Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Dept. of Information Security and Communication Technology

Emil Telstad
Mats Ove Sannes

# Machine Learning for Multi-Source Analysis

Master's thesis in Communication Technology
Supervisor: Otto J. Wittner
Co-supervisor: Olav Kvittem
June 2021

NTNU
Norwegian University of Science and Technology

Emil Telstad
Mats Ove Sannes

# Machine Learning for Multi-Source Analysis

Master's thesis in Communication Technology
Supervisor: Otto J. Wittner
Co-supervisor: Olav Kvittem
June 2021

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Dept. of Information Security and Communication Technology

NTNU
Norwegian University of
Science and Technology

**NTNU – Trondheim**
Norwegian University of
Science and Technology

# Machine Learning for Multi-Source Analysis

**Emil Telstad**
**Mats Ove Sannes**

Submission date: June 2021
Supervisor: Otto J. Wittner, NTNU/Uninett
Co-supervisor: Olav Kvittem, Uninett

NTNU – Norwegian University of Science and Technology
Department of Information Security and Communication Technology

**Title:** Machine Learning for Multi-Source Analysis

**Student:** Emil Telstad & Mats Ove Sannes

**Problem Description:**

The Internet has expanded around the world faster than any other technology. It has transformed from the old circuit-switched duplex telecommunication networks into a global packet-switched network. Research within many different fields of technology, from routing protocols to enhanced link transmission with fibre-optic networks, has increased the network's capacity by many orders of magnitude. As a result, it has raised our expectations for applications we use every day. To manage the complex and heterogeneous traffic mixture while maintaining dependability, network providers must continuously monitor their devices to detect faulty behaviour and optimise network configuration.

Uninett, the national research IP network operator in Norway, collects monitoring data of different types from all their routers in the national backbone network. Uninett also runs active measurements by inducing probe traffic into their network. These datasets have the potential to reveal undesired network behaviour and thus will be the primary focus for the master thesis. Analysing an extensive collection of datasets with the goal of early discovery and sub-optimal configurations for devices is a challenge. Machine Learning (ML) systems have proven to handle pattern-matching in large data sets well. Hence an investigation into what it takes to apply and what might be gained by applying ML techniques to Uninett's monitoring datasets is of interest.

The project objective will be to investigate to what extent ML algorithms may enhance Uninett's monitoring system compared to traditional statistical analysis. A selection of available ML algorithms will be tuned and applied in novel manners to one or more of Uninett datasets to determine which algorithm is more suitable. Different analysis approaches will be looked into, e.g. anomaly detection, forecasting and root cause explanation of disruptive behaviour. Preparation of datasets will be a part of the project. Awareness of the sources' data quality will be essential to determine the discriminative abilities of identified features and the overall applicability of ML on datasets. Novel overall setups with the potential to generate output of value for the engineers in charge of the network operation centre will be the overall target.

**Date approved:** 2021-01-14

**Supervisor:** Otto J. Wittner, NTNU/Uninett

## Abstract

The Internet keeps expanding and creates a higher performance demand due to progressively emerging digital services. The Internet Service Providers (ISPs) closely monitor their network infrastructure to provide stable and reliable networks. In an attempt to further improve the dependability, a collaboration project dubbed "Dragonlab", where Uninett is a contributor, has been initiated to measure the end-to-end quality of network traffic. The probes in the network generate a vast amount of monitoring logs. By analysing this data using traditional methods, Uninett has uncovered an accumulation of micro outages that could indicate network issues. This thesis investigates to what extent ML can produce valuable output to contribute in Uninett's monitoring system. The thesis follows a design science research methodology, where we divide the iterating cycles into several implementation approaches.

These approaches consists of investigating Uninett's collected and analysed data to gain knowledge of the current solution in production. We apply Kibana's ML-based anomaly detection to find anomalies in the measured network delay, and conduct root cause analysis on these findings. The other approaches manually inspect multiple resources whilst evaluating possible ML solutions. The results show little information of interest in the datasets and low potential for ML. Most features in the datasets are derived from a single metric, nodal delay, and contribute no context to observed deviations. Uninett's monitoring system, as of today, is primarily applicable in statistical analysis and threshold methods for problem discovery. Proceeding with a statistical analysis approach, we discover correlations between streams of packets on different paths. These results indicate that unwanted events affect multiple parts of the infrastructure. This discovery can be used in combination with other sources to determine root causes in the future.

The conclusion deems ML not profitable for Uninett at the time being. ML would introduce unnecessary complexity and would require expertise to develop and maintain. We suggest expanding on their existing data analysis by combining multiple sources of information and label known root causes. Improving their log-formatting and developing an atomic methodology while doing so is highly recommended. They will then have easily accessible and dynamic resources ready for numerous analysis methods in the future. If able to reliably combine multiple sources of information to labelled events, a reevaluation of ML can be conducted.

## Sammendrag

Internett fortsetter å utvide seg, og brukere forventer stadig bedre ytelse ettersom nye og bedre digitale tjenester blir tilgjengelig. Internettleverandører overvåker nettverksinfrastrukturen sin nøye for å opprettholde stabile og pålitelige nettverk. I et forsøk på å forbedre påliteligheten ytterligere, har et samarbeidsprosjekt kalt "Dragonlab", der Uninett er en bidragsyter, blitt initiert for å måle ende-til-ende kvaliteten på nettverkstrafikk. Målenodene i nettverket genererer enorme mengder overvåkingslogger. Ved å analysere disse dataene med tradisjonelle metoder, har Uninett avdekket en opphopning av mikrobrudd som kan indikere nettverksproblemer. Denne oppgaven undersøker i hvilken grad maskinlæring (ML) kan resultere i verdifull informasjon for å bidra i Uninetts overvåkingssystem. Oppgaven følger en designvitenskapelig forskningsmetodikk, der vi har delt iterasjonssyklusene inn i flere tilnærminger med ulike metoder og implementasjoner.

Tilnærmingene til oppgaven består av å undersøke Uninetts innsamlede og analyserte data for å få innsyn i de nåværende løsningene i systemet. Vi bruker Kibanas ML-baserte anomaliedeteksjon for å finne uregelmessigheter i den målte nettverksforsinkelsen, og årsaksanalyserer disse funnene. De andre tilnærmingene inspiserer flere ressurser manuelt mens muligheten for ML blir evaluert. Resultatene viser lite informasjon av interesse i datasettene og lavt potensial for ML. De fleste beregningene i datasettene stammer fra én enkel måling, pakkeforsinkelse, og bidrar ikke med noe kontekst til observerte avvik. Uninetts overvåkingssystem, per i dag, er primært passende for statistisk analyse og terskelmetoder for å oppdage problemer. Vi fortsetter derfor med en statistisk analysetilnærming og oppdager korrelasjon mellom strømmer av pakker på forskjellige ruter. Disse resultatene indikerer at uønskede hendelser gjenspeiles på flere deler av infrastrukturen. Oppdagelsen kan brukes i kombinasjon med andre kilder for å fastslå årsakene i fremtiden.

Konklusjonen anser ML som ikke lønnsomt for Uninett på et nåværende tidspunkt. ML vil innføre unødvendig kompleksitet og vil kreve ekspertise for å utvikle og vedlikeholde. Vi foreslår å utvide deres eksisterende analyse ved å kombinere flere ulike datasett. I tillegg kan de ta markere seg enkelte tilfeller med kjente årsaker. Det anbefales sterkt å forbedre loggformateringen de bruker i dag, samt bryte ned loggene i mindre enkeltstående komponenter. De vil da ha lett tilgjengelige og dynamiske ressurser klare for ulike analysemetoder i fremtiden. Hvis de klarer å kombinere flere ulike informasjonskilder, kan ML revurderes.

# Preface

This thesis was completed summer of 2021 and draws an end to our course of study, Master of Science in Communication Technology, at the Norwegian University of Science and Technology. The project is conducted in cooperation with Uninett and is based on the preliminary project thesis written during fall 2020. The master thesis's main objective is to investigate to what extent ML algorithms can handle Uninett's collection of monitoring data and generate valuable output for their network operation centre. The end goal is to help understand losses in data traffic and possibly mitigate them to increase reliability in the network.

Reading through the list of available projects, solving a real-life problem seemed intriguing to us. The initial problem description, given by the ISP Uninett, raised an interesting issue of lost traffic. We chose this project determined to contribute to a solution with meaning and usage. Both of us have studied networking, but considering that neither of us have experience in ML, the project has been challenging and a great learning experience. It required a great deal of studying related work and theoretical practices. This thesis will provide an overview of basic concepts of ML and anomaly detection and how we implemented our solutions to detect abnormal behaviour in the network.

This report assumes the reader has some background knowledge in computer networking, machine learning, and statistical analysis. The theory is lectured in the following courses at Norwegian University of Science and Technology (NTNU): TTM4175 - Introduction to Communication technology, TMA4245 - Statistics, TTM4180 - Applied Networking, TTM4105 - Access and Transport Networks, TTM4150 - Internet Network Architecture, and TDT4300 - Data Warehousing and Data Mining. The reader is also assumed to have basic knowledge of programming.

Trondheim, Wednesday 30$^{\text{th}}$ June, 2021

Emil Telstad

Mats Ove Sannes

## Acknowledgements

We want to express our gratitude to our supervisor, Otto J. Wittner, and our contact person in Uninett, Olav Kvittem, for all support and guidance during the project. They provide great insight into the problem at hand and contributed ideas on how to approach them. We also want to thank professor Yuming Jiang for his valuable insight into ML and state-of-the-art solutions. He contributed to the project's progress in Chapter 8.

# Contents

# List of Figures

# List of Tables

# List of Acronyms

**BGP** Border Gateway Protocol.

**CRUDE** Collector Real-Time UDP Emitter.

**CS** Compressed Sensing.

**CSV** Comma Separated Values.

**DL** Deep learning.

**DST** Destination.

**GB** GigaByte.

**GDPR** General Data Protection Regulation.

**GUI** Graphical User Interface.

**HTM** Hierarchical Temporal Memory.

**ICT** Information and Communication Technology.

**IP** Internet Protocol.

**ISP** Internet Service Provider.

**MB** MegaByte.

**ML** Machine Learning.

**ms** milliseconds.

**NAB** Numenta Anomaly Benchmark.

**NaN** Not a number.

**NGU** Norges geologiske undersøkelse.

**NREN** National research and education network.

**NTNU** Norwegian University of Science and Technology.

**NTP** Network Time Protocol.

**PCP** Principal Component Pursuit.

**RCA** Root Cause Analysis.

**RCF** Random Cut Forest.

**RED** Random Early Detection.

**RRCF** Robust Random Cut Forest.

**RSA** Cryptographic algorithm by Ron **R**ivest, Adi **S**hamir and Leonard **A**man.

**RTT** Round-Trip Time.

**RUDE** Real Time UDP Emitter.

**SNMP** Simple Network Management Protocol.

**SRC** Source.

**SSH** Secure Shell.

**TCP** Transmission Control Protocol.

**UDP** User Datagram Protocol.

# Chapter 1
# Introduction

The current chapter includes the thesis motivation and an introduction to Uninett's monitoring system. Additionally, the projects research questions, objectives, contribution, and limitations are presented. Finally, an outline of the thesis is included, providing an overview of the following chapters.

## 1.1 Motivation

In the last two centuries, the demand for communication technology has multiplied as more and more digital services are emerging. It has been estimated that 25 billion devices would be connected to the Internet by 2020 [KR16]. Concerning the popularity of new applications and internet trends, it is imperative for an ISP to meet the demands in terms of reliance and service quality. In a market with several competitors, unwanted quality degradation can become costly in revenue for providers. In these modern times, customers are flooded with offers to change service providers, and therefore a provider's reputation among its customers is crucial. The users would not appreciate neither delay nor outages. To prevent this, the ISP must monitor their networks to detect and hopefully mitigate such events. The modern network architecture is a packed-switched "best-effort" service-based network consisting of millions of nodes. Throughout the evolution of Internet, routing protocols, buffer management and scheduling have been added to direct traffic through a web of possible paths. Based on the complexity of large networks, service providers are looking for new ways to help them maintain their service quality.

Uninett is a large Norwegian Information and Communication Technology (ICT) infrastructure company and network provider for the research and education sector [Unic]. They offer services within cyber-security and is a supplier of the login service Feide used by campuses nationwide. Maintaining such an extensive network is a difficult task and requires monitoring to detect faulty behaviour. In an attempt to improve their security and dependability, Uninett's clients have allowed them

to install monitor-nodes within their infrastructures. In essence, these nodes are machines placed adjacent to the routers connected to an optical fibre network. These nodes receive a copy of all traffic passing through the adjacent router. Initially, Uninett used this data for intrusion detection with deep packet inspection. However, they have since been motivated to utilise these nodes further by enabling probe-based measurements to uncover unwanted behaviour in their network. With this approach, Uninett has discovered a lot of "micro-outages", labelled as "gaps", of undetermined causes [TS20]. In September 2019, this accumulated to 1.5 hours lost in a single month [Kvi19] [1]. In addition, a cooperative initiative between three National research and education networks (NRENs), called Dragonlab, has been established. Uninett is one of the contributors, and the current goals of the confederation are to measure end-to-end quality and micro dependability of routing, interact with network operations to diagnose problems, and promote data sharing with research to help understand the Internet [Unib].

The underlying causes of these gaps could evolve into more detrimental issues leading to outages. Internet outages are costly, and an outage preventing users to connect to data centres can cost on average 5000$ per minute [ABM+18]. A problem of emerging incidents in network traffic flow is to determine the causes. The answer to mitigating such gaps may be found in the field of ML. With ever-growing datasets entering the domain of *Big Data*, traditional methods for analysing the data can become complex. ML on the other hand tends to improve its performance as the size of the dataset grows.

ML is about extracting knowledge from data [MG16]. It is the most popular part of sub-field Artificial Intelligence, and the use of ML has quickly grown over the last 20 years. As stated in the preliminary project [TS20], a general approach to ML consists of a data collection phase, feature engineering phase, building/training a model, apply the model, and finally validating the results. A general rule of thumb says that approximately 80% of the time is spent working with and prepare the data, whereas the implementation, application, and validation accounts for the remaining 20%. Sometimes, the feature engineering phase can even implement machine learning itself in order to detect anomalies that can be used in a future model. This strategy is relevant in this project as a means to drive the investigation further [TS20].

Detecting ongoing network outages is essential to qualitatively and quantitatively understand the type, scope, and consequences of a disruptive event and timely activate mitigation and remediation activities [ABM+18]. Today, Uninett use threshold methods to uncover gaps and jitter in the network. This gives only a partial picture of the paths' states and does not determine the cause of the incident. By investigating a broader scope, there might be discovered patterns between gaps and

[1] Details about this result is discussed in Section 8.3

other monitoring data sources. In order to detect anomalies and better understand current data, a variety of approaches to ML will be examined.

## 1.2 Uninett's Probe Network

The active probing consists of inducing a constant stream of packets through the network. The distributed machines have been instructed to simulate a continuous flow of traffic to specific targets in order to generate data with representative coverage of the topology. The selected machines can be found within the Norwegian intranet and the global infrastructure Dragonlab.

An overview of the topologies can be seen in Figures 1.1 and 1.2. The screenshots have been captured from a web tool developed by Uninett. Interested readers can explore this tool at [Unia]. The tool visualises their stream analysis and displays a given stream's state at a certain point in time. The colouring does not serve a purpose for these illustrations and can be ignored.

Figure 1.1 shows the numerous streams between machines in Uninett's intranetwork. Note that only realistic locations of streams' endpoints are shown. The coloured graphs between nodes are for illustration only, and does not depict traversed paths by streams. The circular symbols with numbers in them symbolises groups of machines. For example, there are eight nodes around Trondheim. Note, there are other nodes further north than shown in this figure.

Figure 1.2 shows the topology in Dragonlab. There are connections to Europe, North America, South America, Australia and New Zealand.

The data in question is generated by a RUDE/CRUDE software [JLj], a simple program that generates network traffic between pairs of Source (SRC) and Destination (DST). (C)RUDE stands for (Collector) Real-Time UDP Emitter, where User Datagram Protocol (UDP) is a transfer protocol without any guarantees for arrival contrary to Transmission Control Protocol (TCP). The designated monitoring machines form a large set of RUDE/CRUDE pairs. Uninett has experimented with several packet frequencies between them and settled for 100 p/s as a good trade-off between informational gain and resource usage. Increasing the frequency makes the system susceptible to glitches in the software, and could result in false positives of disruptive behaviour for succeeding analysis. On the other hand, reducing the frequency collects less information, resulting in datasets less likely to capture rapid real-word events.

There are many components included in this system, both hardware and software introduce an uncertainty. A feature of routers, for example, is to drop random packets when it experience high load of traffic. The congestion avoidance mechanism

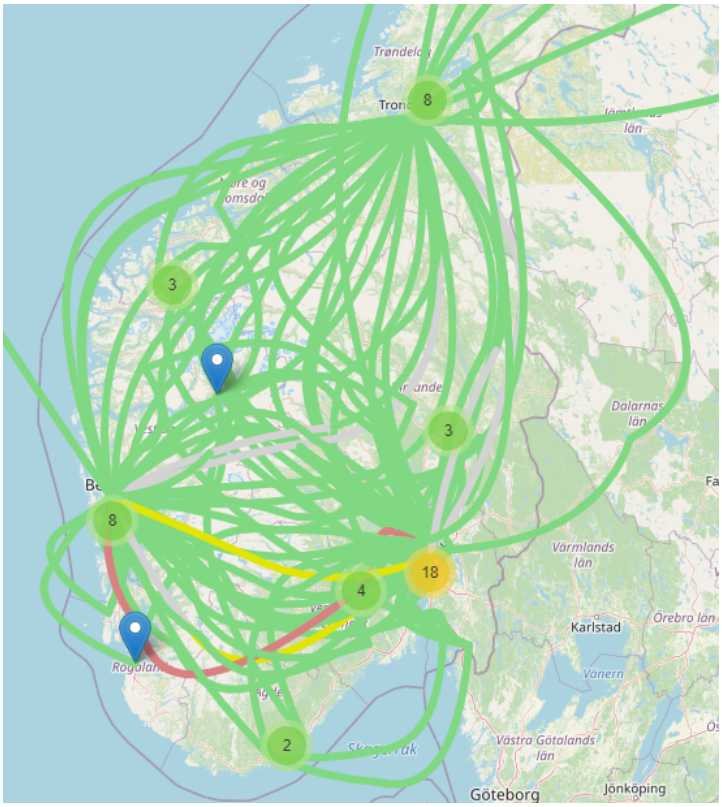



Figure 1.1: Uninett topology.
For interested readers, this map is available at [Unia].

is known as Random Early Detection (RED) [FJ93] and accompanies network transport protocols e.g. TCP. The routers intent is to indirectly warn the transport protocols that they are about to reach maximum capacity, and that they should reduce traffic. We must emphasise that this uncertainty permeates the project as a whole.

The continuous network monitoring has resulted in several resources available to us. To enable future detailed studies and analysis, they have chosen to store all historical logs of network behaviour. At this point in time, they possess over a decade worth of detailed data. We will look more into the raw CRUDE data in Chapter 5.

Uninett has further analysed CRUDE data. Written scripts run separately on each machine to prepare and process raw data received, e.g. mapping them to timestamps. They also register gaps and jitter, which will be introduced in Chapters 6 and 7. Together, all the nodes generate continuously increasing datasets. The

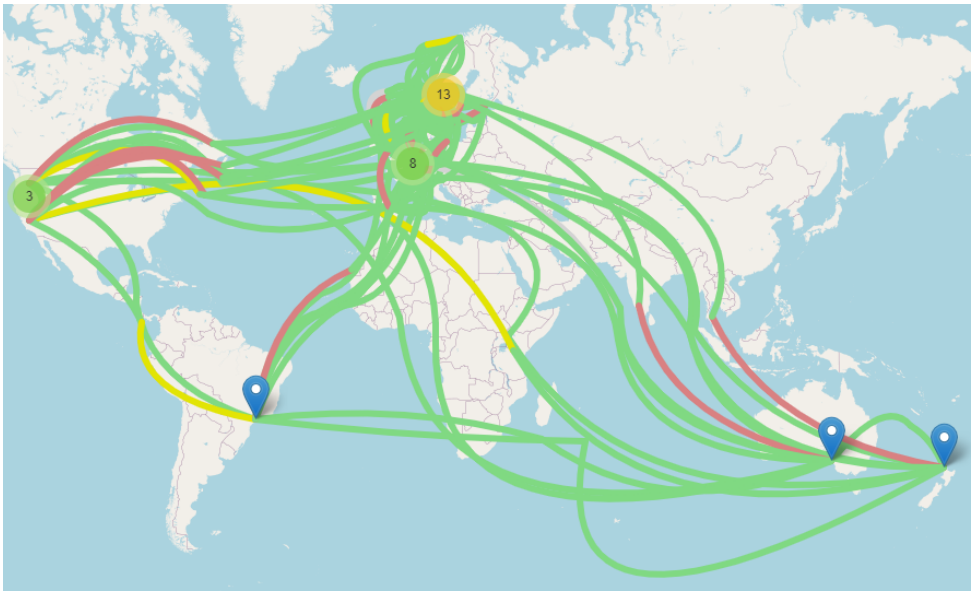



Figure 1.2: Dragonlab topology.
For interested readers, this map is available at [Unia].

centralised server, called IOU2, hosts Elasticsearch and Kibana for visualisation. And it is with this approach, Uninett discovered the aforementioned gaps of undetermined causes.

Figure 1.3 is an overview of how the multiple datasets to be examined relate to the monitoring system. This figure is a simplification of the current system. There is included an example of a single RUDE/CRUDE pair from trondheim-mp (Norway) to auckland-mp (Australia). As shown by the arrows, there is a flow of UDP packets and traceroutes with different frequencies through a path in a simplified network of four routers (R1, R2, R3, R4). The receiving end, auckland-mp, stores the raw data and analyses the stream. At the end of the day, raw and processed data are sent to the main server IOU2, which in turn feeds Elasticsearch with new data. Only prepared datasets such as jitter, gap and traceroute are included in Kibana. Our contribution comprises extracting data from IOU2, reading and analysing the multiple datasets, and evaluating them with respect to ML. If a dataset shows promise, an implementation of ML will be inquired.

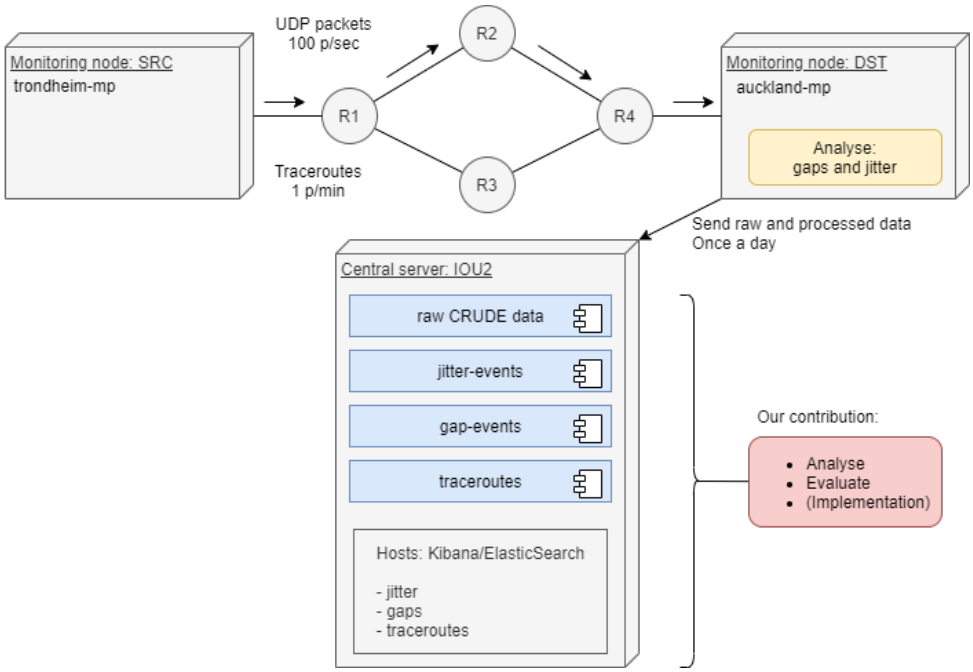



Figure 1.3: Active monitoring system overview.

