. Figure 5.4 show the bandwidth utilisation over all Spine switch ports for the worst case scenario described here.

5.3.1 Bin packing flow placement method

When bin packing was used in experiment A64S18L36D04-33-17-09-0, it efficiently completes the task of setting a subarray configuration of $a_d^1 = 33, a_d^2 = 17, a_d^3 = 9, a_d^4 = 5$, with each digitiser outputting only 4 streams. Bin packing has an asymmetric placement, but has a very good utilisation of the switch ports. This can clearly be seen in Figure 5.5, where the same worst case scenario was played out as demonstrated in Figure 5.4 but with bin packing in place of load balancing placement objectives.

Figure 5.6 show the results of the two packing algorithms for the worst-case scenario. This figure furthermore show some of the intermediate configurations, as experiments, leading up to the worst-case scenario.

It is worth noting that in the bin packing experiments the last Spine switch

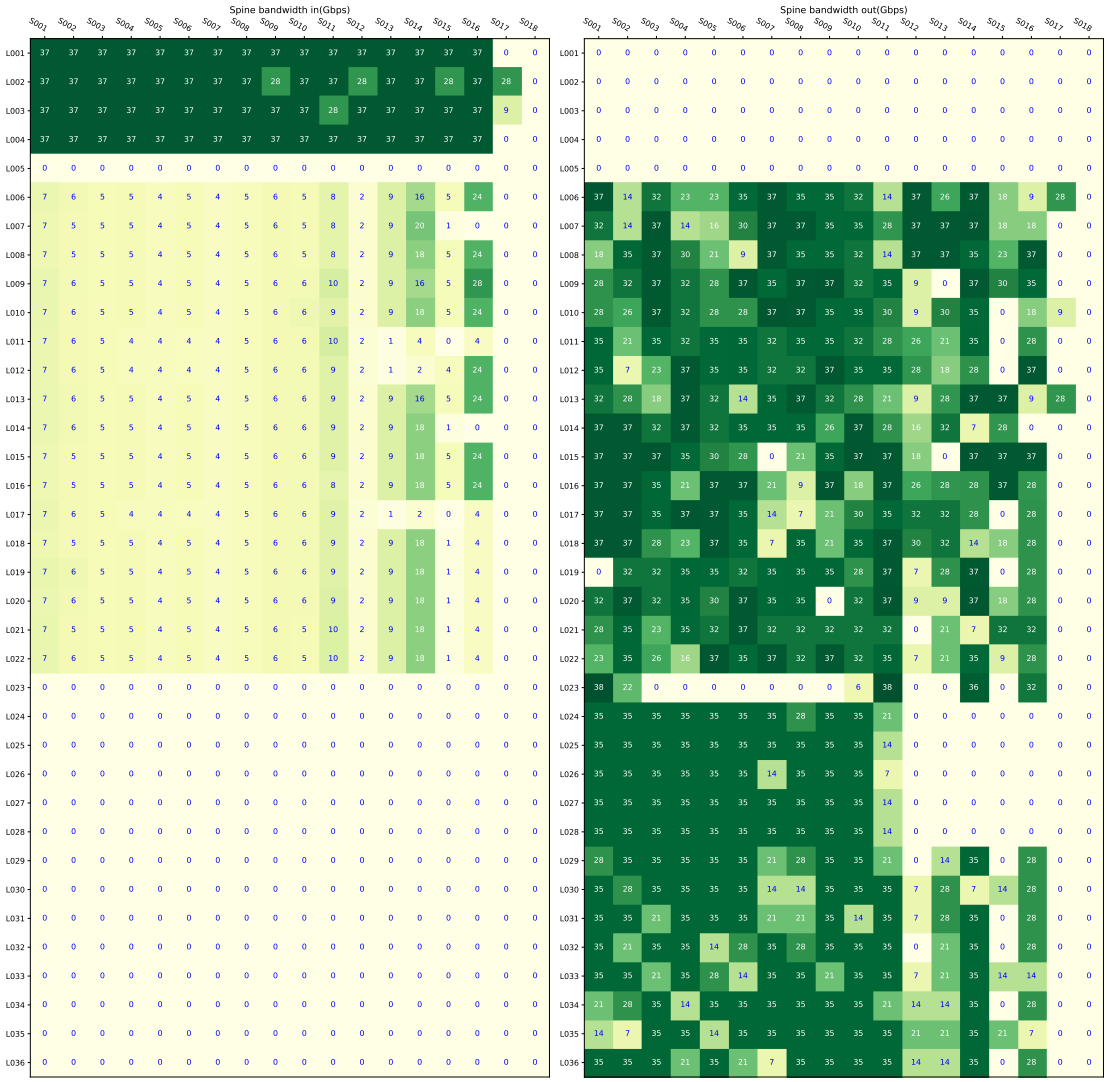


Figure 5.5: Bandwidth utilisation for each link of experiment BA64S18L36D04-33-17-09-05. The bin packing placement method was used on the worst case subarray combination.

