Diagnosis of Faults in Data Centre Networks and Network Visualization using Data Plane Programmability and Relational Queries

THESIS

Submitted in partial fulfillment of the requirements of
BITS F421T Thesis

By

Sankalp Sanjay Sangle
ID No. 2016A7TS0110P

Under the supervision of:

Dr. Mun Choon CHAN
&
Dr. Virendra S SHEKHAWAT

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI, PILANI CAMPUS
June 2020

Declaration of Authorship

I, Sankalp Sanjay Sangle, declare that this Thesis titled, 'Diagnosis of Faults in Data Centre Networks and Network Visualization using Data Plane Programmability and Relational Queries' and the work presented in it are my own. I confirm that:

- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: _____

Date: _____

Certificate

This is to certify that the thesis entitled, "*Diagnosis of Faults in Data Centre Networks and Network Visualization using Data Plane Programmability and Relational Queries*" and submitted by Sankalp Sanjay Sangle ID No. 2016A7TS0110P in partial fulfillment of the requirements of BITS F421T Thesis embodies the work done by him under my supervision.

Supervisor
Dr. Mun Choon Chan
Associate Professor,
School of Computing, National University of Singapore
Date:

Co-Supervisor
Dr. Virendra S Shekhawat
Assistant Professor,
Birla Institute of Technology and Science, Pi
Date:

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI, PILANI CAMPUS

Abstract

Bachelor of Engineering (Hons.) Computer Science

Diagnosis of Faults in Data Centre Networks and Network Visualization using Data Plane Programmability and Relational Queries

by Sankalp Sanjay Sangle

Data Centers hav emerged as the basic unit for providing web services to customers by major technology companies. They house servers in the thousands and tens of thousands, with servers often serving as backups to ensure reliability. The reliability of data center network infrastructure is critically important for ensuring seamless user experience. The nature and distribution of faults in data center networks isn't completely understood, and many faults like microbursts due to fan-in traffic go undetected. A closer look is needed at the type of faults that occur in data center networks and on how they can be diagnosed. Modern data centres operate at throughputs of close to a few hundreds of Gbps, implying packets coming in at a few hundreds of nanoseconds apart. Most monitoring solutions offer resolution of the order of milliseconds, and hence are inadequate for recording and detection of events that last for sub millisecond intervals. This thesis aims to discuss techniques for diagnosing faults in data centers, develop a monitoring system to view them, at nanosecond raesolution. It is also demonstrated that appropriate calculations can be performed on the SQL queries to extract quantities like ingress throughputs, egress throughputs, relative ratios of packets in a network, (all at nanosecond resolution) thereby demonstrating that collecting relational data about switches is a simple and yet powerful way to extract meaningful data and create relevant visualizations[4] which can help a network administrator determine the root cause of faults in the network and address them.

Acknowledgements

This thesis has been written in a period when the world is at a standstill due to the threat of COVID-19. As such, it has been a difficult but deeply enriching experience in Singapore, and it would be ungrateful of me to not give people their due acknowledgements.

I am grateful to my supervisor, Dr. Mun Choon Chan, and co-supervisor Dr. Virendra S Shekhawat for their guidance and availability during the period of my thesis. I am also thankful for the guidance provided by Ph.D. candidates Pravein Govindan Kannan and Nishant Budhdev, and for their gracious nature and helpfulness in addressing all my doubts and making me feel comfortable in a foreign place. The numerous Zoom calls with Professor Chan, Pravein and Nishant were a beautiful learning experience for me. I am indebted to my roommate turned friend Yashdeep Thorat for having been a source of comfort, motivation, and stability when things turned difficult to bear. I am grateful to my family for their constant support, for being able to fall back on them, and for always being a phone call away.

I am thankful to Singapore for having proven to be a place where I felt comfortable and at ease right from the first day. The culture and warmth of Singaporeans was a delight to observe and experience and I hope to one day, when it is safe again, return to Singapore and relive some of my experiences.

Finally, I might not have taken the leap and tried to apply for such an opportunity had it not been for the timely intervention and push by Miloni, and for that I am indebted to her...