## 1.3 Research Questions

The following research questions was formulated based on the problem description and the conducted background research:

1. Which requirements need to be fulfilled to facilitate machine learning?
2. Can Uninett utilise machine learning in its already existing network monitoring system?
3. How can Uninett improve its network monitoring to determine root causes of faults better?

## 1.4 Objectives

The following objectives have been defined as guidelines in the thesis work:

- Review related work on time series forecasting, anomaly detection using machine learning, and root cause explanation.
- Evaluate the quality of available datasets from Uninett.
- Preprocess the chosen data to prepare for analysis.
- Implement and evaluate suggested approaches to obtain results for comparison to simpler solutions.
- Evaluate the data with respect to machine learning to determine viability of future real-life implementations.

By completing these chosen objectives, we research to what extent a machine learning solution would produce useful information to a service provider monitoring its network. In addition, we will gain deeper insight into the implementation of such a solution and the challenges it may entail.

## 1.5 Contribution

The thesis is mainly a test of concept to determine if machine learning is a possible solution to enhance monitoring and analysis of network traffic. Based on the combination of such an experiment and the report, the main contribution of this thesis will be:

- An investigation into existing machine learning algorithms for anomaly detection, time series forecasting, and root cause analysis.
- A review of the potential of machine learning as a solution to enhance service providers quality of service.
- An evaluation of advantages and challenges of machine learning anomaly detection compared to less complex solutions.

The project contributes to a positive development in the research topic by evaluating advanced technological solutions to enhance ISPs quality of service. The solutions and techniques in this thesis can be applied on datasets from domains other than ICT as well.

## 1.6 Scope

The general purpose of the study is to investigate if ML can reveal patterns in the observed gaps. The project targets explicitly raw CRUDE data, which has been used to detect these gaps, and other derived datasets generated as an attempt to understand them. This data has never been evaluated regarding ML, so the thesis' scope is the initial survey to uncover their format and content.

Cross-matching of datasets from different sources is not performed in this project. It is an interesting topic but has already been attempted earlier by Uninett. One reason for not pursuing this approach is the limited duration of the project; more reasons are elaborated in Section 10.2.

## 1.7 Outline of Thesis

- ◇ Chapter 1: Presents the motivation for the project, the research questions, the main objectives at focus, the contribution of the project, and the limitations related to the work.
- ◇ Chapter 2: Introduces background theory within ML, anomaly detection, ICT-infrastructure, and root cause analysis.
- ◇ Chapter 3: Presents related work within the research field.
- ◇ Chapter 4: Presents the research methodology used in the thesis.
- ◇ Chapter 5: Presents the approach of investigating Uninett's CRUDE datasets.
- ◇ Chapter 6: Presents the approach of testing Kibana anomaly detection on the datasets.
- ◇ Chapter 7: Presents the approach of investigating Uninett's dataset and analysis of Gap-events.
- ◇ Chapter 8: Presents the approach of stream cross comparisons.
- ◇ Chapter 9: Presents the approach of performing a root cause analysis.
- ◇ Chapter 10: Presents a final discussion.
- ◇ Chapter 11: Presents the thesis conclusion.
- ◇ Appendix A, B, E, F, G: Contains additional figures to Chapters 6, 9, 7, and 8.
- ◇ Appendix C: Contains the projects environment setup.
- ◇ Appendix D: Contains utilised programming scripts.

# Chapter 2
# Theoretical Background

The following sections will give an overview of the theoretical background for this master thesis, including time series, machine learning, anomaly detection, and root cause analysis.

## 2.1 The ICT Infrastructure Domain

Internet is described as a 'network of networks'. It relies on a physical infrastructure consisting of a network of nodes, interconnecting computers and users of the Internet. These nodes are routers or switches, with the main objective of guiding data packets through the network from source to destination. Even though modern technology is approaching lightning speed, end-to-end transportation over physical distances still takes a certain amount of time.

Delay in a network consists of several types of delay. In equation 2.1, the different types affecting a packet in transit are added up to a total nodal delay ($d_{nodal}$). When a packet arrives at a node, the header is examined to determine the next destination in the path. The time it takes to perform this task is called *processing delay* ($d_{proc}$). When the packet is processed, the router directs it to a queue to be transmitted into the link. This adds *queuing delay* ($d_{queue}$). The length of the queue at the moment the packet arrives determines the duration of the delay. It can vary from packet to packet, and in a situation where ten packets arrive simultaneously, the first packet transmitted will suffer no queuing delay. In contrast, the last packet will experience queuing delay [KR16]. When the packet is ready to be transmitted into the link, the *transmission delay* ($d_{trans}$) is the time it takes to execute. Once the packet's bits are pushed into the link, it propagates to the next router. The time required to propagate the link, denoted *propagation delay* ($d_{prop}$), depends on its physical medium and geographical distance.

$$d_{nodal} = d_{proc} + d_{queue} + d_{trans} + d_{prop} \tag{2.1}$$

With increased traffic intensity in a network, several issues may emerge. Packets may arrive faster than they can be processed and transmitted. As a result, queues will fill up, and arriving packets may be dropped or lost because of limited queue capacity [KR16]. The increase in data could congest the links and call for rerouting to an alternative, possibly longer path, to redistribute the network load. Additionally, external factors such as power outages, stormy weather or roadwork can disable network nodes and cables, resulting in connection loss.

### Jitter

Jitter, also called stuttering, is the average of differences between a sequence of values. This thesis investigates network packets where a regular interval is expected, and jitter becomes relevant in the gap analysis in Chapter 7. Higher jitter could indicate issues in the network. Given an example of 5 packets with nodal delay {2,4,1,1,3}, jitter is calculated as follows: $(|2-4|+|4-1|+|1-1|+|1-3|)/(5-1) = 1.4$. This is only a simplified example. The actual system in place calculates jitter on a continuous stream of packets and utilises the method defined in RFC3550 [SCFJ03].

### Gaps

*Gaps - a missing segment in stream data* [KR16]. Micro-outages, labelled as gaps, are defined by Uninett as the time between correctly received packets in order and when five packets are received in correct order again. The time lost during gaps in Uninett's system often ranges from 50 to 200 ms, but more severe cases do occur. These events are easy to discover, and have already been identified and stored as a dataset of gap-records. Only gaps where five or more packets were lost are included in this dataset. Gaps become relevant in Chapter 7.

### Traceroute

Traceroute is a tool for computer networks to trace traffic routes through the network to their intended destination. There are sent a certain amount of packets to each node in the path, called a hop. Traceroute records the time taken for each of these hops, know as Round-Trip Time (RTT). It is therefore used to determine the response delay and available pathways across the nodes in the network. If the sent packets are lost more than two times, the connection is lost. Figure 2.1 displays an example of traceroute results. It consists of a timestamp for when the trace was performed, a destination Internet Protocol (IP) address, and lines of information about the performed hops. The first bracket in the line represents the hop number based on the node's position in the path, followed by the node IP address. The six consecutive values are the RTT, measured in milliseconds (ms), of the packets sent to the node. Lines with a star symbol indicated no response from the node.

Traceroutes are often used to identify problems in the network, such as bottlenecks and points of failure in a route. However, there is some inaccuracy associated with traceroutes, seeing that a protective firewall can block required messages to a node. Traceroute data is one of the datasets collected by Uninett's Dragonlab infrastructure and is utilised to conduct root cause analysis in Chapter 9.

```
1614898808 starttime 00:00:08
traceroute to 128.39.65.26 (128.39.65.26), 30 hops max, 60 byte packets
 1  130.216.51.254  1.821 ms  1.867 ms  1.711 ms  1.643 ms  1.540 ms  1.421 ms
 2  172.18.0.54  1.967 ms  1.931 ms  1.813 ms  1.745 ms  1.615 ms  1.535 ms
 3  * * * * * *
 4  130.216.252.161  3.302 ms  1.887 ms  2.154 ms  2.986 ms  2.627 ms  2.543 ms
 5  210.7.39.177  1.226 ms  1.111 ms  1.285 ms  1.177 ms  1.131 ms  1.207 ms
 6  207.231.240.33  132.547 ms  132.447 ms  132.344 ms  133.041 ms  132.941 ms  132.715 ms
 7  207.231.240.8  132.757 ms  132.950 ms  133.146 ms  132.758 ms  132.948 ms  133.054 ms
 8  162.252.70.173  165.753 ms  165.522 ms  165.663 ms  165.454 ms  165.455 ms  167.117 ms
```

Figure 2.1: Example of traceroute results.

## 2.2 Time Series

*A time series is a set of observations $x_t$, each recorded at a specific time t* [BD02]. Time series has a wide area of use and can be found in several research areas such as medicine, social science and economics [LA15]. Each point in the sequence typically consists of two items - a timestamp and an associated value. Time series analysis comprises statistical methods for modelling an ordered sequence of observations to extract meaningful statistics [Mad07]. The model can reveal trends, cycles or other pattern and approximate the probable development of the time series by either prediction or forecasting.

Depending on the number of variables, time series can be split into two types. A series with only one variable is a *univariate* time series, while more than one variable is known as a *multivariate* time series. A time series is *continuous* if the observations are recorded continuously over a given time interval. A *discrete* time series consists of observations recorded at fixed time intervals.

A time series is *stationary* if the statistical properties of the series, mean, variance and covariance, do not vary with time. In a *non-stationary* time series however, any of the the properties can alter over time, individually or together. A series is

non-stationary if it contains trends or seasonality but stationary if only cycles are present.

### 2.2.1 Time Series Components

Time series can be decomposed into four main variations that affects them: *seasonal variation, trend, other cyclic variation and irregular fluctuations* [Cha00]. Seasonality is observed as short-term variation occurring due to seasonal factors. A trend is present when there is a steady decrease or increase in the series over a longer period. It can be loosely defined as 'long-term change in the mean level' [Cha00]. Cyclic variation is a medium-term variation caused by events that repeat at irregular intervals. The reoccurring shutdowns during the global crisis caused by the COVID-19 pandemic is an example of cyclic variation. The last category covers unpredictable factors and is described as irregular fluctuations. It incorporates all variations not included in trend, seasonality, or cyclicity. Irregular fluctuation can be caused by incidents, e.g. natural disasters, which makes them difficult to forecast.

The components can be combined in two different ways, based on the trend in the data. The *additive model*, where the time series is the sum of the component, can be written as $Y_t = T_t + C_t + S_t + I_t$. Whereas $T_t$ is the trend, $C_t$ is cyclical variation, $S_t$ is seasonal variation, and $I_t$ is irregular fluctuations. Alternately, the *multiplicative model* can be applied by multiplying the components; $Y_t = T_t \times C_t \times S_t \times I_t$.

### 2.2.2 Cleaning the Data

The initial inspection of the data is done to determine the quality and decide if there is a need to modify them. This action is called *cleaning the data* and can include modifying outliers, identifying and correcting obvious errors, and filling in any missing observations [Cha00]. It can be difficult to distinguish error outliers from authentic outliers in the data. Still, the use of a time plot should help uncover any abnormalities such as outliers and discontinuities. A found background knowledge of the problem is essential in deciding how to clean the data [Cha00].

### 2.2.3 Time Series Analysis

The main objectives of time series analysis are to describe the data using summary statistics or graphical models. It aims to find a suitable statistical model describing the data-generating process, estimating the future values of the series by forecasting, and consequently facilitate for the analyst to take action to control a given process [Cha00]. Visualisation, such as time plots, is very useful in a time series analysis, and aid the human eye to spot patterns or anomalies. A general approach to time series analysis is to examine the main characteristics of the graph, searching for trends,

a seasonal component, any apparent sharp changes in behaviour, or any outlying observations [BD02].

### 2.2.4 Correlation

Correlation is a method of calculating the relationship between two variables containing a set of values. The strength is given as a score between [-1,1]. A score closer to 1 means that if one of the variables increases, the other one does similarly. If the variables move in opposite directions, the score is closer to -1. 0 means no linear relationship at all. Because the correlation is calculated using pairs of values, the variables are required to be of equal length. This thesis use correlation during time series analysis in Chapter 8.

### 2.2.5 Linear Regression

Linear regression is used to explain the relationship between an explanatory variable $x$, and the dependent variable $y$. There are mainly two types, simple regression, and multi-variable regression, where the former has only one explanatory variable. Linear regression aims to minimise the total difference between all datapoints and the $y$ function. A line is best fitted to model the strength of the relationship, which is displayed by the incline of the line. A simple linear regression can be written as Equation 2.2. $a$ is the slope of the line, and $b$ is the interception on the y-axis. This theory serves to explain multiple fields calculated for recorded gaps in Chapter 7.

$$y = a \cdot x + b \tag{2.2}$$

### 2.2.6 Moving Average

To estimate trend cycles, a useful method called *moving average* can be applied. It is often implemented together with an autoregressive method to form the autoregression moving average (ARMA) model. In equation 2.3, moving average of order $m = 2k+1$, estimates the trend cycle at time $t$ by averaging values of observation in the time series in $k$ periods of time $t$ [HA18]. This method smooths input data and dampens random variations and outliers. It also makes underlying patterns more visible.

$$\hat{T}_t = \frac{1}{m} \sum_{j=-k}^{k} y_{t+j} \tag{2.3}$$

### 2.2.7 Forecasting

Time series forecasting is a technique to predict future observations based on historical data. By analysing these historical data series, the prediction of future observations can be based on detected trends and past behaviour. A fitting forecasting model is chosen and prediction are made on the belief that future trends will stay similar to previously observed trends [HA18]. The prediction fidelity increases with the amount of available data, but may require resampling if frequency is too high. The minimum required observation depends on the chosen modelling approach.

## 2.3 Machine Learning

This section presents content from Chapter 2 in the preliminary project [TS20]. The content is to a large degree reused since the material is not publicly available and the theory is still relevant to this project.

ML is a field within computer science that aims to fit the input data. Broadly speaking, it is used to analyse *unknown* data, and there are many different approaches and applications depending on the issue to be solved. To clarify, unknown data is considered to not yet have been analysed for information other than raw values. An example can be a table of one million rows with personal data. The format may be known beforehand, let us define three columns as a persons name, age and gender. We may have a certain idea of what the table contains, but it is still unknown if for example only teenagers are included in this list, or other complex patterns exist. We introduce another term *labelled* data, also called *ground truth*, by extending on this hypothetical example. Another phrase for labelled data is known as the final classification for a given observation. After analysing the table, the analyst can create a fourth column where all the teenagers have been marked. The table is now considered labelled data, and this characteristic is relevant when approaching ML solutions.

Desired outcomes of ML are often a model that builds itself, i.e. learning from data fed into it. The models can have certain characteristics like data grouping, pattern detection, or predictive properties. The accuracy of these properties relies heavily on the data provided. There are defined four main categories of ML: regression, rule extraction, classification, and clustering. ML algorithms are also categorised into several learning paradigms. They give an overview of how the respective algorithms process and utilise input data. We have unsupervised, supervised, semi-supervised, and reinforcement learning. Supervised learning methods require labelled datasets for training the model. Reinforcement learning algorithms create iteratively learning models. Semi-supervised learning is somewhat similar to supervised, but some of the training data is unknown. Lastly, unsupervised learning can be applied to datasets

where there are no known ground truths, thus highly relevant for this project.

Given a wide range of methods, there is still a generic approach to ML solutions. Most of the work actually lies in the preparation. The initial stages consist of defining the problem and choosing a solution method. Then begins the data collection phase and feature engineering phase. As mentioned, ML models are heavily dependent on input data. Usually one needs to generate a large enough dataset to be able to work with ML. *Features* in the ML-domain is a term for an independent variable for an observation. We use 'feature' in this thesis as a synonym to familiar terms in other domains such as: field, column or attribute.

### 2.3.1 Clustering

Clustering is a typical unsupervised learning method and is used for grouping similar data. Because outcomes of such a model are unlabelled grouped data, an external interpreter is required. Often humans are the ones to do post result analysis because the classic goal of clustering is to reveal patterns. Clustering is discussed in Chapter 7.

### 2.3.2 Tree Based Methods

Tree based methods, often known as decision trees, are methods where the idea is to build a model by growing one or multiple trees from the input data. Multiple trees in a model is called a forest. Input data is typically a set of multidimensional observations with or without labels. Oftentimes the general goal is to predict the class of unlabelled data with a model, but it is also possible to approximate missing features in an observation. For this reason, decision tree algorithms are categorised as classification with supervised learning.

In the case of decision trees, a great example is a handful of people with or without heart problems.[1] They all have individual characteristics such as age, gender, height and weight. A decision tree will ask a series of questions in order to classify an observation as best as possible. To do this, it needs to identify correlations between the features and the labelled class used to train the model. It seeks the most discriminative feature, e.g. if all the people in the sample over 50 years of age had heart problems, it makes sense to ask this question first.

#### Random Cut Forest

An unsupervised learning approach to tree based methods, is Random Cut Forest (RCF). It holds no classification abilities, but can be applied for anomaly detection

[1] A more comprehensive explanation is provided by StatQuest with Josh Starmer: https://youtu.be/7VeUPuFGJHk

(see Section 2.4 for more details). The idea is to continuously grow trees as input is supplied, while underlying mathematical formulas decides how the trees are built. It has been developed methods for calculating how much the trees change for each datapoint. An outlier drastically changes the structure of the tree, hence we can mark anomaly scores for every datapoint. As the name indicates, it utilises an ensemble of trees to form a forest. Each tree is built by a random subset of training data. Separately they vote an anomaly score for each datapoint. By themselves they are weak predictors, but together they vote more accurate results by taking the average of anomaly scores.

One such algorithm is RRCF, an open-source project especially designed for stream data [BMT19] [BMT]. This algorithm is highly scalable and therefore suitable for stream data. By setting a predefined max tree size, it is able to ”remember” a set of earlier observations as context for the next input. When max size is exceeded, the oldest datapoint is omitted or ”forgotten”. This specific algorithm has been used in Chapter 8 for marking anomalies in an attempt to create a discrete univariate dataset (good/bad) of network behaviour.

## 2.4 Anomaly Detection

Although an *anomaly* is defined differently in the many research areas, a generally accepted definition is: *‘An anomaly is an observation which deviates so much from other observations as to arouse suspicions that it was generated by a different mechanism’* [BSL+18]. Anomaly detection is used to disclose data points that digress from the ”normal” behaviour patterns in the dataset and can widely be categorised in: supervised and unsupervised anomaly detection. The input data is the deciding factor of which category to be used [CBK09], specifically the availability of labels in the data, which supervised anomaly detection requires. Unsupervised anomaly detection is more widely applicable since it does not require training data. It will therefore be the natural choice of method in this thesis.

### 2.4.1 Anomaly Detection on Streaming Application

Early detection of anomalies can provide critical information that may be used to prevent system failures. For the information to be usable, it must be accurate, which is a significant trade-off with early detection [ALPA17]. An algorithm that quickly marks anomalies might not be applicable if it results in many false positives.

In a streaming application, machine learning models have to continuously analyse data in real-time, which entails that the entire dataset is not available. The learning must be done in an online fashion, as data arrive [ALPA17]. When the systems observes new data records, the algorithm must examine the current and previous

behaviour to determine if the system is experiencing anomalies. It must be conducted before the next input, in addition to performing retraining and updates. These restrictions result in an algorithm that can not look ahead but only rely on historical behaviour. In a real-world scenario, the defined *normal* behaviour might change over time, a problem known as *concept drift*. Systems are often dynamic and affected by environmental changes. Examples are maintenance work, hardware malfunctions, or software configurations. To cope with this problem, the anomaly detection method must automatically adjust to a new normal.

### 2.4.2 Anomaly Types

Anomalies can be divided into three categories [CBK09]: point, contextual, or collective anomalies. Point anomaly is a data instance that is anomalous compared to the rest of the evaluated data. An example would be a single point lying outside the boundaries of a group of normal data points. The distance from the group can determine if the anomaly is global or local [GU16]. A global anomaly is very different from a distinct gathered group, considering their attributes such as time and measured values. Contextual anomalies are only anomalies in a specific context but might be deemed normal in another. For this technique to be applied, the context has to be able to be defined. Collective anomalies are a collection of related data instances considered anomalies as a group compared to the entire dataset.

### 2.4.3 Anomaly Detection Output

Anomaly detection techniques can produce two types of output when reporting an anomaly: *scores*, or *labels* [CBK09]. Scoring techniques are used to score the evaluated test data, reflecting the degree of confidence that instance is an anomaly. The results will provide a complete overview for an analyst to classify anomalous behaviour in respect to context. This technique allows the analyst to choose the most relevant instances based on their score or filter out low scoring instances with a threshold. Techniques using labels assigns one to each instance in the test data in a binary fashion. A decisive threshold is set to determine which are normal and anomalous. This results in only two types of outcomes, and the threshold must be tuned to prevent an unacceptable amount of false negatives and false positives. Kibana's anomaly detector, tested in Chapter 6, utilises a scoring technique to grade the detected anomalies in the provided datasets.

## 2.5 Root Cause Analysis

Root Cause Analysis (RCA) can be simply stated as a tool designed to help identify not only what and how an event occurred, but also why it happened [RH04]. It is meant to be used to solve a problem by determining the underlying cause, and

find ways to eliminate these causes. Performing RCA over time and identifying reoccurring root causes can result in major opportunities for improvements. In the thesis, RCA is utilised in Chapter 9 to determine the cause of anomalous delay in Uninett's network. The process of RCA is further described in Section B.2 in Appendix B.

# Chapter 3
# Related Work

The following chapter presents related work in the thesis research field. Since the thesis addresses the use of ML in a networking context, the work is grouped and presented in two sections.

## 3.1 Network Behaviour

Because the importance and value of service quality for network providers, a great deal of academic work on improving reliability and security in networks has been done. Understanding the reasons for unwanted behaviour is crucial to be able to improve the quality of service. There are several proposed ways of measuring traffic behaviour in a network. To tackle the challenges of end-to-end delay, [AEJ12] purposes a novel and straightforward approach of performing end-to-end measurements and applying Compressed Sensing (CS) to estimate delay in path hops. By assessing the service quality of an internet path, the origin of the problem could be found. Measurements of delay and loss are utilised to detect abrupt changes using a sparse matrix decomposition called Principal Component Pursuit (PCP) [AJH+14]. As the networks have grown larger, a distributed and more complicated monitoring architecture has been integrated. The nodes in the network generate vast amounts of data in time series that can aggregate parameters as an average delay. The data can be analysed to detect anomalies and determine the root cause [AJS18].