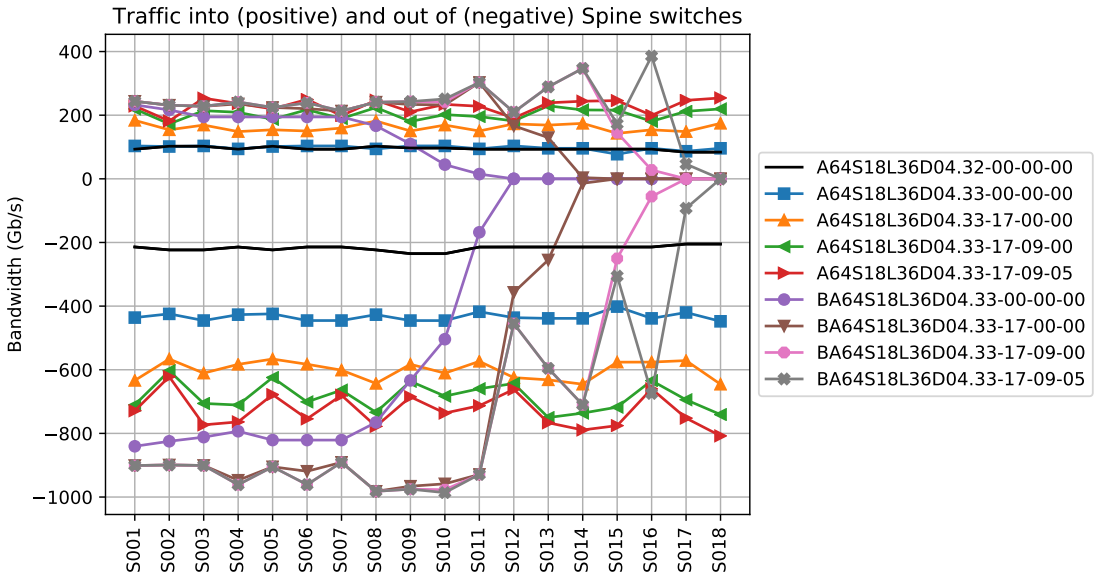


Figure 5.6: Comparing bandwidth utilisation of Spine switches in worst case subarray combination. Biggest difference between A_d and A_n.

(S018) was not utilised. Bin packing allow for a reduction of Spine switches, this would result in a theoretically blocking network. This was not further explored but such a theoretical blocking network could work for a radio telescope and would be less expensive, although the redundancy is more valuable.

5.4 Number of streams per digitiser

A digitiser with corresponding RF stage captures the vertical and horizontal polarised signals that arrive at each receptor. Each MeerKAT digitiser is connected to a receptor data switch with four $10\,\mathrm{Gb/s}$ connections. At the receptor, all digitisers, up to four, connect to a data switch but only one digitiser is active at a time. A receptor switch connects to a Leaf switch of the data network. Refer back to Figure 2.4. The digitiser data streams require a bandwidth of up to $37\,\mathrm{Gb/s}$, as shown in §2.3.1.

For each polarisation, multiple streams are emitted to balance the traffic over the four digitiser to receptor switch data links. Each of these streams have a unique multicast address and can be routed differently through the network.

To evaluate the research question **Q2.1** experiments were performed where the number of streams out of the digitiser were either four, eight, or 16 streams. This was done by splitting each polarisation into two, four, or eight streams.

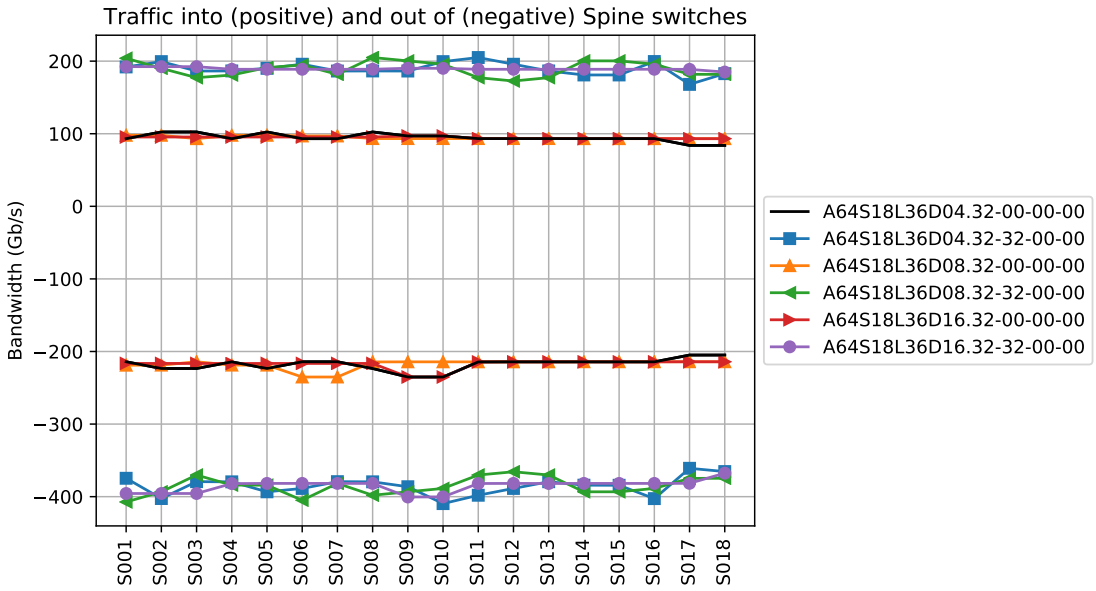


Figure 5.7: Comparing bandwidth utilisation of Spine switches for different number of digitiser output streams.

By splitting the output of the digitiser over more flows, the data rate per flow drops as the number of flows increases. From $9.25\,\mathrm{Gb/s}$ per flow for four flows to $2.3\,\mathrm{Gb/s}$ per flow for 16 flows. The smaller, lower data rate, flows are easier to place. This significantly helps the load balancing process to spread the bandwidth utilisation more evenly over the available Spine switches. With smaller flows, it is also more likely to be able to pack a link to the target flow rate when the bin packing process is used to route data flows through the network.

In the modelling experiments, the majority of subarray combinations were run multiple times with different numbers of flows from the digitisers. The D04, D08, and D16 in the topology section of the experiment name indicate how many flows were emitted by a digitiser.

Figure 5.7 show the bandwidth into and out of each of the spine switches in the network for a few subarray combinations, and each of the digitiser flow counts being discussed here. As the number of flows increases, the more evenly distributed the bandwidth utilisation per Spine becomes.

The load balancing flow placement system was not able to generate a usable flow placement map when the worst-case subarray placement was considered with the digitiser emitting only four flows. But when the flow count per digitiser was increased to eight or 16, a usable and good placement were gener-

ated. In the case where the system was evaluated with 90 receptors again, the load balancing placement system could not generate a usable map but could do so when the number of streams out of the digitiser were increased.

The increase in the number of streams to place had no significant impact on the system. Although the maximum number of digitiser streams, for 64 receptors, went from 256 to 1024 streams, this is negligible as the total number of streams in the system are added up. The increase in flows per digitiser has no effect on the number of flows in other sections of the network, *i.e.*, the number of flows out of the F-engines and X-engines were unchanged for the same number of receptors.

For completeness sake and to ensure that the number of flows does not result in other problems, all the mentioned experiment's were run with different amounts of streams per polarisation. When four or eight flows per polarisation were used both the load balancing and bin packing algorithms performed well.