Contents

List of Figures

List of Tables

Chapter 1

Introduction to Thesis Topic

Data Centers have emerged as the basic unit for providing web services to customers by major technology companies. They house servers in the thousands and tens of thousands, with servers often serving as backups to ensure reliability. Traffic in data centers is often organized in a tree like structure, with top of rack switches, aggregate switches and core switches. The tree like structure is easy to maintain and scale to meet the growing number of clients.
The reliability of data center network infrastructure is critically important for ensuring seamless user experience. Companies like Facebook[8] and Google[5] have spent considerable investments into developing frameworks for automating fault detection and repair of networks. Even so, the nature and distribution of faults in data center networks isn't completely understood, and many faults like microbursts due to fan-in traffic go undetected. A closer look is needed at the type of faults that occur in data center networks and on how they can be diagnosed.
Modern data centres operate at throughputs of close to a few hundreds of Gbps, implying packets coming in at a few hundreds of nanoseconds apart. Most monitoring solutions offer resolution of the order of milliseconds, and hence are inadequate for recording and detection of events that last for sub millisecond intervals. This motivates the need for having a diagnosis system to distinguish events at nanosecond level by leveraging advances in data plane programming[6].
This thesis aims to discuss techniques for diagnosing faults in data centers, develop a monitoring system to view them, at nanosecond resolution. It is also demonstrated that appropriate calculations can be performed on the SQL queries to extract quantities like ingress throughputs, egress throughputs, relative ratios of packets in a network, (all at nanosecond resolution) thereby demonstrating that collecting relational data about switches is a simple and yet powerful way to extract meaningful data and create relevant visualizations[4] which can help a network administrator determine the root cause of faults in the network and address them.