Even in modern networks, a perfect system without outages is not achievable in practise. Issues and threats can not be prevented entirely. In survey [ABM+18], the author presents a number of reasons for outages including accidental misconfigurations, software bugs (in routers), network equipment hardware failure, and many other factors. Clearly understanding how to prevent, detect, and mitigate Internet outages is essential to ensure positive development in the future. The survey states that a fast and accurate detection of an ongoing outage is the essential preliminary step to trigger effective countermeasures. The primary goal is to mitigate as much impact

as possible of the outages as perceived by the final users [ABM+18]. Online outage detection can be divided into passive and active monitoring. Active probing can be approached based on ping and traceroute, or based on tomography, where failures on links are detected by sending coordinated end-to-end probes. Considering the necessary trade-off between several targeted destinations and sampling period, relying exclusively on outage detection systems, it is likely to only report significant and long-lasting network outages [ABM+18].

In pursuit of global Internet availability of "five nines", i.e. being available 0.99999 of the time, an analysis of interdomain availability and causes of failures based on active measurements were conducted [MHH+10]. Active measurements were performed between Norway and China through the Global Research Network, and end-to-end downtime statistics were collected. Examination of the collected data (packet delay, loss, periodic traceroute and number of hops in a used path), enabled identification of paths between end-points and causes of observed network failures. The outages that were observed could be divided into three categories: lengthy outages exceeding 10 seconds, medium outages 1-10 seconds, and short outages less than 1 second. It was concluded that end-to-end availability is mainly affected by extended service downtime (exceeding 10 s) caused by interdomain rerouting. To improve the service availability further towards the goal of "five nines", the interdomain rerouting time needs to be shortened. Additionally, removing cases where no rerouting was executed presents considerable potential for improvements in availability [MHH+10]. The lack of rerouting was caused by "irregular events", link failures, operational activities in the network, or misconfigurations of routing systems. The paper presents the basis of the ongoing Dragonlab initiative Uninett is part of today, of which the data collection used in this thesis stem from.

## 3.2 Machine Learning

Anomaly detection in time series is a topic with extensive studies performed within several areas. [ALPA17] presents an evaluation of real-time anomaly detection algorithms by utilising the Numenta Anomaly Benchmark (NAB), an open-source environment specifically designed for real-world use [LA15]. The Hierarchical Temporal Memory (HTM) algorithm received a high score, indicating the algorithm's ability to learn continuously.

The recent years it has become more popular to use machine learning to analyse the enormous datasets produced by measuring nodes. The anomaly detection technique must be chosen based on the anomaly detection problem. The problems can be divided into supervised and unsupervised. In some situations, a partially supervised anomaly detection can be implemented. Still, it is deemed unsuited for real-time anomaly detection, just like supervised classification that requires

labelled data. In [GKRB13] the authors conclude that a hybrid solution of semi-supervised anomaly detection significantly improved the prediction accuracy with only a small amount of labelled instances, respectively combining the advantages of both paradigms. In a comprehensive evaluation of 19 different unsupervised anomaly detection algorithms on datasets from multiple application domains [GU16], the paper investigates the anomaly detection performance, computational effort, the impact of parameter settings, and the global/local anomaly detection behaviour. The paper concludes with an advised algorithm selection based on real-world tasks, where nearest-neighbour based algorithms are recommended if computation time is not an issue. Clustering-based anomaly detection is better suited for larger datasets when faster computation is needed.

Internet path changes can disrupt performance, cause high latency and congestion, or even loss of connection. By using traceroute and machine learning techniques, the paper [WCD16] study the problem of predicting the Internet path changes and path performance. By building a learning system relying on supervised machine learning algorithms, they predict path performance metrics such as RTT.

This thesis attempts to combine related work in investigating the applicability of algorithms mentioned in Section 3.2, with datasets studied by work mentioned in Section 3.1

# Chapter 4
# Methodology

The method in this project is structured as a set of multiple approaches. [1] Due to the problem in question being the initial investigation of Uninett's ability to utilise ML, a broad scope is to be covered.

In order to evaluate ML applicability, the overall approach consists of a literature study of the field in general, followed by iterations composed by closer inspections of available datasets. The last step is a parallel process of getting familiar with unknown data and evaluating them with respect to ML methods. It is necessary to gain a basic overview of the data before defining a problem and exploring specific algorithms. This chapter will elaborate on the chosen methodology for this thesis, Design Science.

### Design Science

*Research is a logical and systematic search for new and valuable information on a particular topic. It is an investigation of finding solutions to scientific and social problems through objective and systematic analysis* [RPV06]. Research is conducted with a purpose, and sciences like computer science aim to solve a problem by applying new technology to gain knowledge in contrast to natural sciences, which investigates natural phenomena to better describe and understand the observations.

Research methodology is a systematic way to solve a problem. It is a science of studying how research is to be carried out [RPV06]. Research methodologies work as frameworks for researching in order to ensure valid and reliable results in line with the research goals. Design science is the design and investigation of artefacts in context [Wie14]. The focus lies in developing and improving the performance of the designed artefact in interaction with the context to solve the problem. In design science it is referred to as *Design Problem*. It is only in the interaction with the

[1] We perform data exploration directly on Uninett's server, and the environment setup can be found in Appendix C.

context that we can evaluate the quality of the artefact. The main objective of the thesis will work as the design problem. The ML solution will serve as our artefact, and Uninett's datasets will be the context. Design science research can result in different contribution types, from specific to more abstract knowledge [GH13]. This project aims to contribute specific knowledge of level 1, situated implementation of an artefact.

### Engineering Cycle and Design Cycle

Our artefact will be subject to several iterations of designs and investigations to determine a solution that provides the desired outcome. This aligns with the design sciences problem-solving processes, *the engineering cycle* and *the design cycle* [Wie14]. The design cycle can be broken into three tasks: problem investigation, treatment design, and treatment evaluation. Figure 4.1 from [Wie14] illustrates the engineering cycle, consisting of the design cycle and one additional task, implementation evaluation.

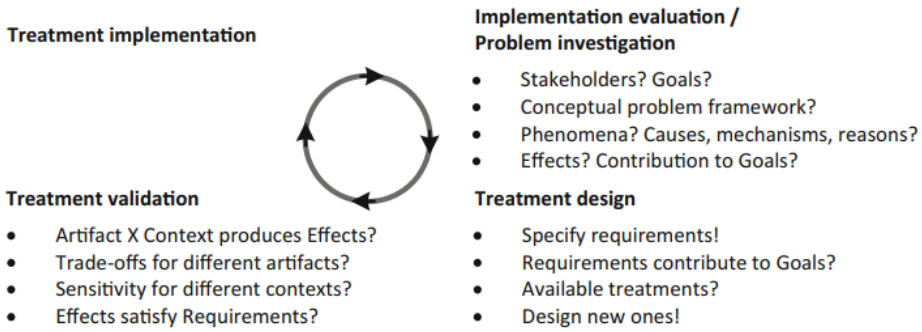



Figure 4.1: The engineering cycle.

### Problem Investigation

Problem investigation is the initial phase of the design and engineering cycle and is essential to prepare for the succeeding design and validation phases. The phase consists of determining what phenomena must be improved and why. The pre-project conducted Fall semester 2020 [TS20] can be considered the beginning of the problem investigation.

To acquire background knowledge, we conduct a comprehensive literature study, continuing on the pre-project report [TS20]. The study mainly focus on gaining a deeper understanding of ICT infrastructure, machine learning fundamentals and anomaly detection techniques. Platforms such as *Google Scholar, ScienceDirect,*

*NTNU Open* and *ORIA* are used to obtain relevant literature. We assess publishing date, author, citations, and publishing area to evaluate the most relevant literature's validity. Furthermore, the thesis supervisors are consulted to recommend state-of-the-art literature worth reviewing. Professors and fellow students at NTNU with experience in machine learning are contacted for guidance and insight in the domain. Openly available tutorials are used for demonstration of how to build and implement anomaly detectors. We choose python as the primary programming language based on the amount of available open-source libraries, such as *Pandas*. They are used to preprocess and analyse data.

The project involves testing several techniques to extract valuable output from Uninett's data. Therefore the redesign of the artefact in the design cycle iterations is a new approach or an expansion on a previous one. As iterations are performed, the problem investigation step is revisited to ensure the required understanding of the problem at hand.

A significant task in the investigation is gaining a solid understanding of the datasets and how they are produced. Uninett has already implemented a solution for gathering measurements from the probes in their network. They have also developed quantitative methods for calculating performance characteristics. An observational case study [Wie14] is performed to gain insight into their established system. To evaluate this, we define a second design cycle. The project use the artefact as the method to collect and store the measured network traffic. The context is still Uninett's network. The last step in the engineering cycle is implementation evaluation, where the goal is to investigate how implemented artefacts interact with their real-world context [Wie14]. Seeing that an artefact will not be set into production during the project, it will be evaluated in an artificial environment replicating real-world context.

Through an iterative process, programming scripts are crafted to improve the way data from the network probes are labelled and stored. This treatment results in a shorter loading time of datasets, which we validate by timed tests. It is implemented in the project's context and used as if it was in production at Uninett. The implementation is evaluated by consulting experts and stakeholders at Uninett.

**Treatment Design**

The final design of the artefact, the treatment, is defined as treatment design. In this project, the treatment is the recommended ML solution to improve Uninett's data analysis. For every iteration of this step, the problem investigation is revisited several times to confer subject material. Thus, the model is designed based on the updated understanding of the problem and results from previous iterations.

### Treatment Validation

An important part of the project is validating the design treatment to determine if the model actually contributes to solving the problem. In this project, the treatment is validated by consulting experts at Uninett over several meetings. This way, they remain well-informed of the development and can contribute meaningful feedback based on the defined validation criteria. Additionally, we validate by exposing the artefact to different datasets in a closed environment to predict how it will interact with the real-world context.

### Method Assessment

Design science as a framework is well-suited for this project, given that splitting the problem into smaller parts seems natural. Iterating through the cycles to improve upon previous prototypes from errors and mistakes allows the researcher to better evaluate the treatment before redesigning. The method facilitate a deeper understanding of the problem and serves as a structure to guide the development of an artefact prototype. The method can be used for further iteration, but the project is limited on time.

# Chapter 5
# CRUDE

This chapter, will introduce and discuss the dataset we choose as the main focus of attention for this project. Uninett's server contains approximately 70 TB of data. The vast majority of these files stem from simulated traffic in the probe network. For this reason, an investigation of ML applicability on this resource is of interest. What information does it contain, and can it be used for more than it is today?

## 5.1 Implementation

All streams of UDP packets received at CRUDE nodes are written to a single .gzip file per node every day.[1] A stream is a path from SRC to DST containing a varying number of routers. Depending on the number of input streams, these files can accumulate to several GigaBytes (GBs).

### Preprocessing

We develop a simple way of locating desired files from their server, see Listing D.1. This script was used extensively when we analysed the available resources during the project. To read the compressed files, we use the *gzip* library provided by python developers. An excerpt from a CRUDE log can be seen in Table 5.2. Each line in the file represents a received packet and contains multiple fields of information. The fields are explained in Table 5.1.

Familiarising with this resource require some preprocessing, seeing how the raw format is slow and unpleasant to work with, see Figure 5.1. In certain cases, we desire to look at specific streams over multiple days. With the current format, the analyst is forced to process millions of packets unnecessary.

We limit the CRUDE data to be extracted within March of 2021. There is not enough time to parse their entire server into new datasets, and this interval is

[1] gzip is a (de)compression algorithm used to reduce memory usage.

| | |
|---|---|
| *ID* | An auto-generated id for a stream between two IP addresses. |
| *SEQ* | The sequence number for each packet in a given stream. |
| *SRC* | The source node for a stream, annotated as IP:PORT. |
| *DST* | The destination node for a stream, annotated as IP:PORT. |
| *Tx* | Unix timestamp for transmitted packet from source node. Unix time is measured in elapsed seconds since epoch time (1970.01.01 00:00:00.000). Timestamps Tx and Rx are measured by two different clocks. |
| *Rx* | Unix timestamp for received packet at the destination node. |

Table 5.1: Descriptions of fields in CRUDE records.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| ID=6 | SEQ=0 | SRC=128.39.65.26:3106 | DST=130.216.51.132:10001 | Tx=1615330812.711761 | Rx=1615330812.872090 | SIZE=64 | HOPLIMIT=-1 |
| ID=3 | SEQ=0 | SRC=13.82.53.167:3103 | DST=130.216.51.132:10001 | Tx=1615330812.589545 | Rx=1615330812.705517 | SIZE=64 | HOPLIMIT=-1 |
| ID=6 | SEQ=1 | SRC=128.39.65.26:3106 | DST=130.216.51.132:10001 | Tx=1615330812.721264 | Rx=1615330812.880612 | SIZE=64 | HOPLIMIT=-1 |
| ID=3 | SEQ=1 | SRC=13.82.53.167:3103 | DST=130.216.51.132:10001 | Tx=1615330812.599427 | Rx=1615330812.711252 | SIZE=64 | HOPLIMIT=-1 |
| ID=6 | SEQ=2 | SRC=128.39.65.26:3106 | DST=130.216.51.132:10001 | Tx=1615330812.731267 | Rx=1615330812.890540 | SIZE=64 | HOPLIMIT=-1 |
| ID=3 | SEQ=3 | SRC=13.82.53.167:3103 | DST=130.216.51.132:10001 | Tx=1615330812.619449 | Rx=1615330812.731192 | SIZE=64 | HOPLIMIT=-1 |
| ID=6 | SEQ=3 | SRC=128.39.65.26:3106 | DST=130.216.51.132:10001 | Tx=1615330812.741244 | Rx=1615330812.902948 | SIZE=64 | HOPLIMIT=-1 |
| ID=3 | SEQ=4 | SRC=13.82.53.167:3103 | DST=130.216.51.132:10001 | Tx=1615330812.629408 | Rx=1615330812.741181 | SIZE=64 | HOPLIMIT=-1 |
| ID=6 | SEQ=4 | SRC=128.39.65.26:3106 | DST=130.216.51.132:10001 | Tx=1615330812.751262 | Rx=1615330812.915790 | SIZE=64 | HOPLIMIT=-1 |
| ID=3 | SEQ=6 | SRC=13.82.53.167:3103 | DST=130.216.51.132:10001 | Tx=1615330812.649443 | Rx=1615330812.761059 | SIZE=64 | HOPLIMIT=-1 |
| ID=3 | SEQ=7 | SRC=13.82.53.167:3103 | DST=130.216.51.132:10001 | Tx=1615330812.659413 | Rx=1615330812.771785 | SIZE=64 | HOPLIMIT=-1 |
| ID=3 | SEQ=2 | SRC=13.82.53.167:3103 | DST=130.216.51.132:10001 | Tx=1615330812.609424 | Rx=1615330812.781229 | SIZE=64 | HOPLIMIT=-1 |
| ID=3 | SEQ=8 | SRC=13.82.53.167:3103 | DST=130.216.51.132:10001 | Tx=1615330812.669453 | Rx=1615330812.791250 | SIZE=64 | HOPLIMIT=-1 |

Table 5.2: Raw CRUDE example excerpt.

relatively recent at the time of writing. Columns ID, SEQ, SRC, DST, Tx, and Rx are extracted and saved as CSV files, separating the braided streams into individual files. We parse the data because CSV format is supported by the analysis library pandas we intend to use for data exploration. For convenience, we introduce another field, delay, while parsing ($delay = Rx - Tx$). Delay represents the nodal delay of any given packet. The script to parse raw CRUDE files into CSV files is displayed in Listing D.2.

We mainly perform manual inspections on this dataset. Visualisations are made to show the development of packet delays during the day. Historical experience from Uninett has established that streams to NGU are more troublesome than other nodes. For this reason, we choose NGU for further exploration. Via Elasticsearch, the day in March with the longest registered gap, is located as one of the samples to look into. This exact day is shown in the results in Section 5.2. Why we choose not to investigate the CRUDE dataset further is discussed in Section 5.3.

In [21]:

```
start_time = time.time()

fields = "id seq src dst tx rx size hoplimit"
fields = fields.split()

def get_val(item):
    """item is of format: k=v"""
    try:
        return item.split('=')[1]
    except:
        return item

def get_ip(item):
    """item is of format: k=ip:port"""
    try:
        return get_val(item).split(':')[0]
    except:
        return item

funcs = {
    0: get_val,
    1: get_val,
    2: get_ip,
    3: get_ip,
    4: get_val,
    5: get_val,
    6: get_val,
    7: get_val,
}

df = pd.DataFrame()
for f in files:
    tdf = pd.read_table(f, header=None, delimiter=' ', skiprows=4, converters=funcs, names=fields)
    df = df.append( tdf, ignore_index=True)


elapsed = datetime.timedelta(seconds=time.time() - start_time)
print(f"\n===== Done! ({datetime.datetime.now()}), elapsed {elapsed} =====")
```

```
===== Done! (2021-04-14 15:36:34.952215), elapsed 0:08:39.930388 =====
```

Figure 5.1: Parse raw CRUDE file to dataframe.
Elapsed time to parse a compressed CRUDE file for Auckland is 8 minutes 40 seconds. The file is 3.6 GB in size.

## 5.2 Results

Transforming raw crude logs to CSV format enables us to read files data approximately 75% faster. From 8 minutes 40 seconds previously shown in Figure 5.1, down to 2 minutes 12 seconds, see Figure 5.2. The same streams were used in both examples. Tables 5.3 and 5.4 show the new format of the extracted streams from Table 5.2.

```
start_time = time.time()

dfs = {}

for i, f in enumerate(files):
    try:
        dfs[i] = pd.read_csv(f)
    except:
        continue

elapsed = datetime.timedelta(seconds=time.time() - start_time)
print(f"\n===== Done! ({datetime.datetime.now()}), elapsed {elapsed} =====")
```

```
===== Done! (2021-05-28 20:57:14.274609), elapsed 0:02:12.882442 =====
```

Figure 5.2: Read CSV file to dataframe.
CSV format reduced disk reading to 2 minutes 12 seconds.

| id | seq | src | dst | tx | rx | delay |
|---|---|---|---|---|---|---|
| 3 | 0 | 13.82.53.167 | 130.216.51.132 | 1615330812.589545 | 1615330812.705517 | 0.115972 |
| 3 | 1 | 13.82.53.167 | 130.216.51.132 | 1615330812.599427 | 1615330812.711252 | 0.111825 |
| 3 | 3 | 13.82.53.167 | 130.216.51.132 | 1615330812.619449 | 1615330812.731192 | 0.111743 |
| 3 | 4 | 13.82.53.167 | 130.216.51.132 | 1615330812.629408 | 1615330812.741181 | 0.111773 |
| 3 | 6 | 13.82.53.167 | 130.216.51.132 | 1615330812.649443 | 1615330812.761059 | 0.111616 |
| 3 | 7 | 13.82.53.167 | 130.216.51.132 | 1615330812.659413 | 1615330812.771785 | 0.112372 |
| 3 | 2 | 13.82.53.167 | 130.216.51.132 | 1615330812.609424 | 1615330812.781229 | 0.171805 |
| 3 | 8 | 13.82.53.167 | 130.216.51.132 | 1615330812.669453 | 1615330812.791250 | 0.121797 |

Table 5.3: Stream 3.
Stream 3 extracted from Table 5.2 into CSV format. Packet nr. 2 was delayed and packet nr. 5 was lost.

Figure 5.3 shows the packet delays for the stream between Oslo and NGU on the 29th of March 2021. Approximately 8.6 million packets were sent throughout the day, and the vast majority used under 0.1 ms in transit. We notice very few outliers with spikes in delay. There is also no clear buildup of delay before these incidents.

The largest gap (410 ms) in March occurred at timestamp 00:26:43 in this stream. We observe a sudden change close to the start of the stream, matching with the

| id | seq | src | dst | tx | rx | delay |
|---|---|---|---|---|---|---|
| 6 | 0 | 128.39.65.26 | 130.216.51.132 | 1615330812.711761 | 1615330812.872090 | 0.160329 |
| 6 | 1 | 128.39.65.26 | 130.216.51.132 | 1615330812.721264 | 1615330812.880612 | 0.159348 |
| 6 | 2 | 128.39.65.26 | 130.216.51.132 | 1615330812.731267 | 1615330812.890540 | 0.159273 |
| 6 | 3 | 128.39.65.26 | 130.216.51.132 | 1615330812.741244 | 1615330812.902948 | 0.161704 |
| 6 | 4 | 128.39.65.26 | 130.216.51.132 | 1615330812.751262 | 1615330812.915790 | 0.164528 |

Table 5.4: Stream 6.
Stream 6 extracted from Table 5.2 into CSV format.

registered gap. A slightly higher delay has been measured most of the day, followed by a lower delay again after the 6 million mark.

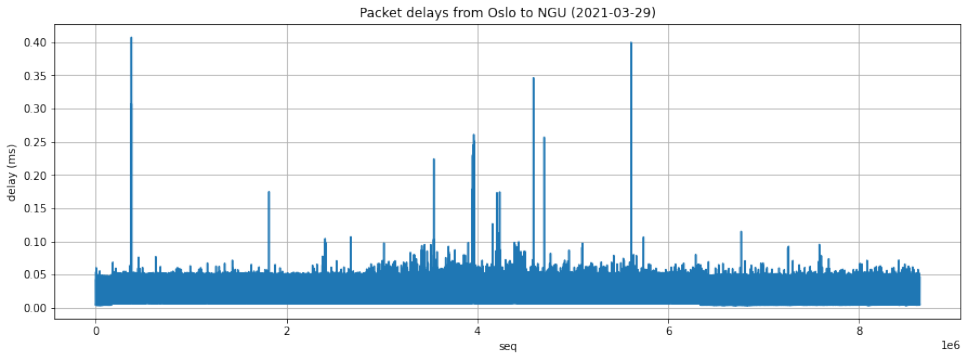



Figure 5.3: Packet delays from Oslo to NGU.

## 5.3 Discussion

By introducing CRUDE software for monitoring, the system generates a simplified dataset compared to actual traffic data. A fixed packet frequency means a constant expectation at the receiving end. The network has essentially been abstracted to a level where disruptions are trivial to discover by threshold methods. Such analysis is already performed by Uninett and is discussed in Chapter 7.

We suspect that the sudden change shown in Figure 5.3, followed by a period of increased lower-bound delay, indicate a reroute to a sub-optimal path. However, we do not know the reasons for this, nor have we confirmed this suspicion. The results in Section 5.2 points out that the deviating interval is accompanied by a gap at the very beginning. The Motivation, Chapter 1.1, mentions that gaps may lead to more severe issues. This observation supports that gaps may be indicators of such incidents.

Although there is an immense set of available data, each datapoint has low informational value. The only field of interest is the transmit delay of packets. On its own, it does not mean much. One must look at the surrounding packets for better insights into the streams' status at any point in time. For this reason, CRUDE data is considered a time-series problem. "Time series problems" is a complex branch of machine learning. Datapoints are dependant on each other, thus, more complex algorithms are required. We are interested in understanding the underlying causes of disruptive incidents in the network. Considering that CRUDE data has only a single dimension other than time, any context data of these occurrences is non-existent. As it is, we see few benefits from applying any ML algorithm on it and we choose not to investigate this dataset further. In other terms, we find this dataset to be a continuous stationary univariate dataset with irregular fluctuations, and we struggle to identify trends or extract patterns from lack of other variables. These terms are documented in Section 2.2.

The stream shown in Figure 5.3, does to our knowledge, represents how most of the streams behave. We feel the need to remind that vast amount of data is a limiting factor, there is not enough time in this project to inspect them all. The recurring traits of streams are stationary series with low delay, few delay spikes and few lost packets. This supports the notion that Uninett has developed a robust network over the years.