5.5 Increased number of receptors

The final experiments are in relation to research question **Q2.2**, where we ask if an SDN network could easily accommodate more receptors. For this we increased the number of receptors in the modelled network to 90.

The results of this can be seen in See Figure 5.8. As mentioned, when digitiser outputs were discussed the load balancing placement cannot place a subarray with 90 receptors where each receptor has only 4 outputs from the digitiser. But when bin packing is used or the digitiser output 16 streams, the placement is possible. Similar configurations but with only 64 receptors are also shown in Figure 5.8 for comparison. Although a prominent spike is shown on the line plot of the failed experiment A90S18L36D04.90-00-00-00 one cannot categorically state that an experiment failed by only looking at the line plot. However, most failed placements do exhibit visual queues on line plots. Figure 5.9 show the link utilisation for the full compliment ($An = 90$) of receptors in a single subarray where each receptor emits 16 streams, and this experiment shows the load balancing algorithm at work.

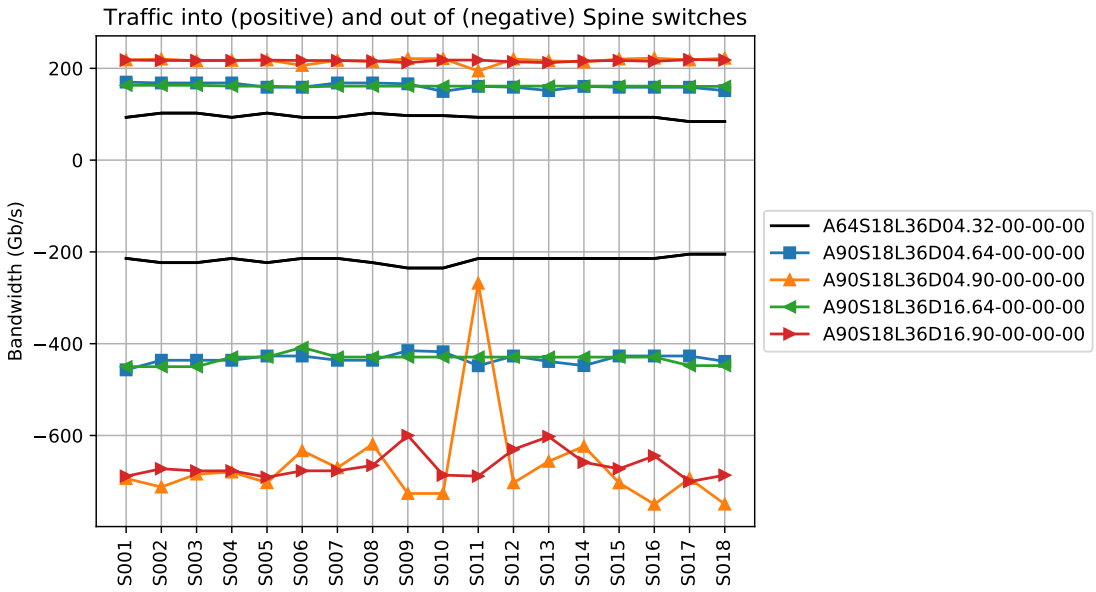


Figure 5.8: Comparing bandwidth utilisation of Spine switches for subarray 1 combinations of $A_n \in \{64, 90\}$ and digitisers with either 4 or 16 streams.