A high level architecture diagram of the proposition is shown in Figure 1.1

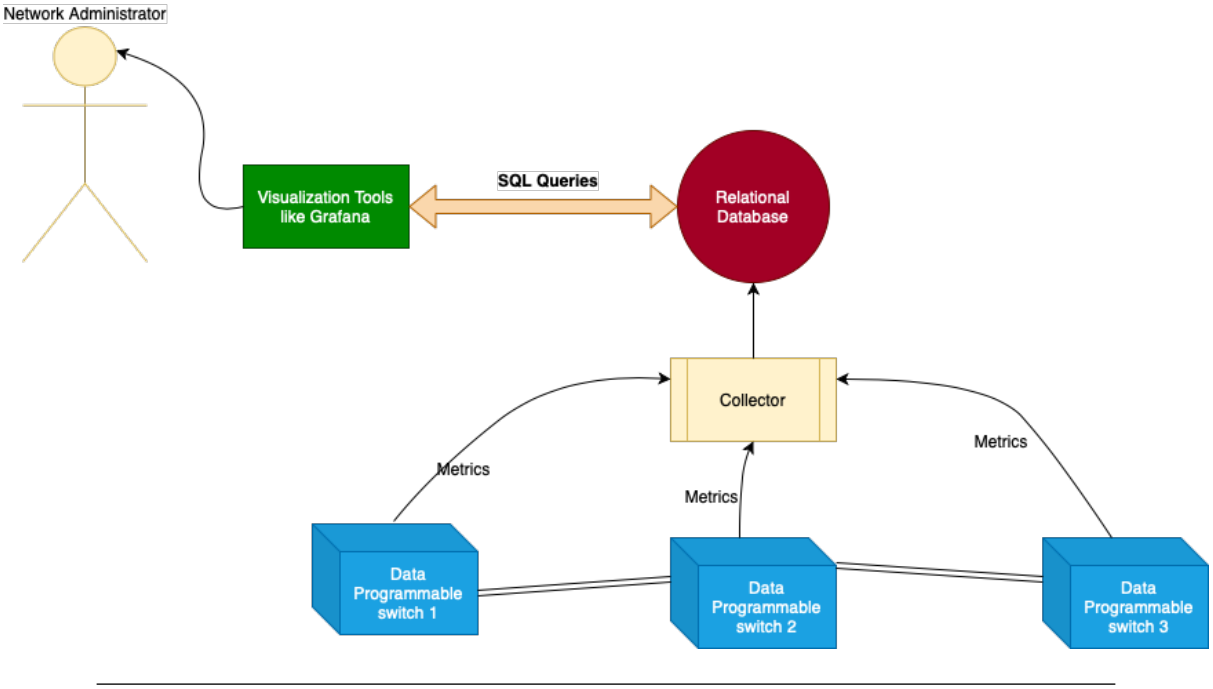

FIGURE 1.1: Architecture Diagram of proposed system

Chapter 2

Background Literature

Historically, computer networks have always been relatively blackboxed entities that provide no substantial facilities for configuration. The devices that are present in them, from traditional switches and routers, to other middleware like firewalls and network address translators, often execute software that is proprietary, and hence rigid by nature. These devices are configured by network administrators using highly specialised interfaces that vary from vendor to vendor. The software for communicating with these interfaces is not centralized, and network devices need to be individually configured to achieve the desired functionality. This frozen structure has hindered flexibility and increased costs of network management.
It is interesting to note that this structure was by design and not an oversight. The nascent stage Internet needed to be absolutely reliable and fault tolerant if widespread adoption was to take place. One way to minimize failures in the network was to make the network devices rigid and inflexible. However, by the turn of the century, the increase in number of end hosts and the widespread adoption of the Internet has given rise to a need for managing large flows of traffic effectively. The end host explosion has made the question of programmable networks much more pragmatic to discuss.

2.1 Why do we need programmable networks?

The ability to remotely configure behaviour of switches/routers finds use in a number of applications, some of them being:

- Dynamic Routing of heavy hitters
- Testing of new protocols/routing policies
- In-Network Computing

- Layer 4 Load Balancing
- Network Monitoring and Debugging (The focus of this thesis)

A programmable network is easier to monitor and control than traditional networks. Network devices whose forwarding policies can be customized can be made to behave like a router, a switch, a firewall, a NAT, or any other device we can conceptualize within the constraints of data plane switch programmability. This saves money and physical labour that would have been spent in the purchase and set up of highly specific-function network devices. A programmable network is, by nature, facilitative to network innovation and reduces the barrier to introduction of new protocols/policies.

2.2 SDN and the idea of separation of Control Plane and Data Plane

Software Defined Networking (SDN) is an umbrella term used to address efforts made to make networks programmable. The behaviour of traditional switches is defined in hardware, whereas 'software-defined' switches would have the programmability addressed earlier. One important concept of SDN is *the decoupling of Control and Data planes*. Control plane refers to protocols like OSPF, BGP, Multicast which govern traffic handling on a higher scale. The data plane performs packet switching based on the policies that the control plane dictates. In a general sense, the control plane is the intelligence layer, and the data plane is the manifestation of it in a standalone switch.

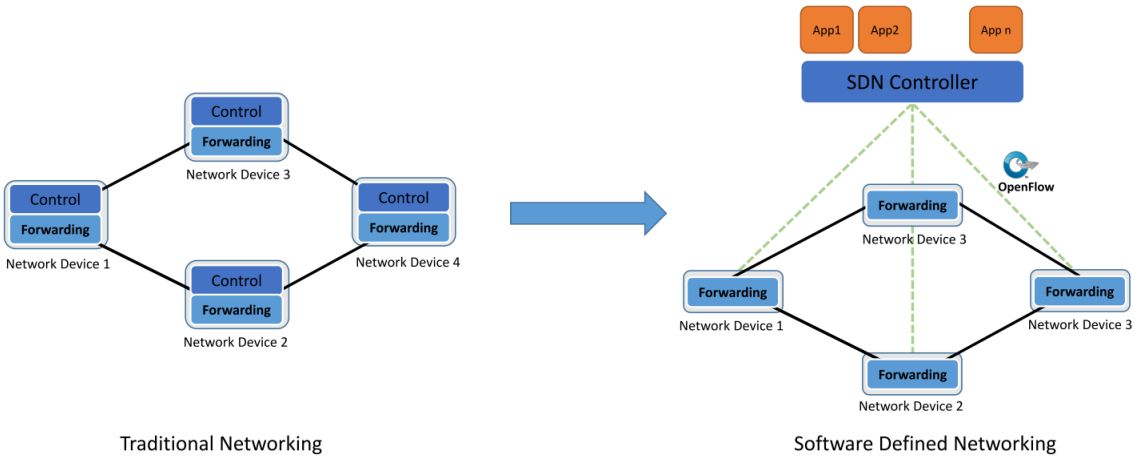

FIGURE 2.1: Separation of Control and Data planes

A traditional switch has the forwarding logic baked into the circuit, and tight integration of data