# Chapter 6
# Kibana Anomaly Detection

This chapter introduces the implementation and testing of Kibana's anomaly detection on Uninett's collected datasets. The main goal is to explore to what extent established tools can extract patterns in Uninett's available data.

## 6.1 Implementation

Kibana is an open user interface, and Uninett utilises it to visualise their stored monitoring data. Combined with the search engine Elasticsearch, data is structured with field names in indices, displayed in Figure A.1 in the appendix. This facilitates the data to be used with Kibana's range of functionalities such as charts, metrics, and anomaly detection. The anomaly detection is automated and utilises the unsupervised ML algorithm (RCF), introduced in Section 2.3. It differentiates an anomaly from normal variations by grading each incoming data point based on a modelled sketch of the incoming data stream [Ela]. Automated detection in changing data ensures the configurations required in terms of specifying algorithms and models. This makes the Kibana anomaly detector a "black box" to the user, who only provides input data and determines the desired fields of focus in the data. The index (dataset) uninett_jitter, which contains data relevant to monitor jitter in the network, is applied to test the detector. The test focuses on the numerical field *h_ddelay*, representing an average of the 50 packets in the header minus the fastest of the 1000 packets. As shown in Figure 6.1, the detector is configured to uncover anomalies based on the minimum value in our data. The results are presented in Section 6.2.

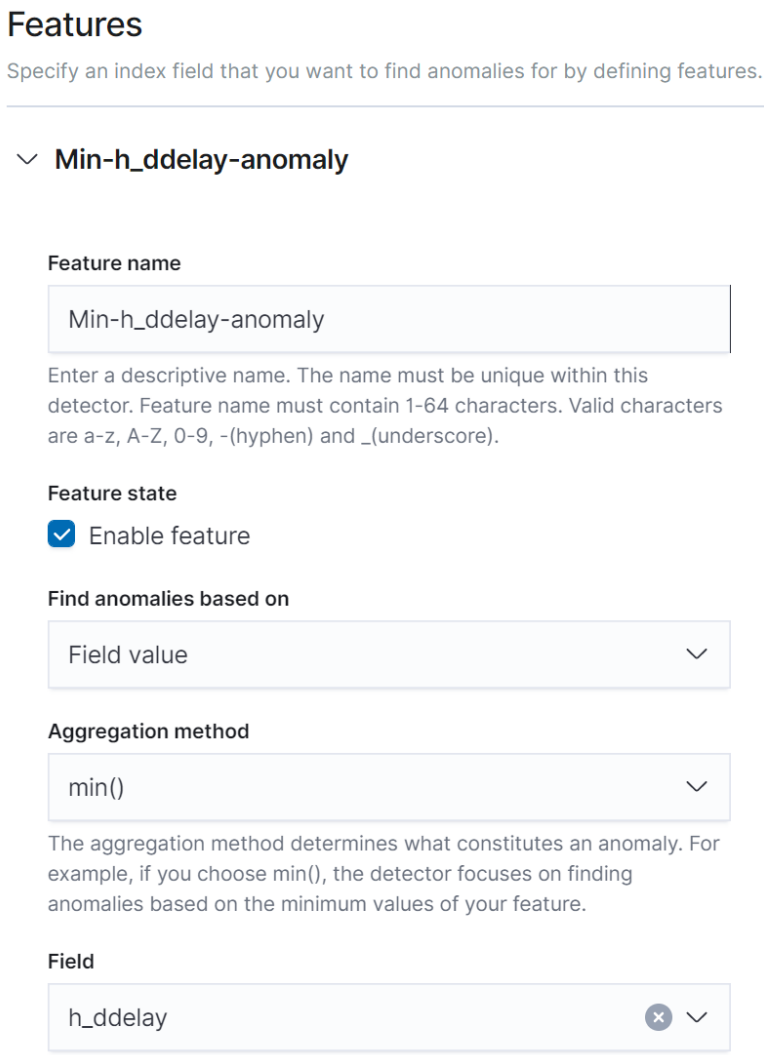



Figure 6.1: Configuration of features in Kibana's anomaly detector.

## 6.2 Results

The anomaly detector in Kibana score the datapoints with an anomaly grade and a confidence grade. The confidence grade between 0 and 1 indicates the level of confidence in the anomaly result, while the anomaly grade between 0 and 1 indicates the extent to which a data point is anomalous. Higher grades indicate more unusual data. As more data is fed into the anomaly detector, the confidence in the anomaly classification has a logarithmic growth, as shown with the green plots in Figure 6.2. The y-axis in the graph represents values in the interval [0,1] for both confidence (green) and anomaly grade (red). Along the x-axis are the dates of the entered data points.

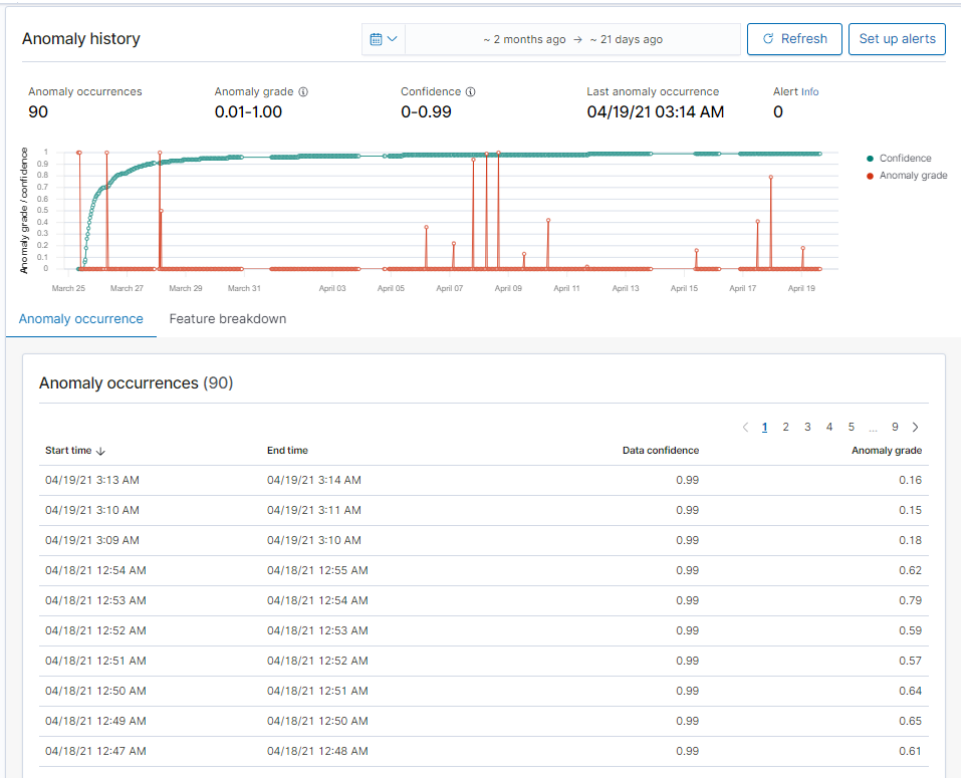



Figure 6.2: Anomaly detection in Kibana.

A slice of the results from the anomaly detector is presented in the figure. The graph shows distinct plotted deviations in the data, with its affiliated score displayed in the attached table. The table contains a start and end time for every occurred anomaly, in addition to the anomaly and confidence grade. The anomaly grading of the occurrences ranges from minor anomalies graded at 0.16 to more concerning

grades of 1 with high confidence. Figure 6.3 shows a close-up view with highlighted anomalies. We see that, within a 10-minute time frame, seven significant occurrences have been highlighted.

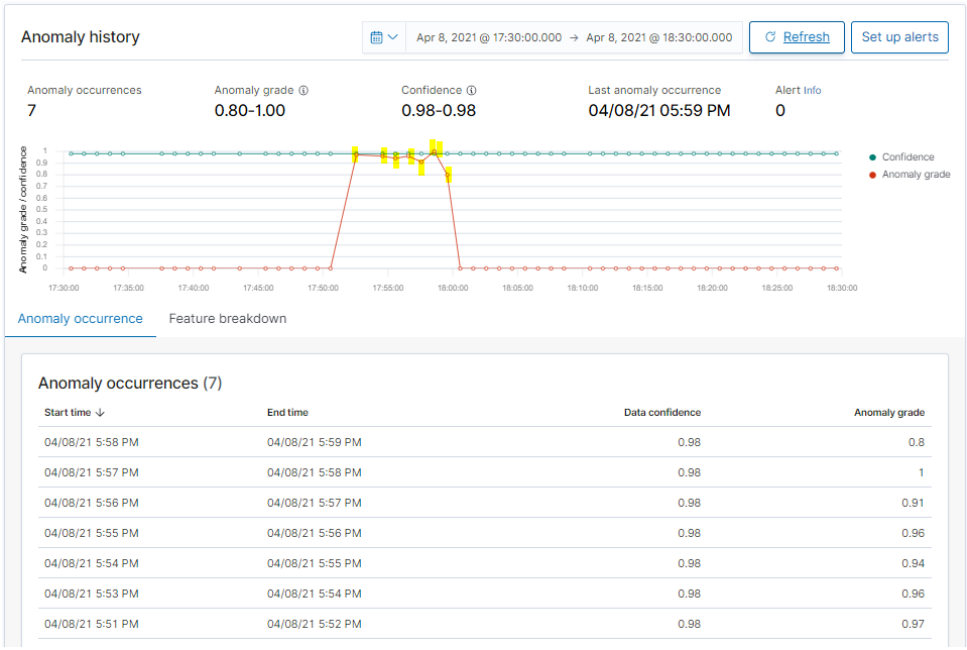



Figure 6.3: A closer scope of anomaly detection in Kibana.

## 6.3 Discussion

The anomaly detection in Kibana is a practical tool for enterprises and private user utilising Kibana indexing and visualising data. It has an intuitive interface and configuration. The only requirements to get started are initiating a detector with a defined interval and deciding which data field to aggregate and how. Open Distro for Elasticsearch is an open-source distribution of Elasticsearch and Kibana. It is a free alternative to the premium version and other costly data platforms such as Splunk. Open Distro is continuously evolving and new plugins are periodically added and improved. A first version of the anomaly detector plugin was included only one year ago, and consequently, users and technician might not be familiar enough to fully utilise the machine learning.

While we used this tool, we experienced issues with the detector caused by missing data in the stream or data not being ingested correctly, which halted the detector. Each detector is only limited to one index, which can be worked around using a greater index pattern that merges several indices. The detector is initially limited to only five features, but can be expanded by altering the configuration. From the results of the implementation, we can see that distinct anomalies are detected. They receive a high grade with an equal amount of confidence. They are point anomalies, where a couple of them are close enough together to make out collective anomalies. Considering the grouping and grading of the anomalies, it can indicate that larger issues have occurred. By investigating the indices in Kibana, we can find the actual measured values of h_ddelay, highlighted in Figure A.2a in the appendix. These values clearly stand out and support the classification of the anomalies. Four of these anomalies occur in pairs of two, separated by milliseconds. Each entry in the index can be expanded to retrieve further information that can be used in a root cause analysis [1] to determine the underlying cause of the incident. Figure A.2b displays one of the routes with the longest delay from ytelse-osl.uninett.no to ngu-mp.ngu.no. The other route was from ytelse-trd.uninett.no to ngu-mp.ngu.no. The source nodes are located in Oslo and Trondheim, and the destination NGU is located outside Lade in Trondheim. Since NGU is common between the two paths, root cause is suspected to be related to this destination.

In the user Kibana interface, it would be useful to inspect the data instances directly, not only the anomaly grade. By investigating the entered documents (data elements) in the Uninett's index in Kibana, shown in Figures A.2a and A.2b, we can see that Uninett already detects and labels these anomalies binary by a set threshold. The expanded document displays a report_type threshold, highlighted in yellow, which is generated when the measured value surpasses a defined threshold. The graded anomalies found by Kibana might provide a more nuanced impression of the

[1] Introduced in section 2.5

situation, but require manual inspection to determine its validity and to classify the anomaly. The classification based on grading can be done automatic by a threshold, which would then make the grading excessive.

# Chapter 7
# Gaps

Gap-events, defined in Chapter 2, is another dataset available on Uninett's server. This dataset is derived from raw CRUDE data introduced in Chapter 5 and is composed of individual gap occurrences. This data is only available for Uninett's intranet. We were recommended to explore gaps by our Uninett-employed co-advisers during the preliminary project. The purpose of investigating this resource is to uncover if Uninett's analysis has resulted in data suitable for ML algorithms.

## 7.1 Implementation

Listing 7.1 seen below is an example of a registered gap record. Except for *overlap*, *h_n* and *t_n*, all the numerical values are measured in milliseconds (ms). When gaps are found, a fixed number of packets before and after is used for context analysis. We call these head *h* and tail *t*. Most fields are calculated for both head and tail. A star (*) has been inserted to represent both head and tail for the following field descriptions in Table 7.1.

Because this dataset comprises individual gap events extracted from CRUDE data, we no longer consider it a time series. We evaluate this dataset with the assumption that gaps are independent of time. A traditional ML approach seems suitable in this scenario. What is meant by traditional ML is using algorithms designed for parsing input data, learn from it, and apply the gained knowledge for some purpose. Acknowledged methods are clustering, classification, and regression within unsupervised and supervised learning.

There are no labelled root causes for gap records, which scopes the study of this dataset to unsupervised algorithms. To better understand the numerical features, we make visualisations of coordinate systems. We choose to compare the numerical measurements from head and tail with *tloss* and *timestamp*, attempting to uncover correlation or patterns of gap-length with context metrics. Why we not pursue this resource any further is explained in Section 7.3.

| | |
|---|---|
| *from* | The source hostname for the path of packets. |
| *to* | The destination hostname for the path of packets. |
| **_n* | The exact number of packets used for head and tail, usually set to 50. |
| **_delay* | The average transmit time. |
| **_ddelay* | Equals **_delay* minus the fastest packet in the current window of 1000 packets. The sliding window method is used when parsing raw data to reduce memory usage. |
| **_jit* | is the jitter value. See Chapter 2 for further explanation. |
| **_slope_*** | The value $a$ in a linear regression $y = ax + b$ for a given number of packets (**) closest to the gap. Linear regression is explained in Chapter 2. |
| *overlap* | Sometimes gaps occur close enough together such that head and tail overlap. This field displays those occurrences. Usually the value is 1. |
| *tloss* | The time lost in a gap. |

Table 7.1: Descriptions of fields in gap records.
Note, a single star (*) represents both head $h$ or tail $t$.

```
{
    "datetime": "2018-05-16 18:06:25",
    "from": "volda-mp.hivolda.no",
    "h_ddelay": 0.042,
    "h_delay": 2.278,
    "h_jit": 0,
    "h_min_d": 2.249,
    "h_n": 50,
    "h_slope_10": 0,
    "h_slope_20": 0,
    "h_slope_30": 0,
    "h_slope_40": 0,
    "h_slope_50": 0,
    "overlap": 1,
    "t_ddelay": 140.241,
    "t_delay": 151.886,
    "t_jit": -7.655,
    "t_min_d": 2.221,
    "t_n": 49,
    "t_slope_10": -1,
    "t_slope_20": -1,
    "t_slope_30": -1,
    "t_slope_40": -0.974,
    "t_slope_50": -0.851,
    "timestamp": 1526486785711,
    "timestamp_zone": "GMT",
    "tloss": 631070,
    "to": "ntnu-mp.ntnu.no",
    "event_type": "gap",
}
```


Listing 7.1: Example gap-record

## 7.2 Results

Below are visualisations of the numeric features of gap records. We select a few samples to discuss in Section 5.3. Most comparisons are available in Appendix E because all possible combinations of comparisons do not provide more varying results. These figures are a result of pattern searching in gap events. *tloss* and *timestamp* is used as the x-axis and the other features on the y-axis. The gaps we use occurred on streams to ytelse-trd.uninett.no from January 2019 to May 2021. A total of 5795 gaps were gathered during this period. We omit gaps with *tloss* over 4000 ms for illustrative purposes, reducing the set to 5611 gaps. They are rare and extreme outliers skewing the axis if included.

Datapoints compared with *tloss* mostly hug the x-axis, indicating no trend or correlation when gaps get longer. Still, a minority of tiny gaps less than 500 ms tend to have a broader range of values. Figures h_delay, h_jit, and h_slope_50 (7.1, 7.2, 7.3) have this behaviour in common.

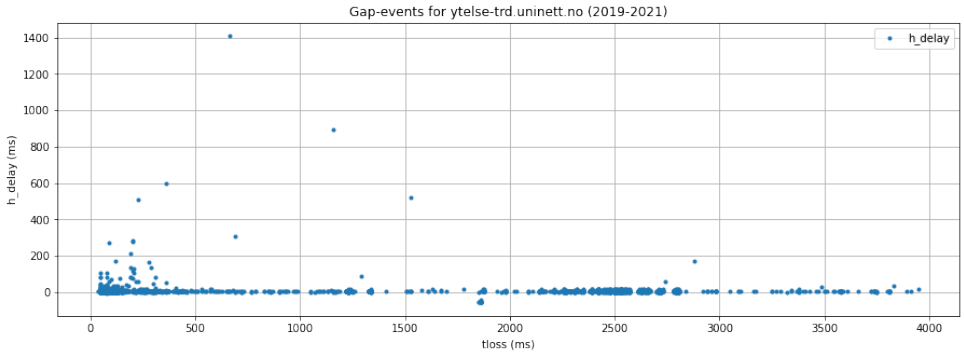



Figure 7.1: Gaps tloss/h_delay.

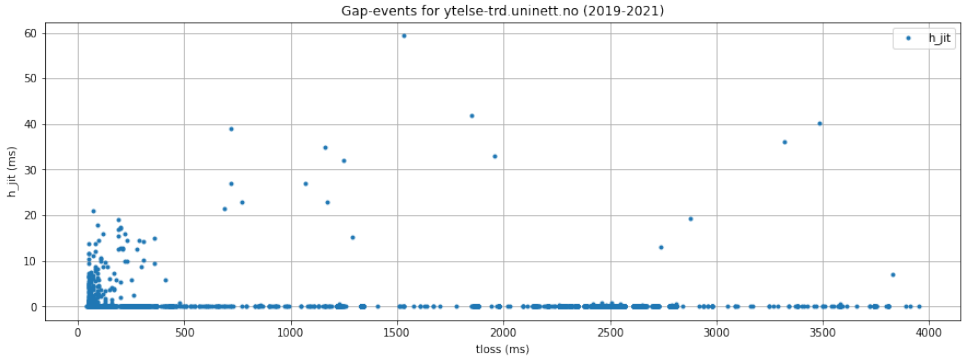



Figure 7.2: Gaps tloss/h_jit.

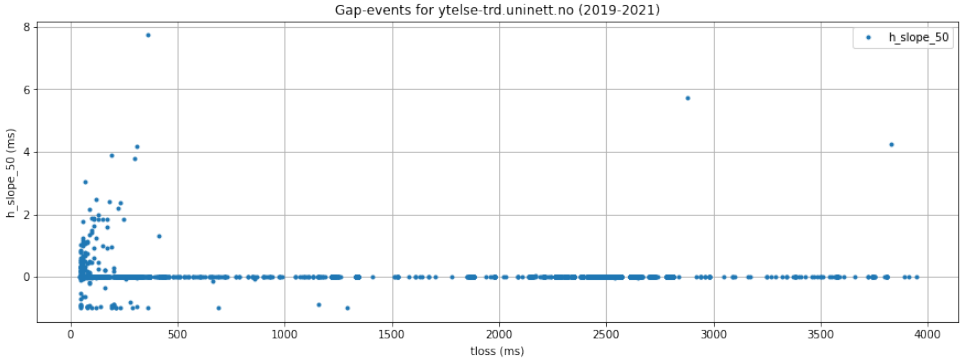



Figure 7.3: Gaps tloss/h_slope_50.

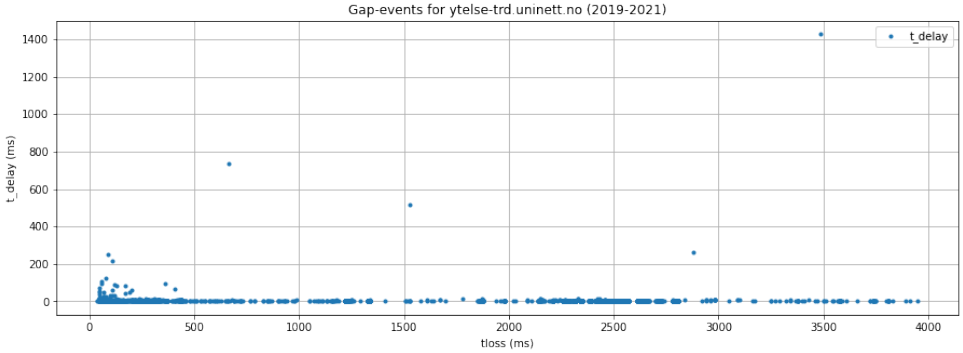



Figure 7.4: Gaps tloss/t_delay.

The results for t_jit in Figure 7.5 show that most jitter-measurements in tails are close to either an upper band of 50 ms or the lower band of 0 ms, while the other datapoints appear in between the two. Looking at jitter in tails over time (Figure 7.6) reveals two spikes before the 1.600 and 1.606 1e9 marks. We also see a sudden change from the usual value of 50 ms before the 1.625 1e9 mark (beginning of March 2021).

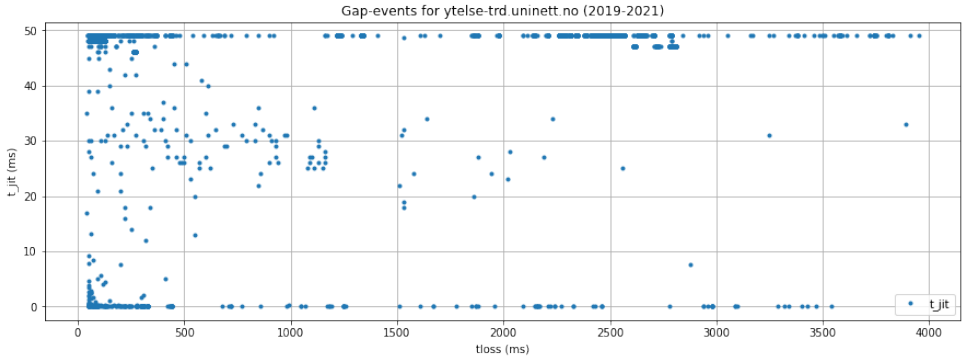



Figure 7.5: Gaps tloss/t_jit.

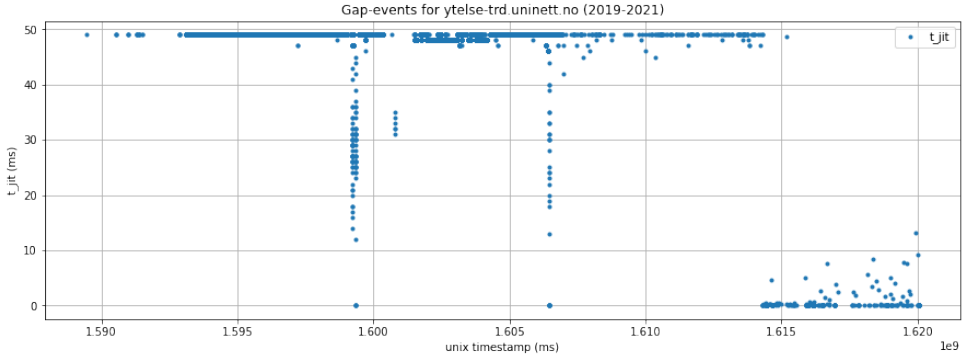



Figure 7.6: Gaps timestamp/t_jit.