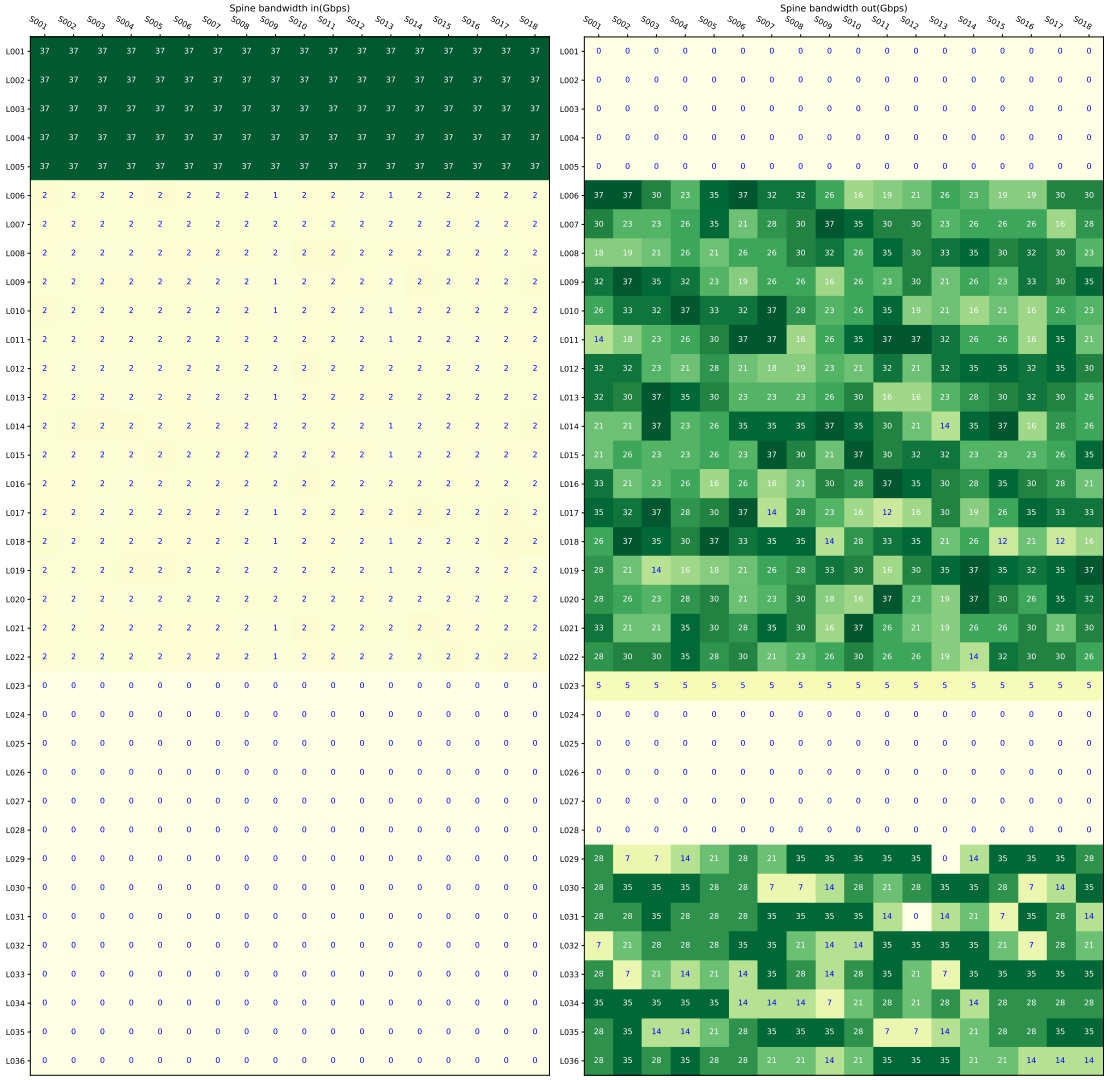


Figure 5.9: Bandwidth utilisation for each link of experiment A90S18L36D16-90-00-00-00.

5.6 Summary

In this chapter, the results of the different experiments are described and visualised. An attempt is made at presenting the results in a progressive manner. Initially, experiments are described using the load balancing packing method. Then the worst-case scenario is defined, for which the load-balanced placement method fails, thereafter the bin-packing method is introduced. Finally, increased streams from digitisers and increased number of receptors are shown.

From the example experiments presented in this chapter one can clearly see that the model is able to determine suitable placements on the network for all scenarios. Different algorithms can be selected to give different placement results. Certain scenario's, §5.3, favour one algorithm over the other.

A telescope like MeerKAT has many operational modes and it might be necessary to have different traffic patterns for different modes. This type of traffic engineering will be very difficult with the current multicast routing protocols but can very easily be implemented in an SDN network.

Chapter 6

Conclusion

This final chapter of the dissertation summarises the findings and outline future work.

The objectives (§1.3) and research questions (§1.4) combined can be summarised into a hypothesis: *Operating the multicast-based data delivery network of a radio telescope as an SDN will make the network more dynamic and easier to manage.*

The finding is that indeed SDN can be used for a multicast network of this nature and that it will make the network simpler to manage. This is mostly due to the single control authority (the SDN controller). The SDN controller has a full view of the network, thus decision logic running on the SDN controller is simpler compared to the distributed logic of non-SDN based routing protocols.

The first objective of this dissertation was to perform an in-depth investigation into the current network and how the processing components influences the utilisation of the network. The outcomes of this investigation is presented in Chapter 2. The components are described and enough of the correlator working is explained for network engineers to understand the processing pipeline and the cardinal role the network has to play. Where deemed useful the findings were reduced to equations that will be vital for implementing a production equivalent of the proof of concept presented in this dissertation. See research question **Q1.1**.

For the second objective, a full-scale model with two different algorithms was developed to demonstrate how multicast steams can be represented with flow rules. A system was architected to store and disseminate these rules to the SDN controller. A design was chosen where the SDN controller was extended with an SDN application that share information with the model via a database. See research questions **Q1.2** and **Q1.3**.

The third objective was to simulate an SDN network and test the programming of the flow rules created by the model. This verification step was done on a small-scale virtual network with virtual switches and we were able to verify correct functionality. See research questions **Q1.3**.

With the core research questions answered and demonstrated, we were able to use the infrastructure (model and simulation frameworks) to pose two interesting questions. The questions them self are interesting and not hugely onerous to answer within the existing network. But with an SDN it is much easier to contemplate (better metrics and observability) such questions and to quickly build (simpler development environment) possible solutions. As research question **Q2.1** we asked what will the effect be on the network if we had more streams out of the digitisers. Since the digitiser is connected with four $10\,\mathrm{Gb/s}$ interfaces the possible number of streams had to be a multiple of four. Four, eight, and 16 streams per digitiser were considered and we could clearly see that more streams allowed for better distributions. In the case of load balancing an equal distribution was obtained, and in the case of bin packing, the per-port utilisation was closer to the target utilisation. An SDN based data network can handle the increased number of streams and subsequent swell in flow rules. Research question **Q2.2** examines what would happen if the number of connected receptors went from 64 to 90. It was immediately evident that there are limitations in the load-balancing algorithm; the bin packing algorithm could solve for this scenario.

6.1 Discussion

There are other advantages to SDN that we did not explicitly demonstrate but would be worth mentioning.

SDN could be used to rebalance (redistribute streams) the network even

while an observation is active. This would not be the most needed feature but the SDN controller could alter the flow rules without affecting the dataflow. By carefully configuring a new route and subsequently removing the old route entries. SDN allow for dynamic failure handling, reroute multicast streams if a device fails. This is very practical for Spine switch failure but less useful for Leaf switch failures.

Allow for heterogeneous network devices since the SDN controller could be made aware of the strengths and weaknesses of these devices. Although networks like the MeerKAT data network are typically only ever built with homogeneous switches. The model could take into account the characteristics of every switch and add such features (or inabilities) to the parameter space when the flow rules for the data streams are calculated. *e.g.*, different models of switches could be used, certain streams are directed to Spine switches with bigger or better ingress/egress buffers to smooth out the packet rate[1] of a particular multicast stream.

No lock-in. Not just hardware lock-in but also no network behavioural lock-in. With an SDN if the network engineers are not satisfied with the placements of flows the model can be changed (it's just code) to suit the new behaviour. This is especially appealing for handling special cases, the outlier observations that wish to operate in a way that was not foreseen.

A final powerful possibility that SDN makes possible is that the dataflow behaviour of the network can be swapped out, by using different placement algorithms or completely different modelling programs. This concept could even be pushed as far as giving each subarray a different behaviour, by having the placement algorithm as part of the subarray configuration.

6.2 Future work

The work's progression would be to deploy the SDN controller and other components to manage physical switches. There is always a set of challenges that emerges when traversing from a simulated system to a physical system.

[1] With processing nodes like SKARAB that are synchronised to a high precision clock, all the nodes transmit packets in the network at exactly the same time. This can cause switches to drop packet since they are not able to process all packets at that instance. Increased port buffers help to alleviate the consequences of this phenomenon.