and control planes. The idea behind SDN is to decouple the planes (Figure 2.1), and have a programmable control plane that can communicate with the data plane via a standardized API (like OpenFlow[7]) so as to offer freedom in specifying how a switch handles packets rather than 'hardcoding' it into the switch.

2.3 A Brief History of Programmability in Computer Networks

2.3.1 Early approaches towards programmability (1990s - late 2000s)

The early days of SDN research revolved around what were known as Active Networks[1]. In this, two approaches were used:

- *Capsule model*, where code to be run at the nodes is carried within packets
- *Programmable router/switch model* where the code is pushed to the nodes by out-of-band means

While capsule models were feasible, there were concerns about attackers injecting malicious code into a router and compromising the network. The programmable model was much more aligned to what SDN is today. Most of the opposition towards adopting these approaches was due to a mixture of concerns over reliability and performance, and also because it was important to ensure proliferation of the Internet by keeping the network core as simple as possible.

2.3.2 OpenFlow (and its limitations)

OpenFlow started off as a research project that was implemented at Stanford University[7] by researchers in a campus wide network. As mentioned before, it defines the API for communication between a decoupled data plane and control plane. OpenFlow quickly became popular due to it being easy to adopt (most commodity switches required a firmware upgrade). The OpenFlow specification provides means to specify entries in match-action tables(called rules) where a match is made against the headers of the packet, and an action is taken(forwarding, dropping, broadcasting). Widespread adoption by companies like Google[5] boosted its popularity and many new switch vendors entered the established markets by undertaking early adoption of OpenFlow. There is a false belief that OpenFlow is the same as SDN; SDN is a general term used for addressing techniques that make networks programmable. It provides no direction or proposal on *how* to go about doing that. OpenFlow is merely one concrete way to give a level of programmability to networks.
OpenFlow however, does not offer any data plane programmability support. There is no means

to specify new protocols and fields to match against, or on how to perform reassembly of packets; consider a scenario where one needs to test a new protocol. OpenFlow enabled switches will not be able to perform match-actions on fields specified in the protocol and will have to wait until that protocol gets added into the specification (which can take of the order of months/years). This use case motivates the desire to make the data plane itself programmable.

2.4 Towards Data Plane Programmability

Data plane programming offers the flexibility of describing how the packet headers can be parsed and matched onto non-standard fields (for example, those of a new protocol) and modified, as well as the order in which the headers are reassembled before being transmitted by the switch. It is natural to think that there will be a tradeoff between throughput/speed and customizability, and this was the notion in the research community for a number of years. However, programmable switches have been made possible by recent advances networking architectures[2] in Figure 2.2. They expose such functionality in the data plane:

- Flexible parsing and deparsing on packet headers
- Ingress/Egress processing using match-action tables
- Stateful maintenance of network states using SRAMs

These switches offer throughputs of close to 1 Tbps (the rate demanded by modern data centres) even with the ability to process and modify the packet. Such speeds would be impossible if packet processing was to happen in CPU due to various overheads, which is what makes data plane programmability so attractive.