Regression measurements from t_slope_10 (Figure 7.7) appear to form horizontal lines between the interval of 0 to 15 ms. The exception is a cluster of a few outliers with approximately -55 ms with tloss of 1800 ms. The measurements over time in Figure 7.8 appear similar to Figure 7.7.

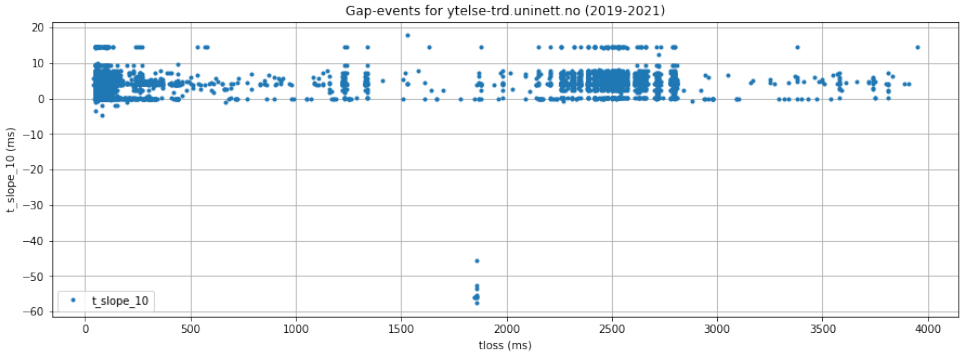



Figure 7.7: Gaps tloss/t_slope_10.

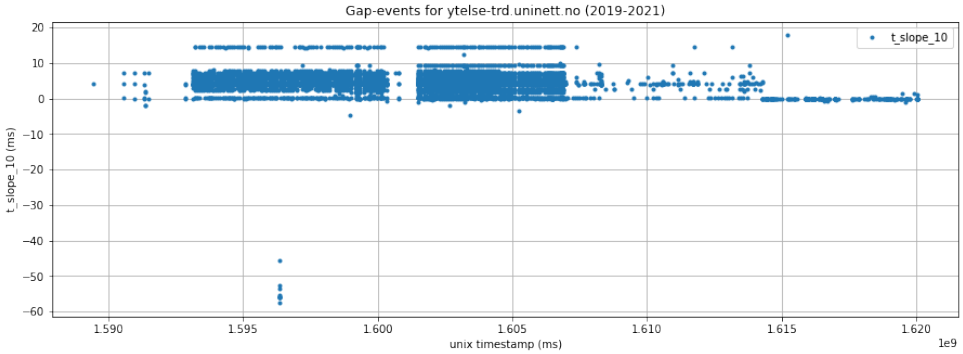



Figure 7.8: Gaps timestamp/t_slope_10.

Increasing the slope measurements to 50 packets shows a flat, uninteresting comparison. This result means that the stream is relatively stable 50 packets after a gap.

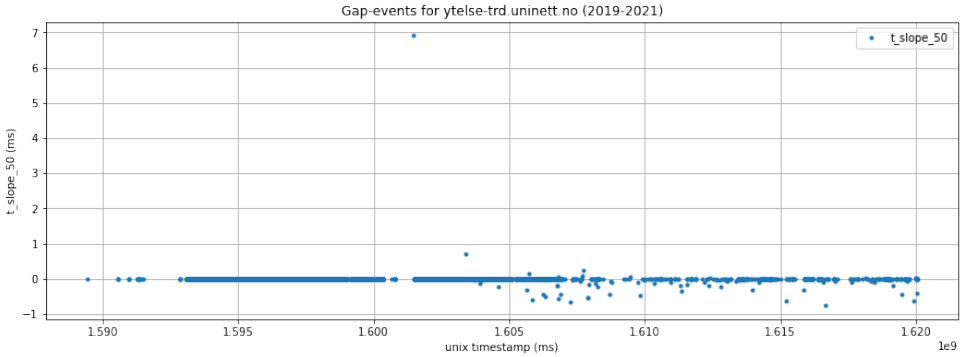



Figure 7.9: Gaps tloss/t_slope_50.

## 7.3 Discussion

The results point out several observations that will be explained in this section.

During the initial stages of the project, we envisioned ourselves utilising clustering as a method to extract information on this dataset. This plan comes to a halt when we with visualisation find little to nothing that spark more interest or can reveal more about these mysterious gaps. Realising that most features are derived from a single metric, namely *delay* from CRUDE data, the correlations add no new information. On the surface, the dataset present itself like a multivariate dataset, but we know it has poor informational value because the fields are related. We believe that to apply ML algorithms, designed for multivariate data, on this dataset will likely not output any great result.

Multiple lines that look like patterns in *t_slope_10* (Figures 7.7, 7.8) can be explained by the different geographical distances the multiple streams travel. We have seen more examples like these, and they are not particularly interesting. However, there are tendencies of recorded gaps with equal length and timestamp. This observation is investigated further in Chapter 8.

The sudden shift in *t_jit* can be explained by a rework of Uninett's analysis algorithms in March 2021. Unix timestamp 1.615 1e9 matches this description. Gaps after this timestamp are the reason for the divided graph shown in Figure 7.5 (tloss/t_jit). It goes to show that this is a real-life system. Things change over time, and data is imperfect.

It is still unknown what happened during the two spikes seen in Figure 7.6. They do not match any abnormal behaviour across the other feature comparisons. This observation identify two distinct events that can be investigated in the future. To achieve that, one must look for information in other datasets. We, therefore, believe that applying ML on this resource alone will not produce valuable information. One can also argue that discovering these incidents does not require ML either, seeing how a simple diagram was sufficient.

To complement this dataset, we have some suggestions. The results highlighted that small gaps (<500 ms), more often than longer gaps, have a wider range of values. A step closer to understand gaps, could be to classify/label/link gaps to typical root causes (if possible) such as ”rerouting” or ”overloaded paths”. Without evidence, smaller gaps could arguably fit the latter class. Measurements, e.g. jitter, could indicate a stressed network. As gaps get longer, they appear to have fewer context data. Not uncommonly, networks behave seemingly fine until they suddenly do not. For this reason, we believe that larger gaps experience more severe and sudden disruptions where there is no indication of buildup. We discuss this further

in section 10.3.

# Chapter 8
# Cross-Stream Comparisons

In this chapter, we introduce another approach. Instead of evaluating already existing datasets, we derive a new one from raw CRUDE logs introduced in Chapter 5. In the previous work, we mainly familiarise ourselves with the data and ML in parallel. With this approach, we aim to investigate if traditional statistical analysis can produce interesting results. The idea is to compare multiple streams contrary to earlier analysis performed on a per-stream basis. The ultimate goal is to better understand the system as a whole by widening the scope.

## 8.1 Implementation

This section is divided into two sub-implementations. The former subsection describes the primary analysis, and the latter describes an ML implementation.

### Preparatory Steps and Statistical Analysis

We choose to focus on streams from Dragonlab. These streams travel over longer distances through a global infrastructure and are known to be less reliable than in Uninett's network. Thus they are more likely to contain interesting behaviour. Using the renowned library pandas[1] in python, *mean_delay* and *packet_count* values are aggregated from each stream. We divide packets into equally sized bins before calculations, and store the results as "resampled" CSV files. Because of the nature of CRUDE data, multiple preparatory steps are needed in order to achieve this.

Firstly, we sort each CRUDE file by the packets' sequence numbers. Lost packets are reinserted as dummy rows with the correct sequence number, while the rest of the features are populated with Not a number (NaN) values. Pandas provides a rolling method that allows for computations on a sliding window over the data. A specific method we use is called moving average, explained in Chapter 2. CRUDE data contains series of nodal delays; for simplicity, we call it "delay" in this chapter.

[1] https://pandas.pydata.org/

Mean-delay, as well as packet-count are calculated for bin sizes (*BINSIZE*): 100, 500 and 1000. Note, pandas does not include NaN values during calculations. The rolling method does not support a step-size argument, so we slice the dataframes[2] by selecting every BINSIZE value starting from BINSIZE. An example can be seen in Table 8.1. The code used for resampling can be found in Appendix D.3.

| seq | mean_delay | packet_count |
|---|---|---|
| 100 | 0.1207615 | 100 |
| 200 | 0.1618642 | 100 |
| 300 | 0.1378201 | 96 |
| 400 | 0.4387954 | 100 |

Table 8.1: Example excerpt from a resampled stream with BINSIZE=100.

We suspect that common root causes reflect on multiple streams. Assuming that streams start approximately simultaneously each day, the resampled datasets should capture the streams' behaviour during fixed intervals and be comparable to others. The method we choose for comparing aligned streams is *correlation*, described in Section 2.2.4.

To simplify and prepare the new dataset for correlation analysis, we perform a type of normalisation.[3] The first step increments the aggregated mean-delay series by their minimal value to mitigate negative delays. Given the speed of networking, unsynchronized hardware clocks sometimes result in negative nodal delay for packets. The second step subtracts the mean-delays by their new minimal value to move the stream as close to 0 as possible. This eliminates the varying minimum delays caused by the different geographical distances.

Our dataset is composed of multiple individual streams of unequal lengths. Sometimes errors occur while collecting the stream of packets, which result in incomplete datasets. A mitigation to this issue is needed. We want all streams to match the longest one. We therefore populate shorter streams with NaN values. This is automatically handled by pandas when adding new columns into dataframes. The missing values are then replaced by values considered to be normal behaviour in order to not create a false correlation. In essence, no packets are delayed or lost. The streams' mean-delays replace NaN delays. We name the preprocessed mean-delays for *pmeans*. The current bin size replaces NaN packet counts. Additionally, we calculate the packet loss ratio per bin $ppackets = 1 - (packets_{received}/BINSIZE)$. We create a variable *ppackets* to annotate preprocessed packet-counts. *ppackets* is

[2]Dataframe is a class in the pandas library that organises data in 2-dimensional tables.

[3]Not to be confused with normalisation in statistics where the entire dataset is mapped to the interval [0,1].

normalised in the interval [0,1] where a value of 0 represents no packet loss and a value of 1 represents maximum packet loss.

A simple function to capture both negative behaviours is the summation *sdf* of *pmeans* and *ppackets*, see Equation 8.1. *alpha* and *beta* can be used to bias the components. We choose to set them both equal to 1 because we want to weight them equally. The prior transformation will with this function create significant spikes if one or both values are abnormal. Finally, we calculate correlation based on scores from Equation 8.1 with the Pearson method, see Equation 8.2. Pandas provides a correlation method on dataframes for all possible combinations of variables. The code can be seen in Listing 8.1.

$$sdf = alpha \cdot pmeans + beta \cdot ppackets \tag{8.1}$$

$$r = \frac{n(\sum xy) - (\sum x)(\sum y)}{\sqrt{[n\sum x^2 - (\sum x)^2][n\sum y^2 - (\sum y)^2]}} \tag{8.2}$$

```
BINSIZE = 1000
means = pd.DataFrame()
packets = pd.DataFrame()

for i, f in enumerate(files):
    df = pd.read_csv(f)
    df = df.set_index('seq')

    means[i] = df.mean_delay
    packets[i] = df.packet_count

pmeans = means.replace( np.nan, means.mean() )
pmeans = pmeans.replace( 0, pmeans.mean() )
pmeans += pmeans.min() # ensure always positive
pmeans -= pmeans.min() # move to 0

ppackets = packets.replace(np.NaN, BINSIZE)
ppackets = 1 - (ppackets / BINSIZE)

alpha = 1
beta = 1

sdf = alpha*pmeans + beta*ppackets
sdf -= sdf.min()

corrdf = sdf.corr()
```

Listing 8.1: Correlation script

### Machine Learning Anomaly Detection

Instead of creating the anomaly score with the approach above, we let an unsupervised ML algorithm attempt to mark anomalies. Robust Random Cut Forest (RRCF) is a tree-based method, described in Chapter 2. This open-source algorithm is able to ”remember” prior observations when analysing the next input it receives [BMT19]. It supports large datasets generated from streams, which fits our problem. The output from RRCF is an anomaly score per input datapoint. The configuration parameters are set to create a forest of 40 trees, each capable of ”remembering” 256 datapoints. In other words, the trees have a max size of 256 datapoints. At this threshold, they throw away the oldest datapoint when they receive new input.

As an initial performance test, we feed the model with a simplified dataset of only packet loss. The implementation is the same setup as the developers' example in their source code [BMT]. The code can also be seen in Listing D.4. We choose an arbitrary stream and day that has experienced significant losses: the resampled stream with BINSIZE=1000 from Zurich to Auckland during the 8th of March 2021. The input can be seen in Figure 8.1. Why we choose not to continue with this

approach will become apparent in the results in Section 8.2. Finally, the result is discussed in Section 8.3.

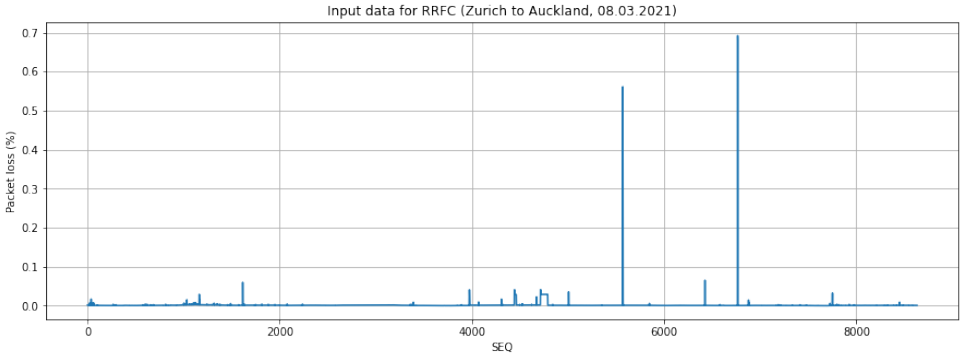



Figure 8.1: Input to RRCF (Zurich to Auckland, 2021.03.08).
Note, the y-axis unit is wrong. It is packet-loss ratio [0,1], **not** percent (%).

## 8.2 Results

With manual inspection, we select a few nodes from Dragonlab to display the results from correlation analysis. Their purpose is to show varying results that is discussed in Section 8.3. There are four types of figures. One type is a visual matrix representation of all-to-all correlation-scores[4] between streams. An example of this can be seen in Figure 8.2, with colours ranging from dark blue to bright yellow. A correlation score is in the interval [0,1], as displayed by the colour-bar on the right-hand side. Negative correlation is not included because we are only interested in common spikes. The other three types are coordinate systems where the x-axis represents the sequence number of packets, and the y-axis is some value for the respective bin of packets. Examples of the three plot types can bee seen in Figures [8.7, 8.8, 8.9]. This section begins with an overview of all streams, followed by a specific example from Amazon. For interested readers, the exact source and destination IP addresses can be found in Appendix G.

To avoid confusion when looking at the figures, we must explain the x-axis. The sequence number does not change even though the bin size is different. The raw CRUDE streams usually contain between 8 and 9 million packets. The resampled datasets have kept the original sequence numbers. For example, will the first row in a resampled file with BINSIZE=1000 be indexed with a sequence number equal to 1000. This causes the figures to have seemingly the same amount of datapoints when they do not.

### Bin Size Comparison

The correlation with the resampled datasets of bin sizes 100, 500 and 1000 packets can be seen in Figures 8.2, 8.3 and 8.4, respectively. Due to minimal difference between these figures, we choose bin size 1000 in further studies (see Section 8.3 for elaboration). All available Dragonlab streams during the 10th of March 2021 is used in this analysis. As expected, there is 100% correlation along the diagonal. There are also larger groups of higher correlation around the diagonal. The majority of the other comparisons have low scores, except for a few yellow dots and strips. The basis for correlation-scores can be seen in Figure 8.5. The figure displays the development of the Dragonlab-streams throughout the day.

[4]The basis for correlation scores is annotated as sum(pmeans, ppackets). *pmeans* are the preprocessed packet mean-delays and *ppackets* are the preprocessed packet-losses (decimal ratio [0,1]) for a given stream and BINSIZE.

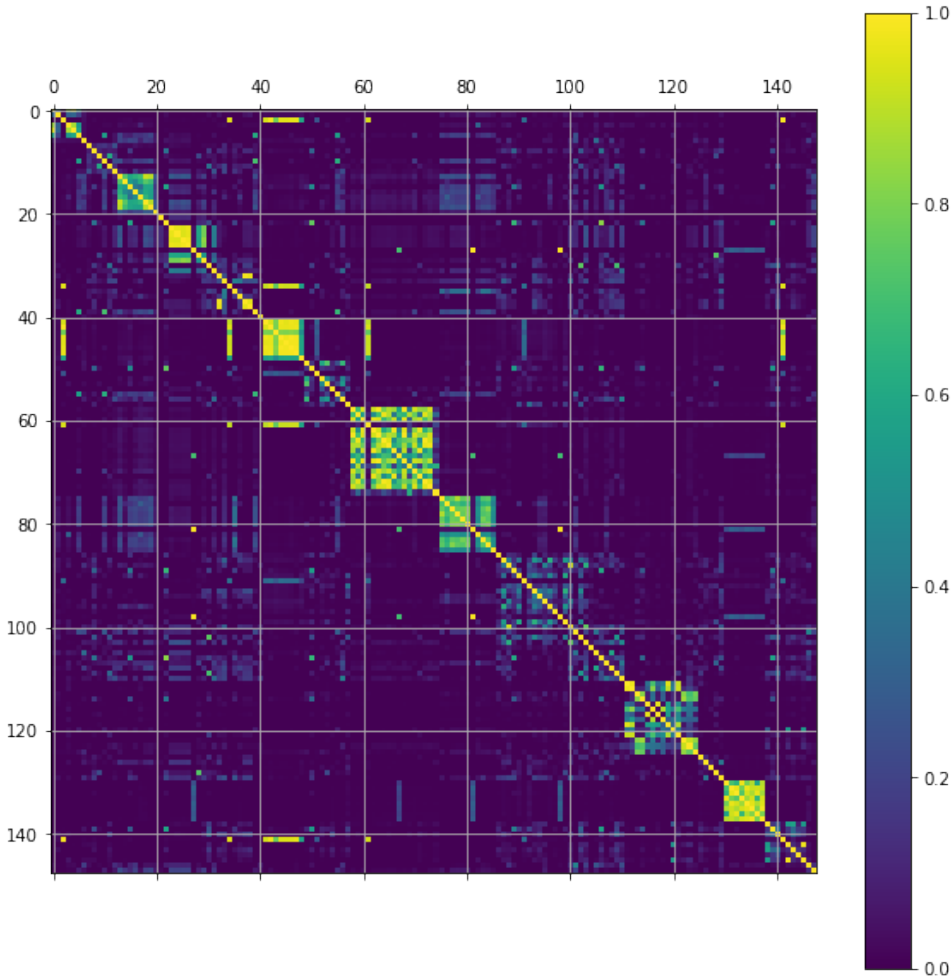



Figure 8.2: Correlation-scores for Dragonlab during 2021.03.10 with BINSIZE=100.

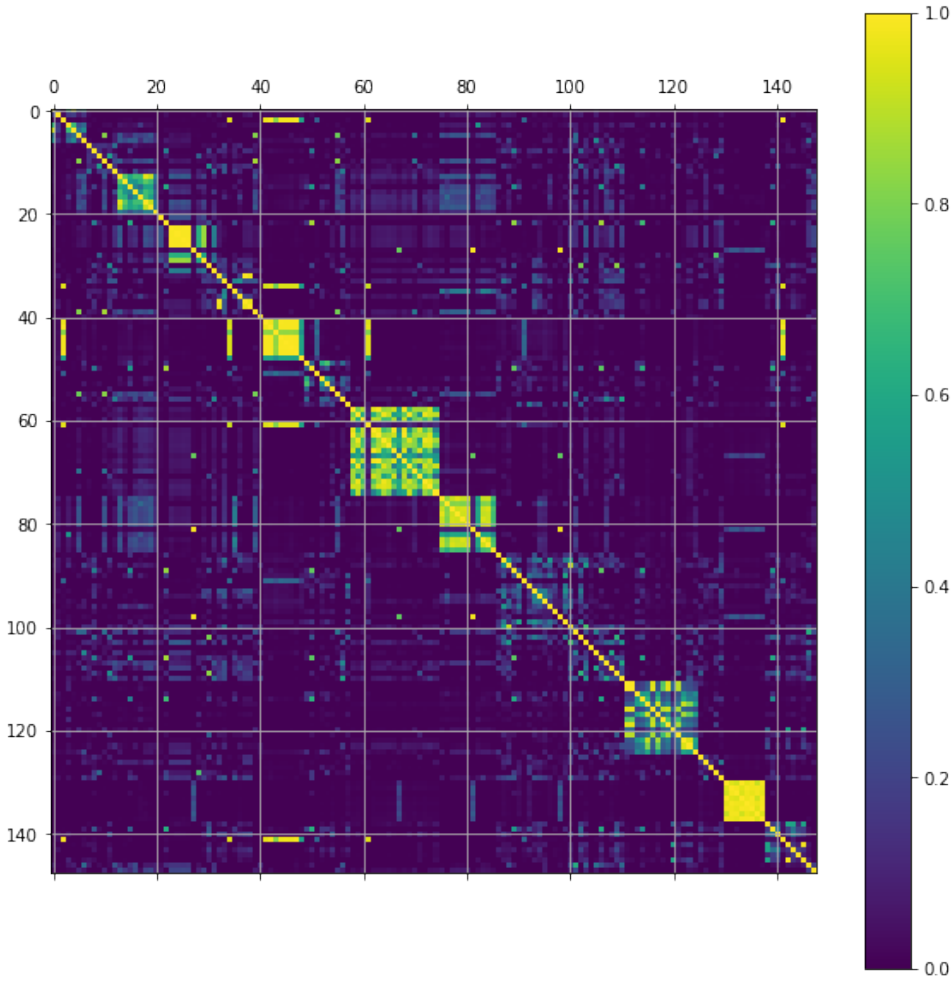



Figure 8.3: Correlation-scores for Dragonlab during 2021.03.10 with BINSIZE=500.