The switching hardware used for the MeerKAT network is the focus of this dissertation. Since the MeerKAT data network procurement, 100GbE has become more accessible, at a lower cost and with a broader selection of equipment. As we stand now, 200GbE and 400GbE are standardised and wide adoption of these high bandwidth interconnects are imminent. A logical next step would be to use the model described in this dissertation and the composition of currently available switches to evaluate different configurations, e.g. 64 and 128 port 100GbE switches or 32 port 400GbE switches, particularly where Spine and Leaf switches are different. While this dissertation set out to show that SDN is suitable for the use in a radio telescope it does not fully explore the different architectures possible with and SDN data network. Further research can be done around larger numbers of receptors (SKA scale), different correlator components (a mix of Graphical Processing Unit (GPU) and FPGA base), and heterogeneous switches (different port speeds, port counts, buffers, and generations). The core components of the SDN implementation will stay as described in this work, but it might be necessary to adapt the demonstration algorithms.

Although P4 [37] is still an emerging technology, it seems plausible to consider that the model created for this dissertation could use P4 to better construct the desired flow rules. The rule definition paradigm in P4 is very different from OpenFlow. P4 is a way to programmatically define the flow of packets through a switch, whereas OpenFlow is based on match-action workflow.

Bibliography

[1] B. P. Abbott, R. Abbott, T. Abbott, M. Abernathy, F. Acernese, K. Ackley, C. Adams, T. Adams, P. Addesso, R. Adhikari *et al.*, "Gw151226: observation of gravitational waves from a 22-solar-mass binary black hole coalescence," *Physical review letters*, vol. 116, no. 24, p. 241103, 2016.

[2] M. J. Jarvis, A. Taylor, I. Agudo, J. R. Allison, R. Deane, B. Frank, N. Gupta, I. Heywood, N. Maddox, K. McAlpine *et al.*, "The MeerKAT International GHz Tiered Extragalactic Exploration (MIGHTEE) Survey," 2017.

[3] S. Deering and D. Cheriton, "Host groups: A multicast extension to the Internet Protocol," RFC 966, RFC Editor, Fremont, CA, USA, pp. 1–27, Dec. 1985, obsoleted by RFC 988. [Online]. Available: https://www.rfc-editor.org/rfc/rfc966.txt

[4] Open Networking Foundation, "Software-defined networking: The new norm for networks," *ONF White Paper*, vol. 2, pp. 2–6, 2012.

[5] E. Mohammadpour and J.-Y. Le Boudec, "On packet reordering in time-sensitive networks," *IEEE/ACM Transactions on Networking*, 2021.

[6] T. Hardjono and B. Cain, "PIM-SM security: Interdomain issues and solutions," in *Secure Information Networks*. Springer, 1999, pp. 191–205.

[7] J. Moy, "OSPF Version 2," RFC 2328 (Internet Standard), RFC Editor, Fremont, CA, USA, pp. 1–244, Apr. 1998, updated by RFCs 5709, 6549, 6845, 6860, 7474, 8042. [Online]. Available: https://www.rfc-editor.org/rfc/rfc2328.txt

[8] Y. Rekhter, T. Li, and S. Hares, "A Border Gateway Protocol 4 (BGP-4)," RFC 4271 (Draft Standard), RFC Editor, Fremont, CA, USA, pp. 1–104,

Jan. 2006, updated by RFCs 6286, 6608, 6793, 7606, 7607, 7705, 8212. [Online]. Available: https://www.rfc-editor.org/rfc/rfc4271.txt

[9] D. Estrin, D. Farinacci, A. Helmy, D. Thaler, S. Deering, M. Handley, V. Jacobson, C. Liu, P. Sharma, and L. Wei, "Protocol Independent Multicast-Sparse Mode (PIM-SM): Protocol Specification," RFC 2117 (Experimental), RFC Editor, Fremont, CA, USA, pp. 1–66, Jun. 1997, obsoleted by RFC 2362. [Online]. Available: https://www.rfc-editor.org/rfc/rfc2117.txt

[10] P. Broekema, R. van Nieuwpoort, and H. Bal, "The square kilometre array science data processor. preliminary compute platform design," *Journal of Instrumentation*, vol. 10, no. 07, pp. C07 004–C07 004, jul 2015. [Online]. Available: https://doi.org/10.1088%2F1748-0221%2F10%2F07%2Fc07004

[11] P. C. Broekema, D. R. Twelker, D. C. Romão, P. Grosso, R. V. van Nieuwpoort, and H. E. Bal, "Software-defined networks in large-scale radio telescopes," in *Proceedings of the Computing Frontiers Conference*. ACM, 2017, pp. 263–266.

[12] J. Manley, "A scalable packetised radio astronomy imager," Ph.D. dissertation, University of Cape Town, 2015.

[13] L. Bondan, L. F. Müller, and M. Kist, "Multiflow: Multicast clean-slate with anticipated route calculation on openflow programmable networks," *Journal of Applied Computing Research*, vol. 2, no. 2, pp. 68–74, 2013.

[14] K. A. Noghani and M. O. Sunay, "Streaming multicast video over software-defined networks," in *2014 IEEE 11th international conference on mobile ad hoc and sensor systems*. IEEE, 2014, pp. 551–556.

[15] S. Islam, N. Muslim, and J. W. Atwood, "A survey on multicasting in software-defined networking," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 1, pp. 355–387, 2017.

[16] R. Booth and J. Jonas, "An overview of the MeerKAT project," *African Skies*, vol. 16, p. 101, 2012.

[17] P. Dewdney, "The square kilometre array," *Proceedings of the IEEE*, vol. 97, no. 8, pp. 1482–1492, 9 2009.

[18] R. Julie, “The MeerKAT array fibre network,” in *12th INCOSE SA Systems Engineering Conference*. INCOSE SA, 2016.

[19] A. R. Parsons, D. C. Backer, G. S. Foster, M. C. Wright, R. F. Bradley, N. E. Gugliucci, C. R. Parashare, E. E. Benoit, J. E. Aguirre, D. C. Jacobs *et al.*, “The precision array for probing the epoch of re-ionization: Eight station results,” *The Astronomical Journal*, vol. 139, no. 4, p. 1468, 2010.