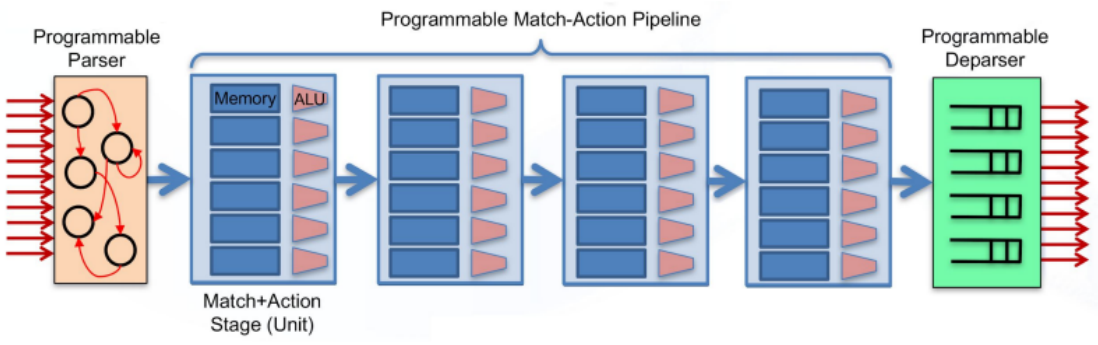

FIGURE 2.2: Generic Data Programmable Switch Architecture

2.4.1 Constraints in data plane programmable switches

In order to maintain packet processing at line-rate with no slow down in throughput, certain constraints need to be followed which impose a limit on the packet processing flexibility. Some of these are:

- No looping constructs in P4[3] (the language used to program switches)
- no floating point computations
- no exact multiply/divides (only approximated by bit-shifts)
- only one read-modify-write per stage to maintain processing at line rate

Even so, data plane programmability has found a number of applications in recent years including Network Monitoring and Debugging (the subject matter of this thesis), time synchronization, detection of elephant or heavy hitter flows[9], and In-Network Computing.

2.4.2 The P4 Programming Language

P4 (*Programming Protocol Independent Packet Processors*)[3] is a programming language developed specifically for programming data plane switches. The language is maintained by the P4 Language Consortium and is widely used for data plane programmability today. It provides constructs for specifying deterministic parsers for header parsing, action constructs for specifying steps to be taken in case of a match, and also steps to be taken in packet reassembly.

Chapter 3

Experiments

3.1 Experimental Setup

Data Center networks often use a fat-tree topology as this topology is easy to maintain and to scale. To simulate a fat-tree topology as seen in data center networks, 4 physical servers and 2 data plane programmable switches were used. Each switch was virtualized to create 5 switches using 10G loopback links. So, we have 10 switches in total, with 4 of them as top-of-rack or edge switches, 4 of them as aggregate switches, and 2 as core switches (Figure 3.1)

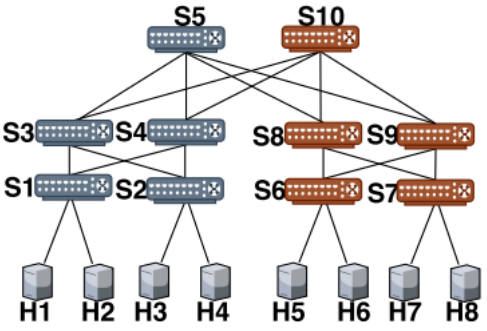

FIGURE 3.1: Evaluation Topology

3.2 SQL Database Schema

The relational schema for the packet records collected from switches is as shown in Figure 3.2

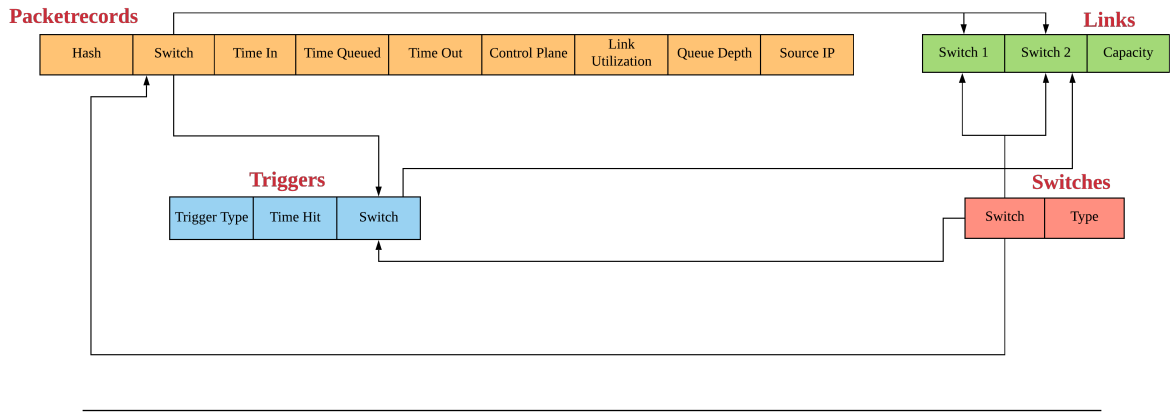

FIGURE 3.2: Database Schema

3.2.1 The Switches table

This table stores two columns described below:

- Switch - The identifier of the switch, a string ID that is used to uniquely identify each switch
- Type - This column specifies the type of the switch. For the purpose of our demonstrations, Switch type could either be a ToR switch (top of rack) or a non-ToR switch.

3.2.2 The Triggers table

This table stores three columns described below:

- Trigger Type - an identifier for the type of trigger, which can indicate what was the cause of the fault in the network. As an example, the switch might be configured to create a trigger when the queue depth at a switch exceeds a certain number.
- Time Hit - the time at which the trigger condition was raised, in the format of a 64 bit representation. The most significant bit in the decimal representation of the 64 bit number would denote the number of seconds passed, while the remaining digits would denote the precision in nanoseconds. For example, the number 1435756194 (stored as a 64 bit number) would mean 1.435756194 seconds.
- Switch - the identifier of the switch that raised the trigger.

3.2.3 The Links table

This table stores three columns described below:

- Switch 1 - the identifier of the source switch for the link.
- Switch 2 - the identifier of the destination switch for the link.
- Capacity - the link capacity in Gbps.

3.2.4 The Packetrecords table

This table stores the following 9 columns corresponding to each packet, as described below. There exists such a tuple for every packet that leaves any switch in the network.

- Hash - the hash of a packet, to identify it uniquely
- Switch - the identifier of the switch the packet has just left.
- Time In - the time at which the packet enters the switch, in nanosecond precision.
- Time Queued - the duration for which the packet is queued in the switch.
- Time Out - the time at which the packet leaves the switch, in nanosecond precision.
- Control Plane - an identifier for the version of control plane that is presently running in the switches.
- Link Utilization - the current egress link utilization of the switch, as measured in the data plane of the switch itself. This field is later used to corroborate our calculations for egress throughput.
- Queue Depth - the current length of the queue in the switch.
- Source IP - the source IP of the packet that the 9-tuple represents.

Chapter 4

Chapter Title Here

4.1 Main Section 1

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aliquam ultricies lacinia euismod. Nam tempus risus in dolor rhoncus in interdum enim tincidunt. Donec vel nunc neque. In condimentum ullamcorper quam non consequat. Fusce sagittis tempor feugiat. Fusce magna erat, molestie eu convallis ut, tempus sed arcu. Quisque molestie, ante a tincidunt ullamcorper, sapien enim dignissim lacus, in semper nibh erat lobortis purus. Integer dapibus ligula ac risus convallis pellentesque.

4.1.1 Subsection 1

Nunc posuere quam at lectus tristique eu ultrices augue venenatis. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Aliquam erat volutpat. Vivamus sodales tortor eget quam adipiscing in vulputate ante ullamcorper. Sed eros ante, lacinia et sollicitudin et, aliquam sit amet augue. In hac habitasse platea dictumst.

4.1.2 Subsection 2

Morbi rutrum odio eget arcu adipiscing sodales. Aenean et purus a est pulvinar pellentesque. Cras in elit neque, quis varius elit. Phasellus fringilla, nibh eu tempus venenatis, dolor elit posuere quam, quis adipiscing urna leo nec orci. Sed nec nulla auctor odio aliquet consequat. Ut nec nulla in ante ullamcorper aliquam at sed dolor. Phasellus fermentum magna in augue gravida cursus. Cras sed pretium lorem. Pellentesque eget ornare odio. Proin accumsan, massa viverra cursus pharetra, ipsum nisi lobortis velit, a malesuada dolor lorem eu neque.