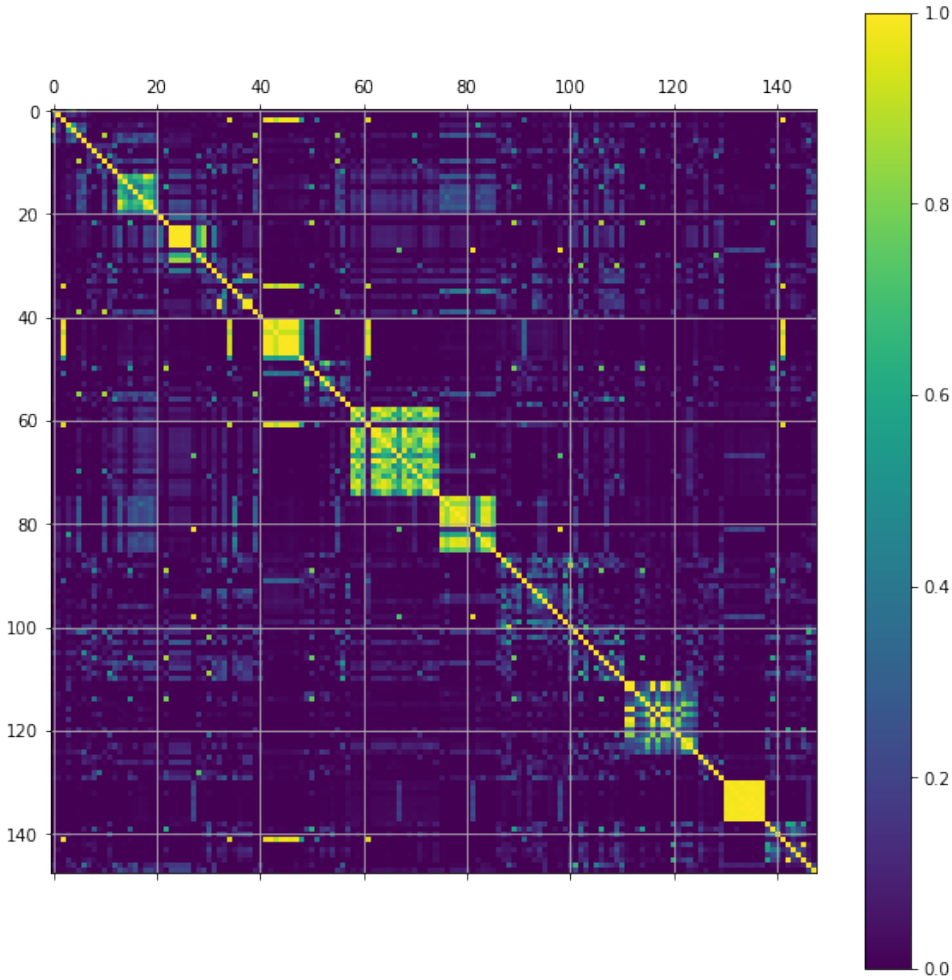



Figure 8.4: Correlation-scores for Dragonlab during 2021.03.10 with BINSIZE=1000.

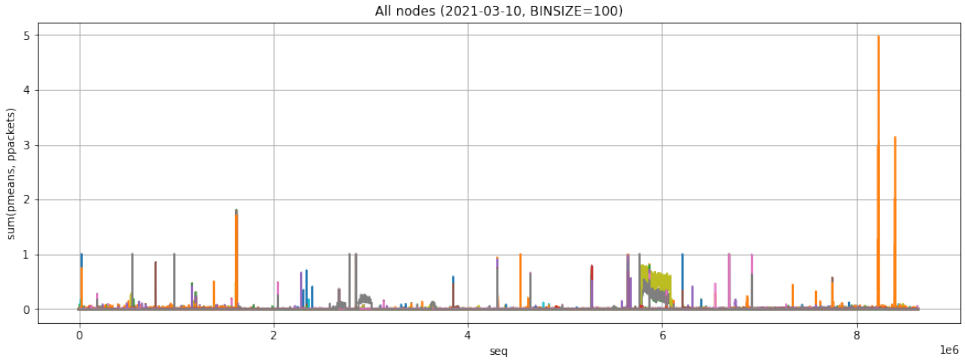



Figure 8.5: Plot of sum(pmeans, ppackets) (Dragonlab, 2021.03.10, BINSIZE=100).

### Correlations for Amazon

Correlation scores for Amazon during 2021.03.10 can be seen in Figure 8.6.[5] More figures can be found in Appendix F. Amazon (Figure 8.6) displays three pairs of strongly correlated streams (0-5, 0-6 and 5-6). Similar behaviour can also be seen in Figure 8.7 due to both higher mean-delay and packet-loss ratio around sequence 6 million.

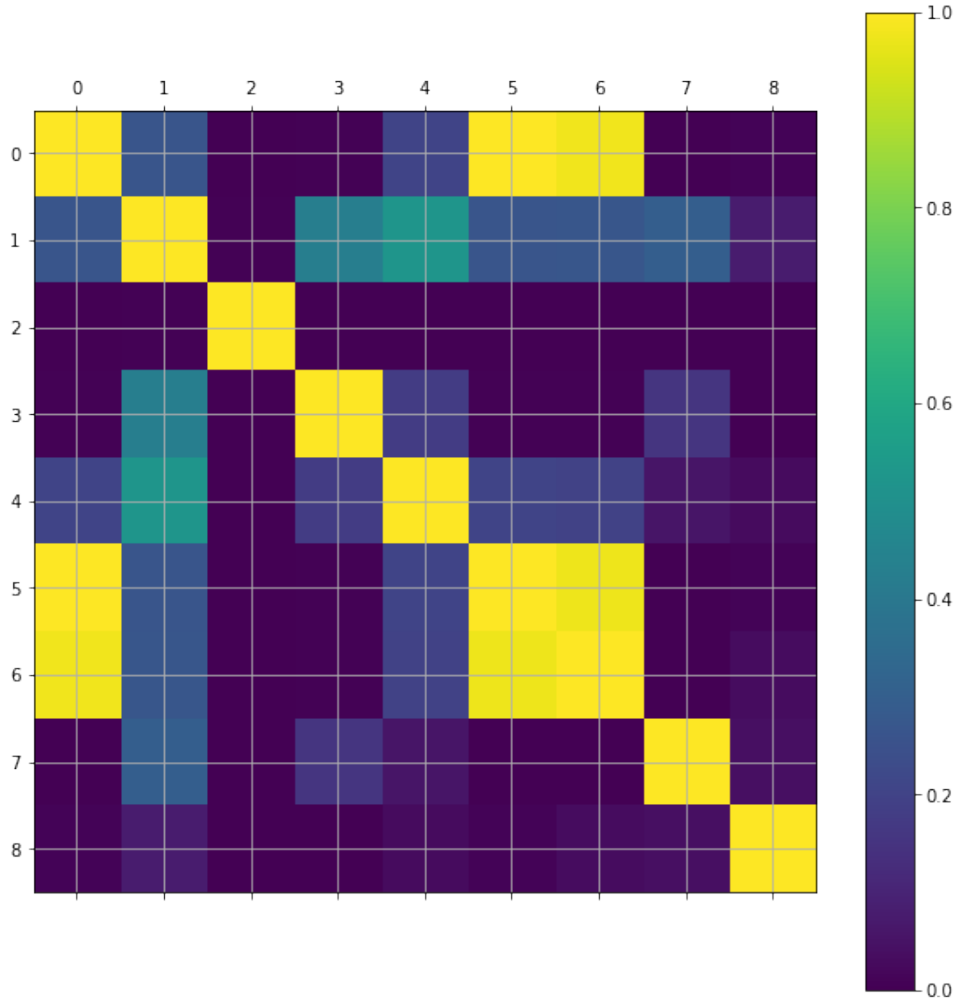



Figure 8.6: Correlation-scores for Amazon during 2021.03.10 with BINSIZE=1000.

[5]Any calculations "for" a node means that the node acted as the destination for gathered streams.

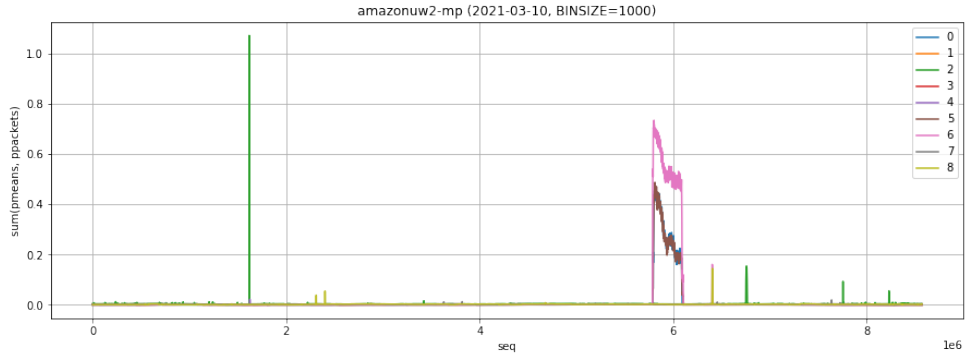



Figure 8.7: Plot of sum(pmeans, ppackets) (Amazon, 2021.03.10, BINSIZE=1000).

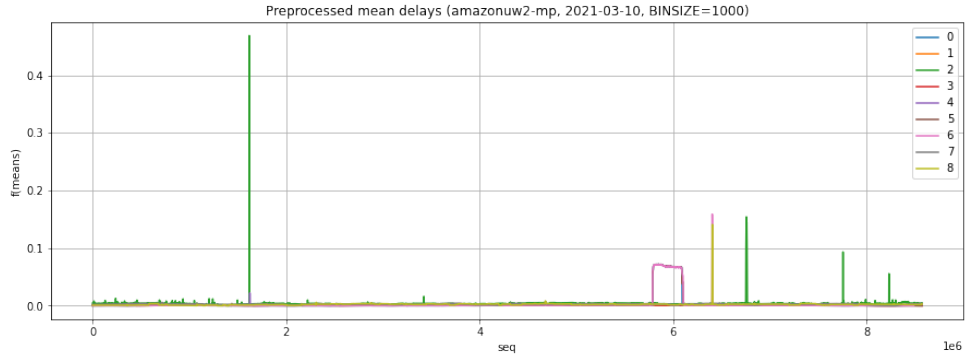



Figure 8.8: Plot of packet mean-delays (Amazon, 2021.03.10, BINSIZE=1000). The y-axis is measured in (ms).

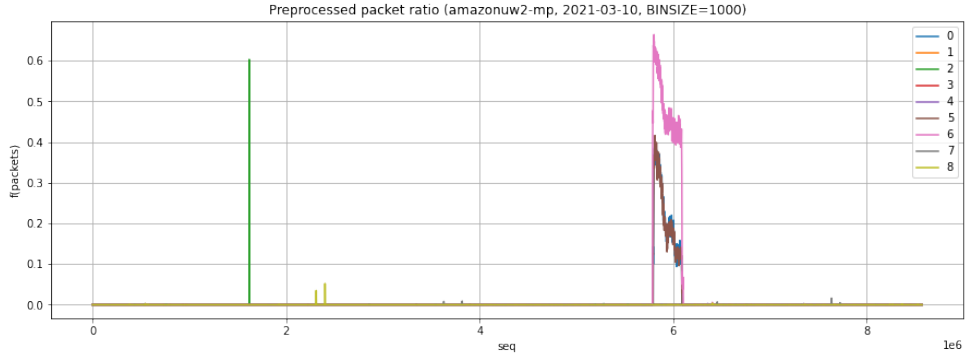



Figure 8.9: Plot of packet-loss (Amazon, 2021.03.10, BINSIZE=1000). The y-axis represents the packet-loss ratio [0,1].

### RRCF Result

In Figure 8.10 we see the plot of anomaly scores marked by RRCF with packet-loss from Zurich to Auckland (2021.03.08).

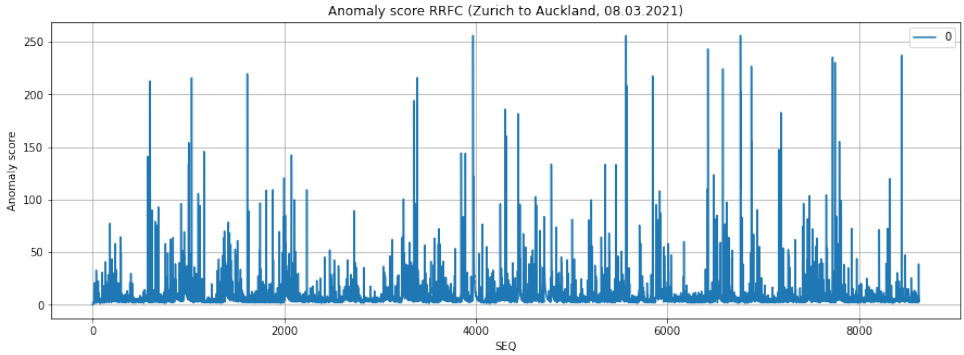



Figure 8.10: RRCF anomaly scores (Zurich to Auckland, 2021.03.08).

## 8.3 Discussion

The idea of this approach was provided by Yuming Jiang through multiple consultations in the duration of the project. It is inspired by his previous research on similar datasets [AEJ12, AJS18, AJH+14]. Locating issues is one of the first steps in RCA. This chapter, and the listed papers are also relevant to Chapter 9.

Key findings from cross-stream comparisons show that some streams do correlate with one another. These streams are often on the path to a common destination, but other incidents have also been found. This section will discuss possible reasons for these correlations. Uncertainties and how the analysis could have been improved is discussed in Section 10.2.

We have previously referred to Uninett's loss of traffic per month. Those results are relevant to this discussion, and we use the opportunity to enlighten details about how they are measured. Uninett analyses on a per-stream basis. That means the total loss is aggregated from each stream's perspective. If an outage of 10 seconds (caused by e.g. a router reboot) affects six separate streams, the aggregated loss equals a whole minute. In other words, the current measurements do not represent the duration of root causes. This distinction is important when analysing loss in the future. Findings from this chapter show that one must also be vary of the possible number of streams that are affected by disruptions.

In this chapter, we compare three different bin sizes of 100, 500 and 1000 packets. Given the frequency of 100 packets per second, each bin equals 1, 5 and 10 seconds, respectively. Ten seconds is considered to be a long time in networking. To put it in perspective, routers can handle millions of packets per second. Because of this reason, it is questionable whether or not such a low packet frequency accurately represents the network's behaviour.

With the obtained results, we notice that the difference in correlation between 1s and 10s bin sizes is marginal (see Figures 8.2, 8.3, 8.4). Larger bin sizes tend to be slightly stronger correlated than smaller bin sizes. We expect this result because a larger window has a higher probability of capturing common behaviour, especially if streams have time unit lags. We choose to only include result figures with bin size equal to 1000 packets given the negligible difference. The project is more interested in proof of principle than precise numbers. One could also argue that a larger bin size is beneficial with respect to memory, as only a fraction of space is needed.

The results include plots of correlation analysis on streams to Amazon. Three identified pairs are highly correlated, shown in Figure 8.6. Given the distinct simultaneous pattern in Figure 8.7, there is a high probability that the streams were affected by the same root cause. Around the 6 million mark, there is a sudden

spike in both mean-delay (Figure 8.8) and packet-loss (Figure 8.9). Each datapoint represents 10 seconds, so the issue has caused abnormal behaviour for a significant amount of time. The most recognisable feature is the pattern in mean-delay. They all display a decrease in delay before coming to an abrupt return to normal behaviour at the same time.

Infrastructure is a keyword in these results. The Internet is a web of links and routers transferring data with a best-effort protocol. Paths are constantly switched for numerous reasons, e.g. congestion, severed wires or hardware reboot. Complex networking protocols are capable of automatic rerouting when any of these events occur. The streams in this project also utilise this network and are likely to share the same components. Supported by a high correlation between streams of same the destination, root cause relates to a failure in a component close to the paths' endpoints. The self-healing ability of networks can be a slow process; therefore, it is not unlikely that Uninett's monitoring is capable of capturing disrupting events.

These results could be helpful in further analysis, based on for instance ML. This information indicates that highly correlated stream-pairs could isolate a set of critical components in the future. Especially interesting are highly correlated stream-pairs without a common source or destination. Matching this information with context data, e.g. traceroutes, might reveal the root cause of an incident.

The anomaly scores from RRCF implementation does not resemble the input. Listing D.4 successfully identifies anomalies in the paper [BMT19] but struggles with the dataset from Uninett. We see limited potential for further investigations with this approach. One can also argue that the input data already is an anomaly score by itself, meaning that anomaly detection methods are redundant.

# Chapter 9
# Root Cause Analysis

In this chapter, a Root Cause Analysis (RCA) is conducted in order to determine the root cause of the network delay detected by Kibana's anomaly detector in Chapter 6. The work performed in said chapter is an important step in the problem understanding. The scope of the analysis will be the two paths from Oslo and Trondheim to NGU, and the theory introduced in Section 2.5.

## 9.1 Implementation

Traceroute is introduced in Subsection 2.1 as a computer network diagnostic tool that traces possible routes from source to destination and measures transit delay of packets. Uninett performs traceroute on their links in the network regularly. Every executed traceroute is timestamped and entered into a report for the individual link. In the interest of discovering the root cause of the problem, an analysis of the exposed paths is performed to determine if one or more shared nodes are responsible for the anomalous behaviour. The traceroute reports from April 8th 2021, for the links between Oslo and Trondheim to NGU serve as the problem cause data.

To map the nodes in the paths, the python script displayed in Listing D.5, traverses the reports and registers the IP addresses. The measured transit delay and maximum experienced delay is collected from the traceroutes' output, and is used to calculate the average delay for each node in the paths. The script appends the addresses to a list representing the nodes in the path. The list is then entered into a dictionary with the source address as key. The code displayed in Listing 9.1 takes the dictionary as parameter and discovers shared nodes by comparing the list.

```
import functools

dst_nodes = []
shared_nodes = []

for node in ipt:
    dst_nodes.append(ipt[node])

def findCommon(L):
    def R(a, b, seen=set()):
        a.update(b & seen)
        seen.update(b)
        return a
    return functools.reduce(R, map(set, L), set())

print('The paths share the following nodes: ')
print('')
for node_ip in findCommon(dst_nodes):
    shared_nodes.append(node_ip)
    print(node_ip)
```

Listing 9.1: Code snippet for determining shared nodes in paths.

Uninett has already implemented a tool to visualise the connection between endpoints of a link, illustrated in Figure 1.1. By utilising traceroutes, topology layouts of these connections are automatically generated. Investigating the topologies can confirm the detected shared nodes from the programming scripts, and might provide further insight.

## 9.2 Results

Figure 9.1 presents the results from the script Listing D.5 introduced in Section 9.1. It lists the detected nodes shared by the two paths, with associated average and maximum delay measured in milliseconds (ms). Closer inspection reveals a significantly higher maximum delay for the node with IP address 193.156.55.218, in addition to the highest average delay.

```
The paths share the following nodes:

Node: 128.39.254.183 - Average delay: 4.864 ms - Max delay: 12.091
Node: 193.156.2.1 - Average delay: 4.825 ms - Max delay: 40.799
Node: 193.156.55.218 - Average delay: 6.024 ms - Max delay: 147.103
Node: 128.39.255.183 - Average delay: 4.646 ms - Max delay: 12.927
```

Figure 9.1: Shared nodes in paths discovered by Listing 9.1.

We select traceroutes based on the detected anomalies during the 8th of April. From these records, Uninett has generated the path topologies [Unia] displayed in Figure 9.2 and 9.4. The figures illustrate the paths utilised from Oslo and Trondheim to NGU, with corresponding information in Tables 9.3 and 9.5. The hop sequence starts from the left, and the arrows indicate the direction of the path. Two vertically placed nodes indicate the same hop with alternative nodes, reflected in the tables by an unnumbered hop. Routers are highlighted in the tables with their related IP addresses, along with maximum measured delay.

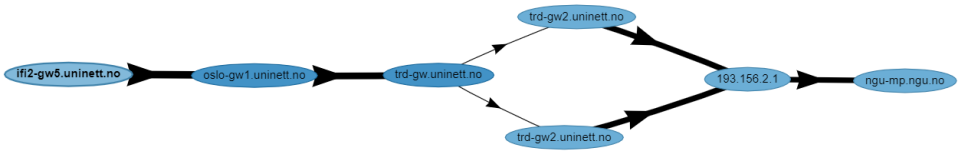



Figure 9.2: Path topology from Oslo to NGU.

| Hop | Router | Avg ms | Min | Max | Sdv | Loss% | Seen | Address | Start | End |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | ifi2-gw5.uninett.no | 1.2 | 0.3 | 34.0 | 2.8 | 0.00% | 7794 | 158.39.1.125 | 08 00:00:07 | 08 23:59:08 |
| 2 | oslo-gw1.uninett.no | 0.9 | 0.3 | 41.2 | 2.5 | 0.00% | 8046 | 128.39.254.82 | 08 00:00:07 | 08 23:59:08 |
| 3 | trd-gw.uninett.no | 8.6 | 7.8 | 50.7 | 3.0 | 0.00% | 8040 | 128.39.255.25 | 08 00:00:07 | 08 23:59:08 |
| 4 | trd-gw2.uninett.no | 8.5 | 8.1 | 10.1 | 0.2 | 0.00% | 3994 | 128.39.254.183 | 08 00:00:07 | 08 23:59:08 |
| | trd-gw2.uninett.no | 8.6 | 8.1 | 12.9 | 0.3 | 0.00% | 4052 | 128.39.255.183 | 08 00:00:07 | 08 23:59:08 |
| 5 | 193.156.2.1 | 8.6 | 8.2 | 40.8 | 1.5 | 5.20% | 7628 | 193.156.2.1 | 08 00:00:07 | 08 23:59:08 |
| 6 | ngu-mp.ngu.no | 9.6 | 8.2 | 147.1 | 9.4 | 11.76% | 7100 | 193.156.55.218 | 08 00:00:07 | 08 23:59:08 |

Figure 9.3: Table of hops from Oslo to NGU.

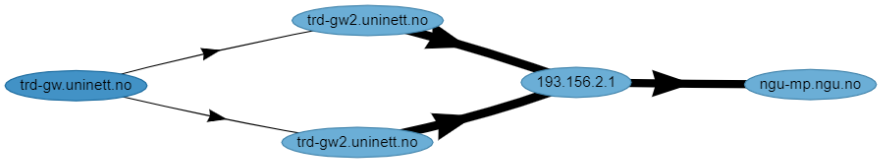



Figure 9.4: Path topology from Trondheim to NGU.

| Hop | Router | Avg ms | Min | Max | Sdv | Loss% | Seen | Address | Start | End |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | trd-gw.uninett.no | 0.8 | 0.2 | 22.3 | 2.1 | 0.00% | 7992 | 158.39.1.89 | 08 00:00:07 | 08 23:58:43 |
| 2 | trd-gw2.uninett.no | 0.9 | 0.5 | 4.6 | 0.3 | 0.00% | 3996 | 128.39.254.183 | 08 00:00:07 | 08 23:58:43 |
| | trd-gw2.uninett.no | 0.9 | 0.5 | 6.0 | 0.2 | 0.00% | 4037 | 128.39.255.183 | 08 00:00:07 | 08 23:58:43 |
| 3 | 193.156.2.1 | 1.0 | 0.5 | 19.3 | 0.7 | 5.65% | 7580 | 193.156.2.1 | 08 00:00:07 | 08 23:58:43 |
| 4 | ngu-mp.ngu.no | 2.3 | 0.5 | 144.3 | 10.5 | 12.76% | 7009 | 193.156.55.218 | 08 00:00:07 | 08 23:58:43 |

Figure 9.5: Table of hops from Trondheim to NGU.

## 9.3 Discussion

In the previous section, results, the Figure 9.1 shows that the implemented program found four distinct common nodes. The two generated topology layouts, in Figures 9.2 and 9.4, contain the same nodes and support the results. The link from Oslo to NGU utilises the same nodes as Trondheim to NGU, which explains why they both experience anomalous behaviour in Chapter 6, almost simultaneously. The node with IP address 193.156.55.218 stands out with the highest maximum measured delay and calculated average delay. It is also highlighted in the tables shown in Figures 9.3 and 9.5. Uninett calculates the packet loss of each node, and ngu-mp.ngu.no displays a significantly higher loss than the other nodes in the path. These results help isolate the cause of the problem to a single node or its links. To conclusively determine the cause, further investigation into more detailed information sources, such as SNMP messages or router-logs, must be conducted.