[20] A. Foley, T. Alberts, R. Armstrong, A. Barta, E. Bauermeister, H. Bester, S. Blose, R. Booth, D. Botha, S. Buchner *et al.*, “Engineering and science highlights of the kat-7 radio telescope,” *Monthly Notices of the Royal Astronomical Society*, vol. 460, no. 2, pp. 1664–1679, 2016.

[21] D. R. DeBoer, A. R. Parsons, J. E. Aguirre, P. Alexander, Z. S. Ali, A. P. Beardsley, G. Bernardi, J. D. Bowman, R. F. Bradley, C. L. Carilli *et al.*, “Hydrogen epoch of reionization array (hera),” *Publications of the Astronomical Society of the Pacific*, vol. 129, no. 974, p. 045001, 2017.

[22] S. H. Reddy, S. Kudale, U. Gokhale, I. Halagalli, N. Raskar, K. De, S. Gnanaraj, B. Ajith Kumar, and Y. Gupta, “A wideband digital back-end for the upgraded gmrt,” *Journal of Astronomical Instrumentation*, vol. 6, no. 01, p. 1641011, 2017.

[23] A. Mattana, “A digital backend for the medicina array demonstrator,” IRA Bologna, Tech. Rep., 2017.

[24] “MeerKAT fact sheet,” http://www.ska.ac.za/wp-content/uploads/2016/07/meerkat-fact-sheet-2016.pdf, 2016.

[25] J. Manley, M. Welz, A. Parsons, S. Ratcliffe, and R. Van Rooyen, *SPEAD: Streaming Protocol for Exchanging Astronomical Data*, SKA South Africa, 10 2010.

[26] X. Yuan, “On nonblocking folded-clos networks in computer communication environments,” in *Parallel & Distributed Processing Symposium (IPDPS), 2011 IEEE International*. IEEE, 2011, pp. 188–196.

[27] C. Clos, “A study of non-blocking switching networks,” *Bell Labs Technical Journal*, vol. 32, no. 2, pp. 406–424, 1953.

[28] N. Pippenger, “On rearrangeable and non-blocking switching networks,” *Journal of Computer and System Sciences*, vol. 17, no. 2, pp. 145–162, 1978.

[29] K. Benzekki, A. El Fergougui, and A. Elbelrhiti Elalaoui, “Software-defined networking (sdn): a survey,” *Security and communication networks*, vol. 9, no. 18, pp. 5803–5833, 2016.

[30] D. Ooms, B. Sales, W. Livens, A. Acharya, F. Griffoul, and F. Ansari, “Overview of IP Multicast in a Multi-Protocol Label Switching (MPLS) Environment,” RFC 3353 (Informational), RFC Editor, Fremont, CA, USA, pp. 1–30, Aug. 2002. [Online]. Available: https://www.rfc-editor.org/rfc/rfc3353.txt

[31] B. Fenner, M. Handley, H. Holbrook, and I. Kouvelas, “Protocol Independent Multicast - Sparse Mode (PIM-SM): Protocol Specification (Revised),” RFC 4601 (Proposed Standard), RFC Editor, Fremont, CA, USA, pp. 1–150, Aug. 2006, obsoleted by RFC 7761, updated by RFCs 5059, 5796, 6226. [Online]. Available: https://www.rfc-editor.org/rfc/rfc4601.txt

[32] M. Handley, I. Kouvelas, T. Speakman, and L. Vicisano, “Bidirectional Protocol Independent Multicast (BIDIR-PIM),” RFC 5015 (Proposed Standard), RFC Editor, Fremont, CA, USA, pp. 1–43, Oct. 2007. [Online]. Available: https://www.rfc-editor.org/rfc/rfc5015.txt

[33] S. Tang, B. Hua, and D. Wang, “Realizing video streaming multicast over sdn networks,” in *9th International Conference on Communications and Networking in China*, 2014, pp. 90–95.

[34] A. Khalid, A. H. Zahran, and C. J. Sreenan, “Prototyping and evaluating sdn-based multicast architectures for live video streaming,” in *42nd IEEE Conference on Local Computer Networks (LCN)*. Institute of Electrical and Electronics Engineers (IEEE), 2017, pp. 1–3.

[35] B. W. Boehm, “A spiral model of software development and enhancement,” *Computer*, vol. 21, no. 5, pp. 61–72, 1988.

[36] M. Peuster, H. Karl, and S. van Rossem, “MeDICINE: Rapid prototyping of production-ready network services in multi-PoP environments,” in *2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, Nov 2016, pp. 148–153.

[37] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese *et al.*, “P4: Program-

ming protocol-independent packet processors," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 87–95, 2014.

Appendix A

Placement methods

This section gives more detail on the balanced and bin packing algorithms used in the model. Pseudo code and several visualisations are provided to illustrate the working of these algorithms. No in depth analysis, performance evaluation, or head-to-head comparison were performed. The pseudo code and visualisations are simplifications of the implementation and aim to illustrate the algorithms only.

A.1 Pseudo code functions

In the pseudo code used to describe Algorithm 1 and Algorithm 2 there are common functions. A brief description of these functions:

$firstSwitch(S)$ Return the first switch of S and set the internal reference to 0.

$nextSwitch(S)$ Return the next switch in S, this increments the internal reference and return the switch at this position in S. An $Empty$ object is returned if there are no more switches in S.

$dbAllocateMtoSpineSW(s, m)$ In the database set the RP for m to switch s.

$dbCheckOverloadedSpineSW(s)$ In the database see if switch s has any oversubscribed interfaces. Return $True$ if the switch is overloaded otherwise $False$ is returned. Both egress and ingress of every interface of the switch are evaluated.

$dbRemoveMfromSpineSW(s, m)$ In the database remove the RP for the multicast streams m from switch s.