4.2 Main Section 2

Sed ullamcorper quam eu nisl interdum at interdum enim egestas. Aliquam placerat justo sed lectus lobortis ut porta nisl porttitor. Vestibulum mi dolor, lacinia molestie gravida at, tempus vitae ligula. Donec eget quam sapien, in viverra eros. Donec pellentesque justo a massa fringilla non vestibulum metus vestibulum. Vestibulum in orci quis felis tempor lacinia. Vivamus ornare ultrices facilisis. Ut hendrerit volutpat vulputate. Morbi condimentum venenatis augue, id porta ipsum vulputate in. Curabitur luctus tempus justo. Vestibulum risus lectus, adipiscing nec condimentum quis, condimentum nec nisl. Aliquam dictum sagittis velit sed iaculis. Morbi tristique augue sit amet nulla pulvinar id facilisis ligula mollis. Nam elit libero, tincidunt ut aliquam at, molestie in quam. Aenean rhoncus vehicula hendrerit.

Appendix A

Jain's Fairness Index

A description of the Jain's Fairness Index

$$\mathcal{J}\left(x_1, x_2, \ldots, x_n\right) = \frac{\left(\sum_{i=1}^{n} x_i\right)^2}{n \cdot \sum_{i=1}^{n} x_i^2} = \frac{\overline{\mathbf{x}}^2}{\overline{\mathbf{x}^2}} = \frac{1}{1 + \widehat{c_{\mathrm{v}}^2}} \tag{A.1}$$

Raj Jain's equation rates the fairness of a set of values where there are n users, each having an associated throughput value. This metric ranges from 1/n in the worst case to 1 in the best case. In this thesis, the metric is normalized to a scale of 0 to 1 before comparing it to thresholds as mentioned in the thesis sections.

Appendix B

Grafana

Grafana (Figure B.1) is a multi-platform open source analytics and interactive visualization software available since 2014. It provides charts, graphs, and alerts for the web when connected to supported data sources. It is expandable through a plug-in system. End users can create complex monitoring dashboards using interactive query builders.

As a visualization tool, Grafana is a popular component in monitoring stacks, often used in combination with time series databases such as Prometheus and Graphite; monitoring platforms

FIGURE B.1: Grafana

such as Sensu, Icinga, Zabbix, Netdata, and PRTG; SIEMs such as Elasticsearch and Splunk; and other data sources.

In this thesis, Grafana is used extensively for graphing the visualizations made using SQL queries using its MySQL plugin. As part of my thesis, I have written a library for automating dashboard creation in Grafana through the provided HTTP API documentation so that the system can autogenerate relevant dashboards and graphs after having analysed the MySQL data using SQL queries.

A link to the GitHub repository with the library is provided:
`https://www.github.com/sankalp-sangle/MySQL-Grafana.git`
A link to the GitHub repository for Grafana is provided:
`https://www.grafana.com`

Bibliography

[1] D. S. Alexander et al. “The switchware active network architecture”. In: (1998).

[2] Pat Bosshart et al. “Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN”. In: *SIGCOMM 2013 - Proceedings of the ACM SIGCOMM 2013 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*. 2013, pp. 99–110.

[3] Pat Bosshart et al. “P4: Programming protocol-independent packet processors”. In: (2014).

[4] *Grafana, The Open Visualization Platform*. URL: https://grafana.com.

[5] Sushant Jain et al. “B4: Experience with a globally-deployed software defined WAN”. In: *SIGCOMM 2013 - Proceedings of the ACM SIGCOMM 2013 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*. 2013.

[6] Pravein Govindan Kannan, Raj Joshi, and Mun Choon Chan. “Precise Time-synchronization in the Data-Plane using Programmable Switching ASICs”. In: (2019).

[7] Nick McKeown et al. “OpenFlow: enabling innovation in campus networks”. In: *ACM SIGCOMM Computer Communication Review* (2008).

[8] Justin Meza et al. “A large scale study of data center network reliability”. In: (2018).

[9] Vibhaalakshmi Sivaraman et al. “Heavy-hitter detection entirely in the data plane”. In: *SOSR 2017 - Proceedings of the 2017 Symposium on SDN Researchs*. 2017, pp. 164–176.

Turnitin Originality Report Pending....