The last two steps in the analysis are root cause identification and solution recommendation before implementation. Identifying the cause can be difficult and relies heavily on a comprehensive understanding and definition of the problem, in addition to a solid data collection. In the conducted analysis, employees at Uninett contribute valuable knowledge to gain an understanding of the problem. The data collection mainly consists of traceroute reports, and all the information is aggregated from them. The issue of using traceroute is its granularity compared to the CRUDE implementation. Traceroutes are sent only once per minute. In other words, a problem could occur and be resolved between two traceroutes without being registered. CRUDE measurements, on the other hand, are performed 100 times per second.

# Chapter 10
# Discussion

The current chapter provides a final discussion of the applicability of machine learning at Uninett. It also elaborates about the challenges we faced during the project before wrapping up with recommended improvements and future work.

## 10.1 Applicability of Machine Learning at Uninett

To evaluate the implementations and results from the iterated approaches, we revisit the problem description and the research questions (RQ) from Section 1.3.

The problem description describes the project's objectives and end goal. Intended methods have been applied, and most objectives fulfilled. However, due to the challenges encountered and the limited amount of time available, some objectives were not completed. We did, for example, not perform fine-tuning of the selected algorithms we implemented. Forecasting is also mentioned as a possible approach, but there was not enough time to thoroughly evaluate it. We briefly looked into forecasting and single-step prediction for anomaly detection and saw little potential to trigger further investigations. The reason for this is rooted in Uninett's reliable system that generates monotonous data without clear patterns related to detected issues. Training a forecasting/prediction model with CRUDE data, means seeing millions of packets where only a tiny proportion is abnormal. This means that any such algorithm will likely always predict normal behaviour and will almost always be correct. Anyhow, we believe we have succeeded in the problem description's goal of finding valuable results for Uninett's engineers. The project contributes to the evaluation of machine learning's future in Uninett, finds interesting results in Chapter 8 and suggests further improvements in their system.

*RQ1: Which requirements need to be fulfilled to facilitate machine learning?* A clear understanding of the datasets is essential before applying an ML algorithm. Without knowledge of the input, the output can be incomprehensible. Often a set of properties must be fulfilled in order to achieve meaningful results. If the applied data

is time-dependent, we desire either distinct trends, patterns or cycles. The data has to be prepared before it can be ingested into an algorithm. Timestamps are crucial in time series data, and factors like time zones must be taken into consideration.

In examining the CRUDE data and gap logs in Chapters 5 and 7, we discover limited informational value or context to categorise or predict gaps. It is evident that most features available in the datasets are derived from nodal delay. Uninett can extract information from a combination of different resources in their system to counter the shortcomings in each of them. Its easier to analyse a dataset comprised of independent variables with a set of standalone features. We elaborate on this in Section 10.3. Machine learning is limited by the data fed into the algorithm, meaning that its greatest strength is also its biggest weakness.

*RQ2: Can Uninett utilise machine learning in their already existing network monitoring system?* The short and simple answer is, yes, Uninett can apply machine learning in their existing system. But the quality of its output is questionable and may not be worth much without undergoing the preparatory improvements. Uninett's available data is best suited for anomaly detection but less applicable to root cause analysis because of its lacking context and low dimensionality. All the data stem from a continuous stream of packets in a relative stable system, resulting in few abnormalities, as discussed in Chapter 5. This makes fault detection easy but causation ascertaining hard. In Chapter 6, Kibana's ML-based anomaly detection discover distinct outliers in the measured delay. These incidents are already uncovered by Uninett's analysis using thresholds, where the only difference is the labelling of the anomalies. The detector can only monitor one data field and contribute no context to its results. Automated machine learning solutions, such as Kibana's, are easy to implement and maintain but leaves little room for users to modify them. A conventional ML solution is complex and demands experts with deep knowledge in the field to fully exploit it. It entails manual inspection of the data and maintenance, which makes for a costly affair.

*RQ3: How can Uninett improve its network monitoring to determine root causes of faults better?* In Chapter 8, cross-stream comparisons are conducted to see if traditional analysis could produce promising results. We find a noticeable correlation between stream-pairs, information that could be used to isolate critical components in the network. The root cause analysis performed in Chapter 9 uses traceroute reports to look at average transit delay in paths. It points out a router, shared by two paths that experience anomalous delay, as a possible cause of the problem. By combining these two approaches, or match incidents with other datasets e.g. SNMP traps, syslogs, router-logs, BGP-updates, the joint dataset might reveal new information. This is a difficult task and has been attempted before with mixed success. In order to match different logs, one must be able to match them on timestamps. Granularity,

i.e. the level of details in the data, then becomes a critical factor. CRUDE data have relatively low granularity compared to routers who experience state changes millions of times per second. Other datasets, such as traceroutes and BGP-updates, are measured even less often. Traceroutes are only sent by Uninett once a minute and are very unlikely to capture gap-causing events in CRUDE data. Theoretically, nearly 50 gaps can occur in between each time a traceroute is sent.

## 10.2 Challenges

One of the greatest challenges we faced during the project is the limited information in the available datasets. In the early phase, we were introduced to previous work done at Uninett by a summer intern. The project aimed to match multiple datasets on timestamps to obtain context data of detected network issues. The data was gathered from system-logs, router-logs, Simple Network Management Protocol (SNMP) traps, traceroutes, gap-events and Border Gateway Protocol (BGP)-updates. Findings from this approach would have been interesting to apply in our ML evaluation. Unfortunately, this project was not made available to us. Therefore, an investigation of ML applicability on such a resource still remains as future work.

This project experienced timestamp challenges related to asynchronous hardware clocks. During manual inspections on CRUDE data, we found negative nodal delays. For this reason, an uncertainty accompanies all comparisons between multiple logs matched on timestamps.

The correlation analysis in Chapter 8 assumes that streams start simultaneously. This is likely not the case every day and is a known source of error that should be addressed in future studies. There are possibly more correlated streams not discovered by our analysis. We recommend that future analysts are attentive to time zones and timestamps when aligning streams. A possible solution is to utilise the same approach we use, in addition to prepend missing segments as well.

One particular limitation in this thesis was the quality of the data used in the research. The datasets consist of synthetic traffic produced by Uninett to monitor the paths in their network. Due to strict General Data Protection Regulation (GDPR), service providers are restricted insight into user data. Netflow data contains more information and would have been a richer data source. Additionally, there was a limitation in terms of assessing the quality of the provided anomaly detection methods. Because of ML-methods' complexity and the level of knowledge required, the true quality of the solutions must be determined by experts in the field.

Considering Uninett's position as an ISP, necessary security measures prolonged the time to obtain the required access to their server and data interface. Some

features were only available for users with sufficient permission levels such as admins, which we could not receive seeing we are not employees. This resulted in delaying the practical implementation with respect to the initially planned schedule.

The large amount of data we evaluate in this thesis was a severe challenge. Manual inspection by humans on network traffic data is an infeasible task. Due to the data size to be prepared and analysed during the project, a simple mistake could result in hours of rework. Quality assurance of the developed scripts was a time-consuming task. For example, it was not always easy to validate that applied functions on millions of data lines were performed correctly.

The thesis was conducted during the COVID-19 pandemic, which limited all contact with the supervisor, academic staff, and between ourselves to strictly digital interaction. In hindsight, the counselling would have been more productive in physical meetings between the students and professors, considering the complexity of the thesis' subject.

## 10.3 Future Work

The thesis mentions combining multiple different types of datasets as future work. This section elaborates details and suggestions for such an approach. We believe that collecting the circumstances to known events and grouping them to independent variables will result in a richer resource.

Uninett can use their historical experience to pinpoint a few known events that caused issues in their network. From there, they can gather all available monitoring data within a defined time interval and analyse them thoroughly. Let us say the data is structured as a table. The goal is to achieve a single row per event with many columns. The columns are the features that describe a single observation. This project faced the issue of low dimensionality. Therefore, Uninett should consider increasing it. One of the columns should optimally be the classification of the issue (also known as the 'label' or 'ground truth'). Creating a set of labelled data can contribute in a semi-supervised algorithm. Related work in Section 3.2 finds semi-supervised methods to improve the prediction accuracy with only a small amount of labelled data. The other columns can contain a variety of information extracted from multiple logs. Uninett can even consider *binarisation* as the initial approach. That means to mark the presence of something. An example in this context: create a column and mark if there was sent an SNMP trap within the interval or not. This might reveal logs that frequently are present in certain issues.

Chapter 5 raises an issue regarding the data format in their CRUDE data. Their current logging braids multiple streams into an impractically structured compressed

file. There is a fixed set of attributes in this dataset, which makes it suitable for CSV. CSV is a format where values are separated by either comma, space or tab. The first line represents the header and contains the name of each column in the data. Every new line of values conforms to the header and represents a row in the table. This universal format enables the usage of multiple acknowledged analysis tools such as pandas, numpy and ML libraries. We recommend adopting an atomic mindset, breaking their information down into smaller and more consistent components. Separating CRUDE streams into separate files, for example, facilitates analysis on single streams over more extended periods. We also suggest that Uninett be consistent with directories on their server. The paths to raw logs represent each file's metadata, such as domain, host, IP address or timestamp. Over time, Uninett has changed the location and naming conventions of their data, making it challenging to perform post analyses on historical data.

The scripts that parse and resample CRUDE data in this project are executed in a python environment without multi-threading. Because we do not utilise multi-threading to spread processes on Uninett's 64 core CPU, the files are parsed in a sequential manner. This is a tedious task and is not recommended. Given the amount of data on Uninett's server, we suggest developing new scripts with threading to parse their old data into better formats, e.g. CSV in the paragraph above.

# Chapter 11
# Conclusion

In the duration of this thesis, we explore multiple datasets and evaluate them with respect to ML. Through different approaches, we examine if available properties in the datasets can facilitate ML algorithms. Kibana's ML based anomaly detection is tested to find anomalies in measured delay, and we perform root cause analysis to determine the underlying cause.

We discover that there is little information to find in the datasets. Most features are derived from a single metric, which leaves little context to work with for ML. Uninett's logs are not yet cross-matched nor labelled. By themselves, they do not produce enough context data to reveal new information with more complex ML algorithms. The correlation between stream-pairs yielded promising results. We discover correlations between different paths, which indicate that unwanted events affect multiple parts of the infrastructure. In combination with other sources, it may enhance Uninett's network monitoring to determine and label the root cause.

We have concluded that as of today, there are high costs with few benefits from introducing ML to Uninett's monitoring system. Our findings show that Uninett has already established a solid and robust system capable of collecting data and detecting issues by statistical methods. In this matter, they uncover unwanted behaviour such as jitter, gaps, and delay that exceed acceptable tolerances. We find the same anomalies Uninett do when we test anomaly detection, only with a grading score instead of a label. Adding ML would introduce unnecessary complexity and would require expertise to develop and maintain. Considering we achieve the same results as threshold methods, the presented ML approach seems redundant, and we suggest that Uninett instead expands on their existing data analysis.

Essential future work to facilitate a reevaluation of the applicability of ML is improving the way data is stored and handled. For example, the way Uninett stores data would be significantly enhanced by introducing CSV format. It would increase the reading speed of the files and also allow for the use of advanced tools.

In addition, to fully utilise the advantages of ML, more preparation of the data needs to be conducted. By grouping data and labelling known incidents, a semi-supervised machine learning method can be considered. It only requires a small amount of labelled data combined with a larger amount of unlabelled data for training. According to related work, it has an improved prediction accuracy compared to other methods.

# References


[ABM+18] Giuseppe Aceto, Alessio Botta, Pietro Marchetta, Valerio Persico, and Antonio Pescap. A comprehensive survey on internet outages. *Journal of Network and Computer Applications*, 113:36–63, 2018.

[AEJ12] Atef Abdelkefi, Yaser Efthekhari, and Yuming Jiang. Locating disruptions on internet paths through end-to-end measurements, 2012.

[AF06] Bjørn Andersen and Tom Fagerhaug. *Root cause analysis: simplified tools and techniques*. Quality Press, 2006.

[AJH+14] Atef Abdelkefi, Yuming Jiang, Bjarne Emil Helvik, Gergely Biczók, and Alexandru Calu. Assessing the service quality of an internet path through end-to-end measurement. *Computer Networks*, 70:30–44, 2014.

[AJS18] A. Abdelkefi, Y. Jiang, and S. Sharma. Senatus: An approach to joint traffic anomaly detection and root cause analysis. In *2018 2nd Cyber Security in Networking Conference (CSNet)*, pages 1–8, 2018.

[ALPA17] Subutai Ahmad, Alexander Lavin, Scott Purdy, and Zuha Agha. Unsupervised real-time anomaly detection for streaming data. *Neurocomputing*, 262:134–147, 2017. Online Real-Time Learning Strategies for Data Streams.

[BD02] Peter Brockwell and Richard Davis. *An Introduction to Time Series and Forecasting*, volume 39. Springer, 01 2002.

[BMT] Matthew Bartos, Abhiram Mullapudi, and Sara Troutman. rrcf: Implementation of the robust random cut forest algorithm for anomaly detection on streams (github). [Online]. Available: https://github.com/kLabUM/rrcf. Accessed: 03.06.2021.

[BMT19] Matthew Bartos, Abhiram Mullapudi, and Sara Troutman. rrcf: Implementation of the Robust Random Cut Forest algorithm for anomaly detection on streams. *The Journal of Open Source Software*, 4(35):1336, 2019.

[BSL+18] R. Boutaba, Mohammad Salahuddin, Noura Limam, Sara Ayoubi, Nashid Shahriar, Felipe Estrada-Solano, and Oscar Caicedo Rendon. A comprehensive survey on machine learning for networking: Evolution, applications and research opportunities. *Journal of Internet Services and Applications*, 9, 05 2018.

[CBK09] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM Comput. Surv.*, 41, 07 2009.

[Cha00] Chris Chatfield. *Time-series forecasting*. CRC press, 2000.

[dev] Python developers. Pathlib.glob documentation. [Online]. Available: https://docs.python.org/3/library/pathlib.html#pathlib.Path.glob. Accessed: 03.06.2021.

[Ela] Elastic. [Online]. Available: https://opendistro.github.io/for-elasticsearch-docs/docs/ad/. Accessed: 11.05.2021.

[FJ93] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, 1993.

[GH13] Shirley Gregor and Alan Hevner. Positioning and presenting design science research for maximum impact. *MIS Quarterly*, 37:337–356, 06 2013.

[GKRB13] Nico Görnitz, Marius Kloft, Konrad Rieck, and Ulf Brefeld. Toward supervised anomaly detection. *J. Artif. Int. Res.*, 46(1):235–262, January 2013.

[GU16] Markus Goldstein and Seiichi Uchida. A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data. *PloS one*, 11:e0152173, 04 2016.

[HA18] Rob J Hyndman and George Athanasopoulos. *Forecasting: principles and practice*. OTexts, 2018.

[JLj] Rui Prior (rprior@inescporto.pt) Juha Laine (juha.laine@soon.fi), Sampo Saaristo (sambo@cs.tut.fi). Crude software. [Online]. Available: http://rude.sourceforge.net/. Accessed: 18.06.2021.

[KR16] James F. Kurose and Keith W. Ross. *Computer Networking: A Top-Down Approach*. Pearson Education Limited, 2016.

[Kvi19] Olav Kvittem. Measuring micro-dependability. [Online]. Available: https://ripe79.ripe.net/wp-content/uploads/presentations/98-ripe-mat-microdep-2019-10-17.pdf, 2019. Accessed: 11.05.2021.

[LA15] Alexander Lavin and S. Ahmad. Evaluating real-time anomaly detection algorithms – the numenta anomaly benchmark. *2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*, pages 38–44, 2015.

[Mad07] Henrik Madsen. *Time series analysis*. CRC Press, 2007.

[MG16] Andreas C. Müller and Sarah Guido. *Introduction to Machine Learning with Python*. O'Reilly Media, Inc., 2016.

[MHH+10] Eugene Myakotnykh, Bjarne Helvik, Jon Hellan, Olav Kvittem, Trond Skjesol, Otto Wittner, and Arne Oslebo. Analyzing causes of failures in the global research network using active measurements. pages 565–570, 10 2010.

[RH04] James J Rooney and Lee N Vanden Heuvel. Root cause analysis for beginners. *Quality progress*, 37(7):45–56, 2004.

[RPV06] S. Rajasekar, Philomi nathan Pitchai, and Chinnathambi Veerapadran. Research methodology. 01 2006.

[SCFJ03] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. Rtp: A transport protocol for real-time applications. RFC 3550, RFC Editor, July 2003.

[TS20] Emil Telstad and Mats O. Sannes. Machine learning for multi-source analysis. Project report in TTM4502, Department of Information Security and Communication Technology, NTNU – Norwegian University of Science and Technology, Dec. 2020.

[Unia] Uninett. [Online]. Available: https://iou2.uninett.no/microdep/. Accessed: 11.05.2021.

[Unib] Uninett. Dragonlab. [Online]. Available: https://www.uninett.no/node/1819. Accessed: 4.05.2021.

[Unic] Uninett. Uninett webpage. [Online]. Available: https://www.uninett.no/. Accessed: 17.11.2020.

[WCD16] Sarah Wassermann, Pedro Casas, and Benoit Donnet. Machine learning based prediction of internet path dynamics. 12 2016.

[Wie14] R. J. Wieringa. *Design science methodology for information systems and software engineering*. Springer, 2014.

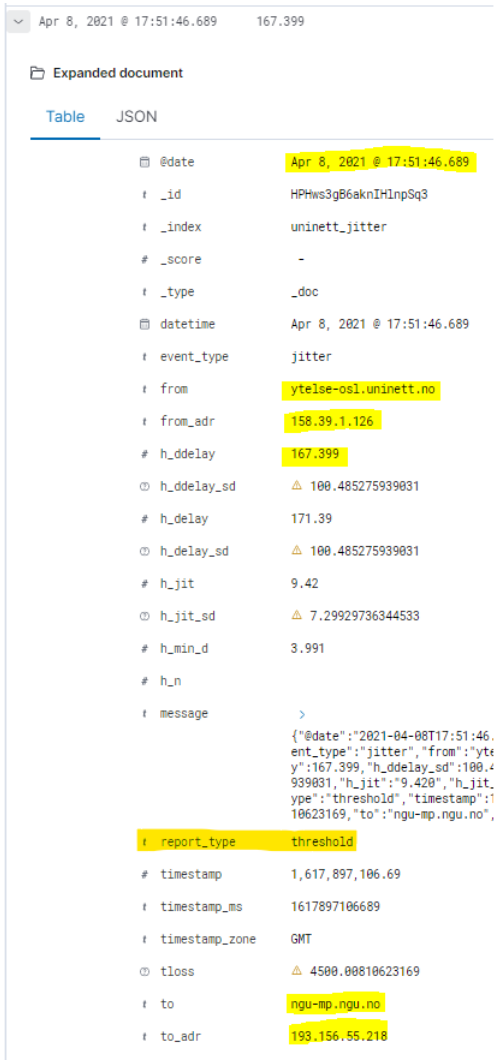
# Appendix A

# Kibana Anomaly Detection

This appendix includes figures from Kibana's anomaly detection in Chapter 6.


Figure A.1: Uninett's data indexed and labelled in Kibana.


(a) Documents with measurements in Uninett's index shown in Kibana. Anomalies with measured delay highlighted with yellow colour.




(b) Expanded document in Uninett's index shown in Kibana. Data of interest highlighted with yellow colour.

# Appendix B
# Traceroute and Root Cause Analysis

This appendix includes figures from RCA in Chapter 9, and theory expanding on root cause analysis.

## B.1 Traceroute

| Time | from | to | path_avg_rtt |
|---|---|---|---|
| Apr 8, 2021 @ 17:54:58.000 | ytelse-trd.uninett.no | ngu-mp.ngu.no | 0.3508 0.79 0.7752 0.907 |
| Apr 8, 2021 @ 17:54:58.000 | ytelse-trd.uninett.no | ngu-mp.ngu.no | 0.3508 0.79 0.7752 0.907 |
| Apr 8, 2021 @ 17:53:50.000 | ytelse-trd.uninett.no | ngu-mp.ngu.no | 0.3467 1.013 0.9087 0.9297 |
| Apr 8, 2021 @ 17:53:50.000 | ytelse-trd.uninett ⊕ ⊖ | ngu-mp.ngu.no | 0.3467 1.013 0.9087 0.9297 |
| Apr 8, 2021 @ 17:52:46.000 | ytelse-trd.uninett.no | ngu-mp.ngu.no | 0.3198 0.7987 0.7352 0.9127 |
| Apr 8, 2021 @ 17:52:46.000 | ytelse-trd.uninett.no | ngu-mp.ngu.no | 0.3198 0.7987 0.7352 0.9127 |
| Apr 8, 2021 @ 17:51:41.000 | ytelse-trd.uninett.no | ngu-mp.ngu.no | 0.338 0.745 0.864 1.202 |
| Apr 8, 2021 @ 17:51:41.000 | ytelse-trd.uninett.no | ngu-mp.ngu.no | 0.338 0.745 0.864 1.202 |

Figure B.1: Traceroute in Kibana.

| | | |
|---|---|---|
| 📅 | @date | Apr 8, 2021 @ 17:52:46.000 |
| 📅 | @timestamp | Apr 8, 2021 @ 17:52:46.000 |
| t | _id | IjPU4ngB1Sy0wiU9Xvgq |
| t | _index | uninett_traceroute-000001 |
| # | _score | - |
| t | _type | _doc |
| t | agent.ephemeral_id | b6552cae-d5d5-4c6c-b6d8-3cc1a2dd2c21 |
| t | agent.hostname | iou2 |
| t | agent.id | 59a694bd-77da-43ea-a272-2fed05930ed1 |
| ? | agent.name | ⚠ iou2 |
| t | agent.type | filebeat |
| t | agent.version | 7.10.2 |
| t | ecs.version | 1.6.0 |
| t | event | periodic |
| t | event_type | traceroute |
| t | from | ytelse-trd.uninett.no |
| t | from_addr | 158.39.1.90 |
| # | hopcount | 4 |
| t | host.name | iou2 |
| t | input.type | log |
| t | log.file.path | /var/lib/microdep/mp-uninett/log/traceroute/traceroute.json.6 |
| # | log.offset | 155,102,653 |
| t | path | trd-gw.uninett.no trd-gw2.uninett.no 193.156.2.1 ngu-mp.ngu.no |
| t | path_addr | 158.39.1.89 128.39.255.183 193.156.2.1 193.156.55.218 |
| t | path_avg_rtt | 0.3198 0.7987 0.7352 0.9127 |
| t | path_sd_rtt | 0.01714 0.05452 0.1178 0.4322 |
| ? | preferred_index | ⚠ uninett_traceroute_latest |
| 📅 | timestamp | Apr 8, 2021 @ 17:52:46.000 |
| t | to | ngu-mp.ngu.no |
| t | to_addr | 193.156.55.218 |

Figure B.2: Expanded Traceroute output in Kibana.

## B.2 Root Cause Analysis

When performing RCA, the ultimate goal is to produce a recommendation for an effective solution. The investigator must strive to accurately identify the specific underlying causes to make it easier to recommend a solution. The recommendation should address the root causes directly to ensure a targeted solution to the experienced problem. General cause classification should be avoided since it complicates the further process of elimination. The culpable root causes must be something that can be modified, i.e. not the weather, to be able to solve the problem [RH04].