Algorithm 1 Balanced placement algorithm.

```
1: procedure BALANCEDPACKING(S, M)
Require: S the available Spine switches
Require: M the multicast streams to place
2:     s ← firstSwitch(S)
3:     for each m ∈ M do
4:         r ← sizeOf(S)
5:         dbAllocateMtoSpineSW(s, m)
6:         while dbCheckOverloadedSpineSW(s) do
7:             r ← r − 1
8:             if r < −1 then
9:                 return False                    ▷ Interrupt procedure
10:            end if
11:            dbRemoveMfromSpineSW(s, m)
12:            s ← nextSwitch(S)
13:            if s = Empty then
14:                s ← firstSwitch(S)              ▷ Reset the lookup on S
15:            end if
16:            dbAllocateMtoSpineSW(s, m)
17:        end while
18:        s ← nextSwitch(S)
19:        if s = Empty then
20:            s ← firstSwitch(S)                  ▷ Reset the lookup on S
21:        end if
22:    end for
23:    return True
24: end procedure
```

A.2 Balanced placement

The balanced placement algorithm aims to spread the utilisation across all Spine switches, in a uniform manner. See Algorithm 1 for a pseudo code implementation. This type of balanced placement algorithm is also referred to as a load balancing algorithm, where load in this context is the summed network bandwidth utilisation over all interfaces of a switch. This algorithm is computationally simple while there is plenty of capacity, as the network fills up, it takes longer to find a suitable placement.

A.3 Bin packing placement

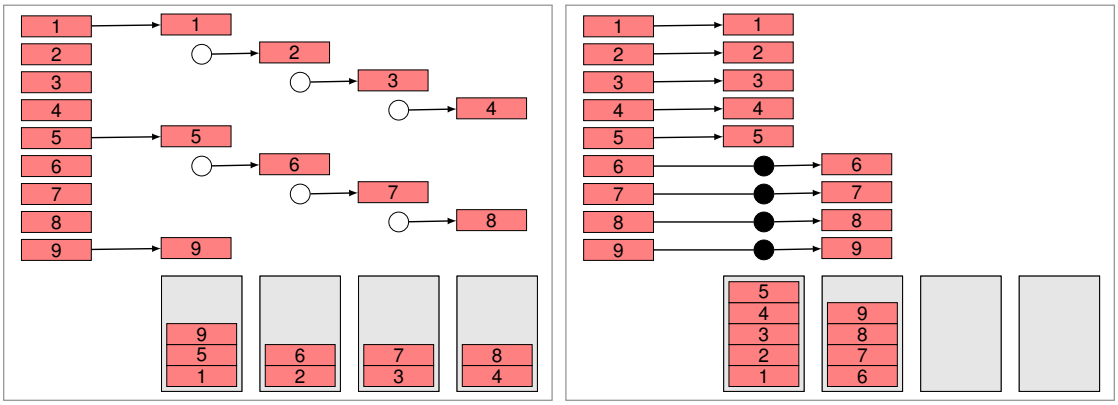
The bin packing placement algorithm aims to utilisation each Spine switch to its maximum capacity. See Algorithm 2 for a pseudo code implementation. Each stream processed by this algorithm will take longer to place than the previous stream.

Algorithm 2 Bin packing placement algorithm.

```
1: procedure BinPacking(S, M)
Require: S the available Spine switches
Require: M the multicast streams to place
2:     for each m ∈ M do
3:         s ← firstSwitch(S)                    ▷ Reset the lookup on S
4:         dbAllocateMtoSpineSW(s, m)
5:         while dbCheckOverloadedSpineSW(s) do
6:             dbRemoveMfromSpineSW(s, m)
7:             s ← nextSwitch(S)
8:             if s = Empty then
9:                 return False                  ▷ Interrupt procedure
10:            end if
11:            dbAllocateMtoSpineSW(s, m)
12:        end while
13:    end for
14:    return True
15: end procedure
```

A.4 Examples showing algorithms side by side

The visualisations presented in this section has the balanced placement on the left and the bin packing on the right. In each figure the same streams, represented by the nine vertically stacked rectangles, has to be placed into the same switches. The switches are the four grey rectangles spaced horizontally at the bottom of the figure. Each switch has a capacity of 5 units, red streams have a size of 1 unit, blue streams have a size of 2 units, and the yellow streams have a capacity of 3 units. The height of the rectangles representing the switches and streams are proportional to its capacity or size. In each figure you can see the way the algorithms pack the streams onto the switches, the streams are processed top-down. The topmost stream, enumerated as 1, is processed first. The bottom stream, enumerated as 9, is processed last. The empty circles indicate that the algorithm moved to the next switch without evaluating the cur-



(a) Balanced (b) Bin packing

Figure A.1: Placement of small flows.

rent switch; this is only applicable for the balanced algorithm. The black circle represents where a switch was considered, but picking it would have resulted in overloading.

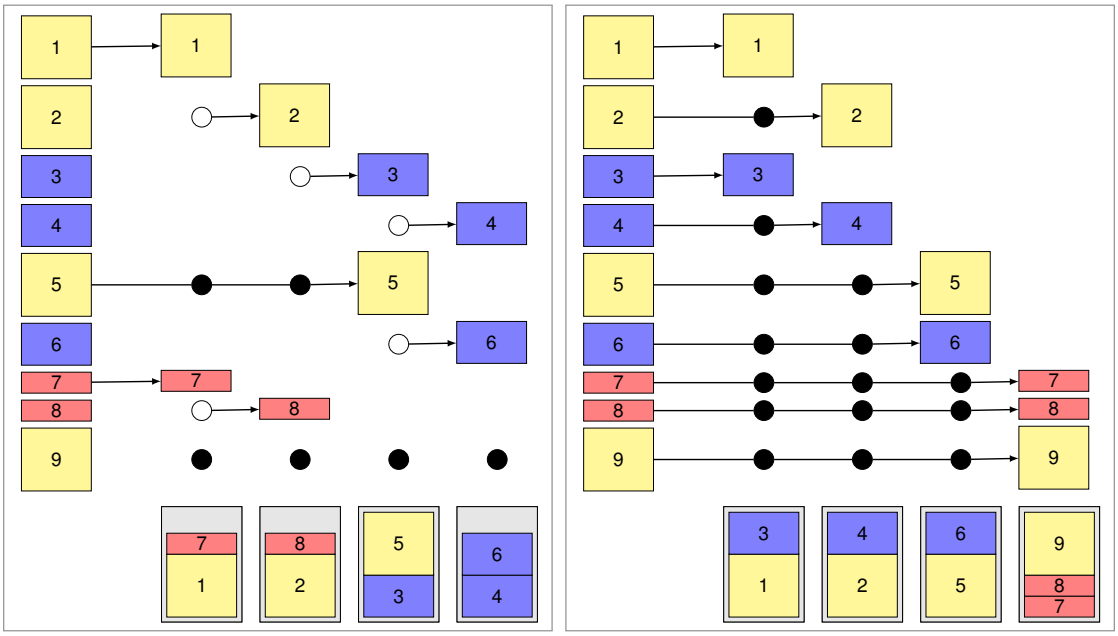
Each stream is drawn as 3 rectangles. The number in the rectangle is the order of processing. Each algorithm has three rectangles that are labelled '1' showing the progressing of decision making, left to right, and the placement in the bottom grey rectangles that represent the switches.

A.4.1 Small streams

Figure A.1 demonstrates the placement of nine small streams to the four switches. In Figure A.1a the streams are spread across all the switch when using the balanced placement. In Figure A.1b, bin packing, the first switch is filled and only then is the second switch considered.

A.4.2 Mixed streams

Figure A.2 show a more realistic example where a mix of streams with different sizes have to be placed. In this example, the balanced (Figure A.2a) is unable to solve the placement problem and the last stream cannot be allocated to a switch. The bin packing algorithm (Figure A.2b) is able to resolve.

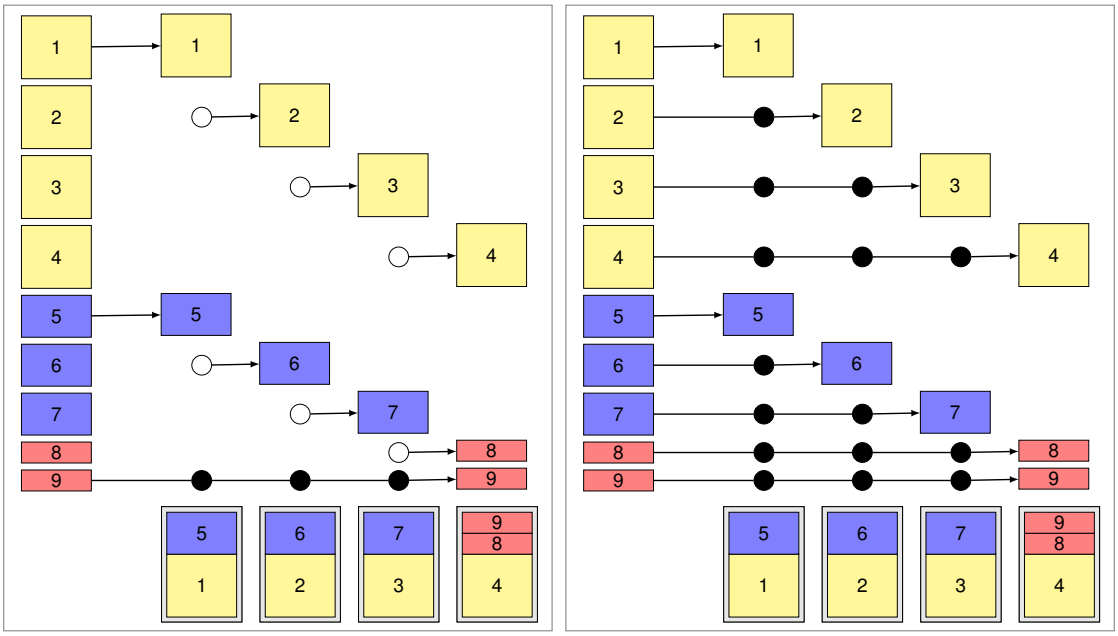


(a) Balanced (b) Bin packing

Figure A.2: Placement of flows of mixed sizes.

A.4.3 Sorted mixed streams

The same mix of streams as shown in Figure A.2 is used but the streams are first sorted by size. By first placing the biggest streams, Figure A.3a, the balanced placement algorithm is able to solve the placement problem and yields the same result as bin packing, Figure A.3b. Although sorting the streams from biggest to smallest before using the balanced packing method yields a better result, it does not always work.



(a) Balanced (b) Bin packing

Figure A.3: Placement of flows of mixed sizes sorted largest to smallest.

A.5 Implementation detail

There are two minor optimisations in the implementation. First, the list of available Spine switches elected from the database only includes spine switches that are not already at their target capacity. Secondly, the list of multicast streams to place are returned in sorted order, descending size, from the database. In both cases, this was a minor change to the SQL query but had a noticeable effect on the model.

Appendix B

Units and notation

To avoid ambiguity, when unit prefixes for multiple units are used, the International Electrotechnical Commission (IEC) binary prefix notation has been adopted in this document. A more extensive and regularly updated explanation can be found at https://en.wikipedia.org/wiki/Binary_prefix.

The decimal prefixes kilo (k), mega (M), giga (G) and other International System of Units (SI) units are used as they were defined for multiple powers of 10. The binary prefixes kibi (Ki), mebi (Mi), gibi (Gi) and other binary prefixes defined by IEC are used to denote multiple powers of 2. See Table B.1.

Table B.1: Decimal and binary prefixes.

Decimal prefix				Binary prefix			
Value		SI unit		Value		IEC unit	
10^3	1000	kilo	k	2^{10}	1024	kibi	Ki
10^6	1000000	mega	M	2^{20}	1048576	mebi	Mi
10^9	1000000000	giga	G	2^{30}	1073741824	gibi	Gi
10^{12}	1000000000000	tera	T	2^{40}	1099511627776	tebi	Ti

Where bits and bytes are used bits (b) will be lowercase and bytes (B) uppercase.
1 Gb/s is 10^9 bits per second or 125×10^6 bytes per second (125MB/s).

Appendix C

Publications

The following conference papers and presentations are a result of work done for this dissertation:

1. M. Slabber, J. Manley, J. Mwangama, and N. Ventura, "MeerKAT data distribution network" in Software and Cyberinfrastructurefor Astronomy V, vol. 10707.SPIE, 2018.
   [Online]. Available: https://doi.org/10.1117/12.2311870
2. M. Slabber, J. Manley, J. Mwangama, and N. Ventura, "Introductory evaluation of software defined networking for the MeerKAT data distribution network" in Proceedings of the Southern Africa Telecommunication Networks and Applications Conference (SATNAC), 2018, pp. 188–191.
   [Online]. Available: http://www.satnac.org.za/proceedingsn2.html