RCA consists of the following steps [AF06]:

- Problem understanding - Start by defining the problem and understanding the nature of its occurrences, before generating ideas for causes.
- Problem cause data collection - Collect all the data related to the problem and possible causes.
- Problem cause data analysing - Analyse the collected data with RCA tools to determine the cause of the problem.
- Root cause identification - Identify the root causes as specific as possible so the reason the problem occurred can be addressed.
- Solution recommendation and implementation - Produce a solution recommendation to prevent recurrence and implement the solution.

To support the execution of the analysis a set of tools can be utilised [AF06]. In gaining understanding of the problem, brainstorming is practical to generate multiple ideas and performance matrices can be used to determine the importance of the suggested causes. Data can be collected through sampling or performing surveys. Different charts, such as Pareto charts, and Scatter charts, can help illustrate relationships between different causes when data is analysed. Utilising Matrix diagrams can simplify the process identifying the biggest contributing cause to the problem.

# Appendix C
# Environment Setup

In order to work on this project, one must gain permissions to the files on Uninett's data-centre. The tool used for this was Secure Shell (SSH), which is a widely used tool for remote access. Private RSA[1] or SSH keys were generated and linked to our personally made users at their proxy login-server and the main server *IOU2*. A Jupyter notebook server was installed and hosted on a forwarded port, serving as the primary development tool. This approach had quick and simple file access and supported our preferred programming language Python.

Port forwarding, also called tunnelling, was enabled when connecting to IOU2 in order to reflect remote processes and Graphical User Interface (GUI) on our local machines. This can be set by complementing the SSH command with the flag -L port:addr:port. Example setup of Jupyter hosting and connection is shown below in Listings C.1 and C.2. The -J and -A flag was used to proxy-jump through login.uninett.no to iou2.uninett.no whilst maintaining authenticated.

Listing C.1: Local terminal

```
$ eval `ssh-agent`
$ ssh-add ~/.ssh/id_rsa
$ ssh -L 9988:127.0.0.1:9988 -J <username>@login.uninett.no
    <username>@iou2.uninett.no -A
```

Listing C.2: IOU2 terminal

```
$ pipenv run jupyter notebook --no-browser --port 9988 --notebook-dir=/
```

Pipenv, used in Listing C.2, is a packet manager tool for the Python world. Creating a virtual environment reduces the risk of affecting the system when installing new packages. We used pipenv for installation of necessary python modules, see Listing C.3.

[1]RSA is a public-key cryptosystem. It is an acronym of its creators: Ron **R**ivest, Adi **S**hamir and Leonard **A**man.

Listing C.3: IOU2 terminal

```
$ py3 -m pip install pipenv
$ pipenv install statsmodels, tslearn, sklearn, pathlib, pandas,
    matplotlib, numpy, rrfc
```

# Appendix D
# Python Code

This appendix includes blocks of code written in Python that are used in this thesis' multiple implementations. They are executed in the Jupyter environment explained in Appendix C.

## D.1 File Glob

This dynamic block of python code serves as a utility block for locating paths to files based on a glob pattern. The variable *files* seen in the other blocks of code stems from this script. See [dev] for documentation. The files from multiple sources are sorted into a path of directories representing their metadata. *node_pattern* allows for dynamic selection of hostnames, *date_pattern* gather only files with specific dates, finally *file_pattern* targets filenames.

```
import pathlib
root = pathlib.Path(f"... path to root of desired files")

node_pattern = f"*" # eg. "madrid-mp"
date_pattern = "*" # eg. "2021-03-*"
file_pattern = f"*.gz" # eg. "128.35.14.26*.gz"
pattern = f"{node_pattern}/{date_pattern}/{file_pattern}"
files = sorted( list( root.glob(pattern) ) )
```

Listing D.1: File glob

## D.2 To CSV

This code is used to parse raw crude data and store as CSV files.

```
import csv
import gzip
import time
import pathlib
import datetime

start_time = time.time()
fieldnames = ['id', 'seq', 'src', 'dst', 'tx', 'rx', 'delay']
op = "path/to/output"

for file in files:
    records = {}

    with gzip.open(file, mode='rt') as f:
        for line in f:
            try:
                # list of key-value pairs # ['ID=1', 'RX=15434', ...]
                key_values = line.split()

                # parse line
                stream_id = int(key_values[0].split('=')[1])
                seq = int(key_values[1].split('=')[1])
                # SRC=ip:port
                src = key_values[2].split('=')[1].split(':')[0]
                # DST=ip:port
                dst = key_values[3].split('=')[1].split(':')[0]
                tx = float(key_values[4].split('=')[1])
                rx = float(key_values[5].split('=')[1])
                delay = rx-tx

                id = f"{src} -> {dst}"

                record = {
                    'id': stream_id,
                    'seq': seq,
                    'src': src,
                    'dst': dst,
                    'tx': tx,
                    'rx': rx,
                    'delay': delay,
                }

                if id not in records:
                    records[id] = []
                records[id].append(record)

            except:
                continue
```

```
49
50      # Write records to compressed csv
51      for id, recs in records.items():
52
53          # Create unique filename based on src and dst
54          output = pathlib.Path(f"{op}/some-filename.csv.gz")
55
56          # Make sure parent folders exists before creating file
57          output.parent.mkdir(parents=True, exist_ok=True)
58
59          try:
60              with gzip.open(output, mode='wt') as csvfile:
61                  writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
62                  writer.writeheader()
63                  writer.writerows(recs)
64          except Exception as e:
65              print(e)
```

Listing D.2: To csv

## D.3 Resample

This code is used to resample crude data. Given bin size, *mean_delay* and *packet_count* are calculated and stored as separate CSV files.

```
import time
import pathlib
import datetime
import pandas as pd

start_time = time.time()
BINSIZE = 100

for i, f in enumerate(files):

    # should create a unique filename based on current file and binsize
    output = pathlib.Path("path/to/unique_filename.csv.gz")

    # ensure that the path exists before proceeding
    output.parent.mkdir(parents=True, exist_ok=True)

    df = pd.DataFrame()

    try:
        df = pd.read_csv(f)
    except:
        continue

    df = df.set_index('seq')
    df.sort_index(inplace=True)

    # get rid of duplicate packets if exists
    df = df[~df.index.duplicated(keep='first')]

    # insert dummy rows for lost packets
    df = df.reindex( range(len(df) ), fill_value=np.NaN)
    df.sort_index(inplace=True)

    csvdf = pd.DataFrame()
    csvdf['mean_delay'] = \
        df.delay.rolling(window=BINSIZE, min_periods=0) \
        .mean()[BINSIZE::BINSIZE]

    csvdf['packet_count'] = \
        df.delay.rolling(window=BINSIZE, min_periods=0) \
        .count()[BINSIZE::BINSIZE]

    csvdf.to_csv(output, compression='gzip')
```

Listing D.3: Resample

## D.4 RRCF

Implementation of RRCF. This code is originated from paper [BMT19], github repository can be found here [BMT].

```
import rrcf

# Set tree parameters
num_trees = 10
tree_size = 256

# Create a forest of empty trees
forest = []
for _ in range(num_trees):
    tree = rrcf.RCTree()
    forest.append(tree)

# Create a dict to store anomaly score of each point
avg_codisp = {}

# For each shingle...
for index, point in enumerate(data):
    # For each tree in the forest...
    for tree in forest:
        # If tree is above permitted size, drop the oldest point (FIFO)
        if len(tree.leaves) > tree_size:
            tree.forget_point(index - tree_size)

        # Insert the new point into the tree
        tree.insert_point(point, index=index)

        # Compute codisp on the new point and take the average among
    all trees
        if not index in avg_codisp:
            avg_codisp[index] = 0
        avg_codisp[index] += tree.codisp(index) / num_trees
```

Listing D.4: RRCF

## D.5 Traceroute

Analysing traceroutes to find average and maximum delay to each node.

```
ROWS = None
break0 = False

node_delay = {}
max_delay = {}

def stop(i, limit):
    global break0
    if not limit:
        return False
    if i >= limit:
        break0 = True # stop outer loop
        return True
    return False

def parse(line):
    invalids = ['traceroute', 'starttime']
    if any(item in line for item in invalids):
        return []

    return line.split()

for n, file in enumerate(files):
    file_ip = file.name[11:-3]

    if break0:
        break

    with gzip.open(file, mode='rt') as f:
        oldc = 0
        for i, file_line in enumerate(f):
            if ROWS and i > ROWS:
                break

            line = parse(file_line)
            if not line:
                continue

            ### line is cleansed ###
            if line[1] in shared_nodes:
                if line[1] not in node_delay.keys():
                    node_delay[line[1]] = [0,0]
                    max_delay[line[1]] = 0.0
                    print(max_delay)
                for item in line:
                    if line.index(item)+1 > len(line)-1:
                        break
                    if line[line.index(item)+1] == 'ms':
```

```
                        node_delay[line[1]][0]+=float(item)
                        node_delay[line[1]][1]+=1
                        if float(item) > float(max_delay[line[1]]):
                            max_delay[line[1]] = item

print('The paths share the following nodes: ')
print('')
for node in node_delay:
    node_delay[node] = node_delay[node][0] / node_delay[node][1]
    print('Node: ' + str(node) + ' - Average delay: ' +
    str(round(node_delay[node], 3)) + ' ms ' + '- Max delay: ' +
    str(max_delay[node]))
```

Listing D.5: Code for traceroutes

# Appendix E
# Gap Analysis

This appendix includes all comparisons between gap record fields not included in the gap analysis in Chapter 7.

## E.1 h_ddelay

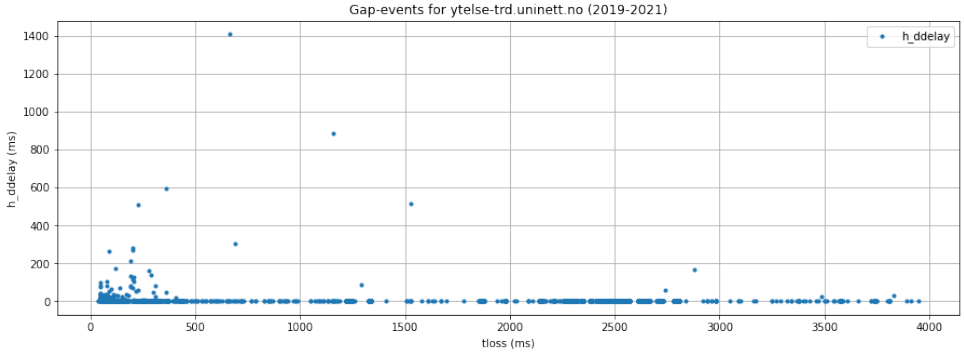



Figure E.1: Gaps tloss/h_ddelay.
h_ddelay show no correlation with tloss.

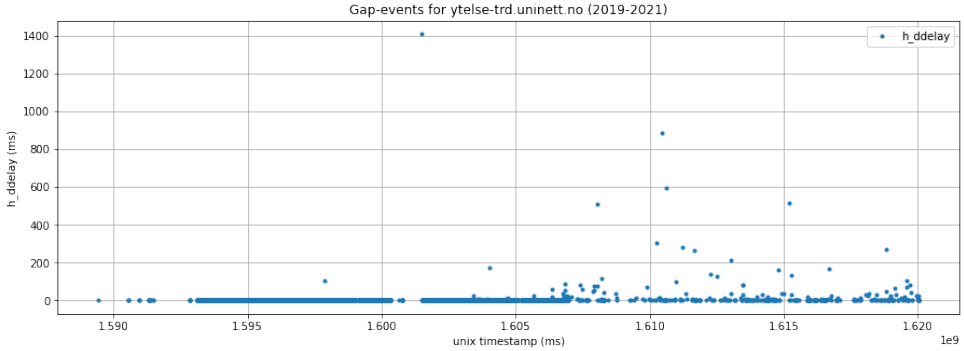



Figure E.2: Gaps timestamp/h_ddelay.

## E.2 h_delay

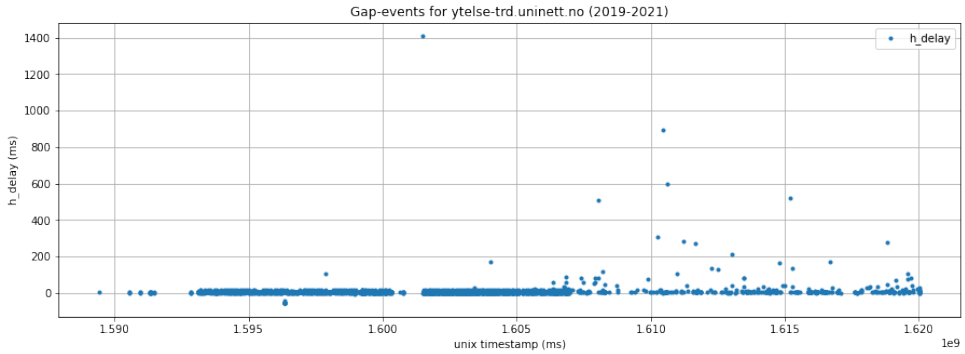



Figure E.3: Gaps timestamp/h_delay.

## E.3 h_min_d

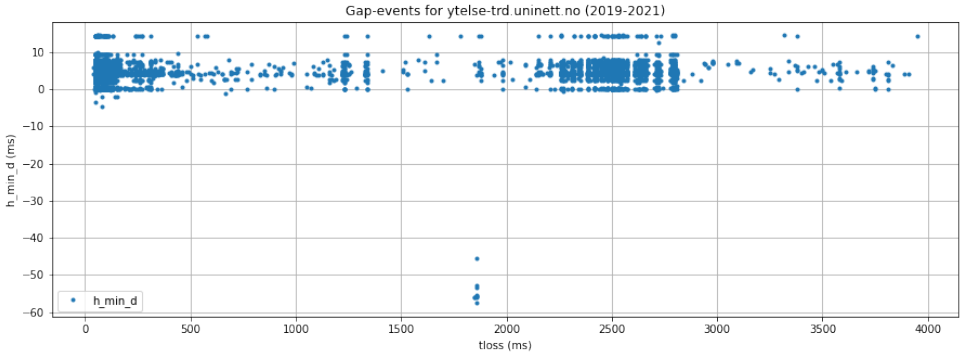



Figure E.4: Gaps tloss/h_min_d.

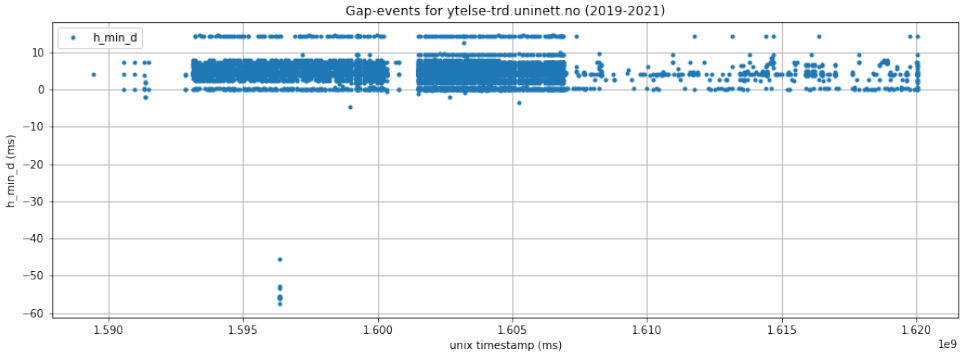



Figure E.5: Gaps timestamp/h_min_d.

## E.4 h_jit

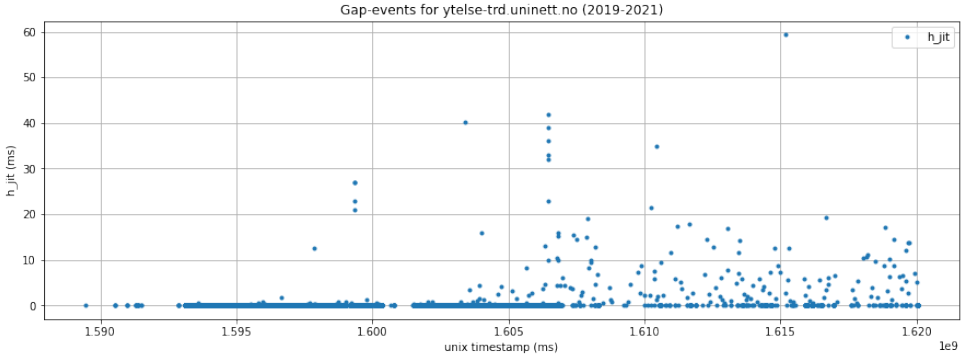



Figure E.6: Gaps timestamp/h_jit.

## E.5 h_slope_d

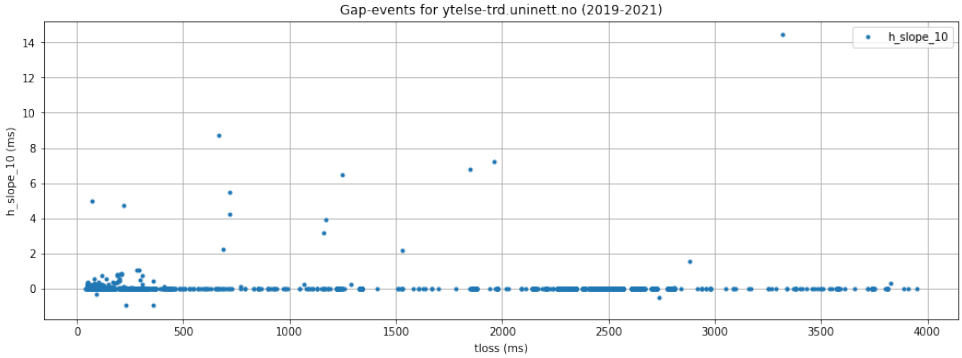



Figure E.7: Gaps tloss/h_slope_10.

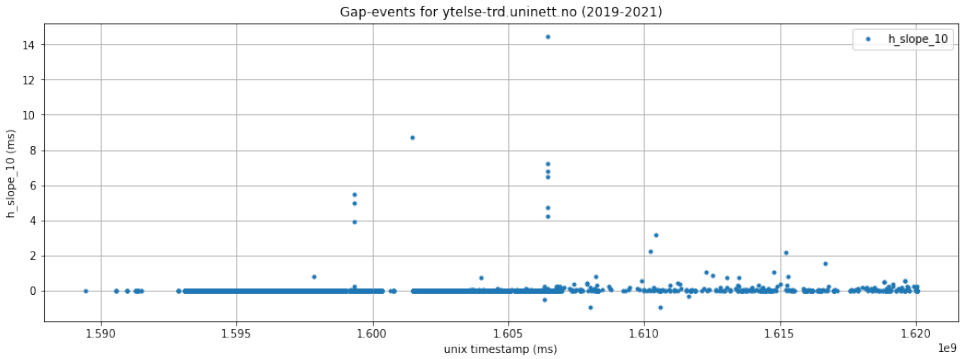



Figure E.8: Gaps timestamp/h_slope_10.

## E.6 h_slope_50

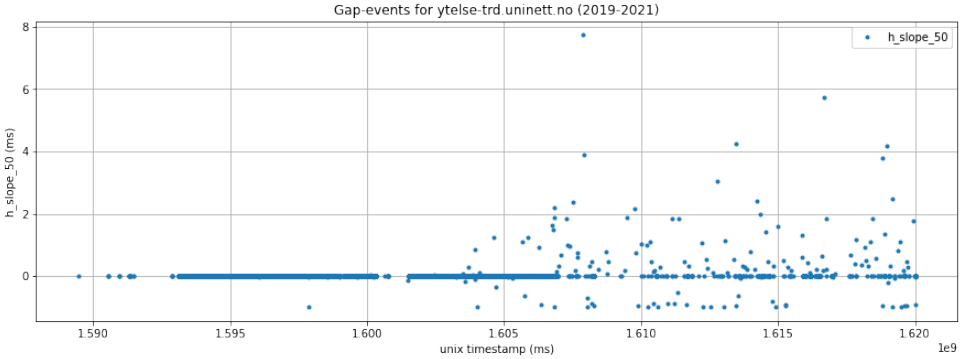



Figure E.9: Gaps timestamp/h_slope_50.

## E.7 overlap

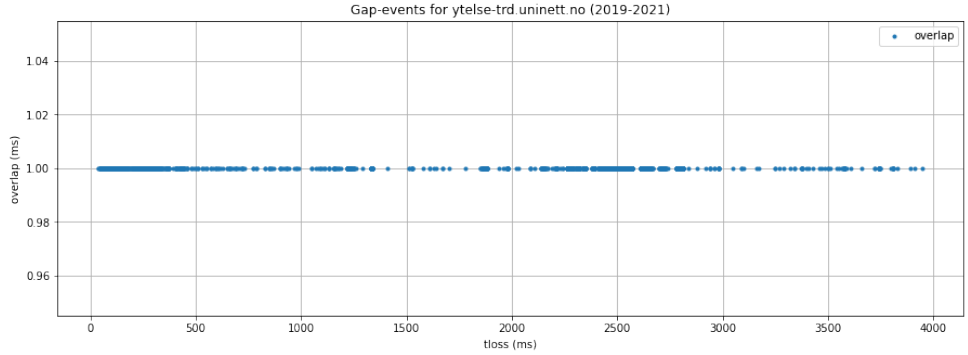



Figure E.10: Gaps tloss/overlap.

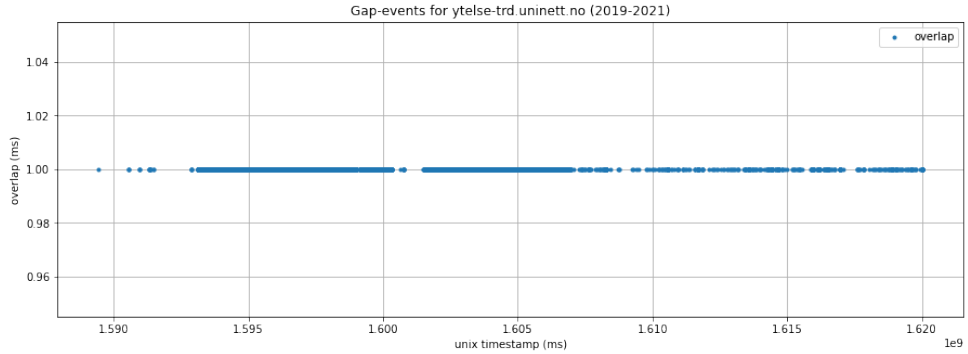



Figure E.11: Gaps timestamp/overlap.

## E.8 dTTL

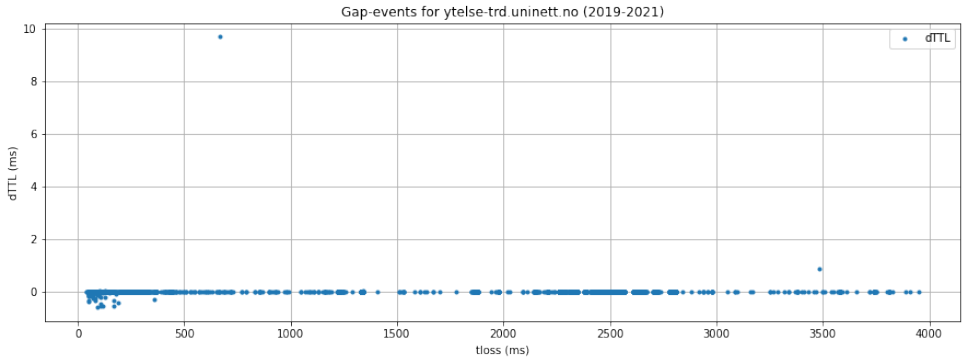



Figure E.12: Gaps tloss/dTTL.

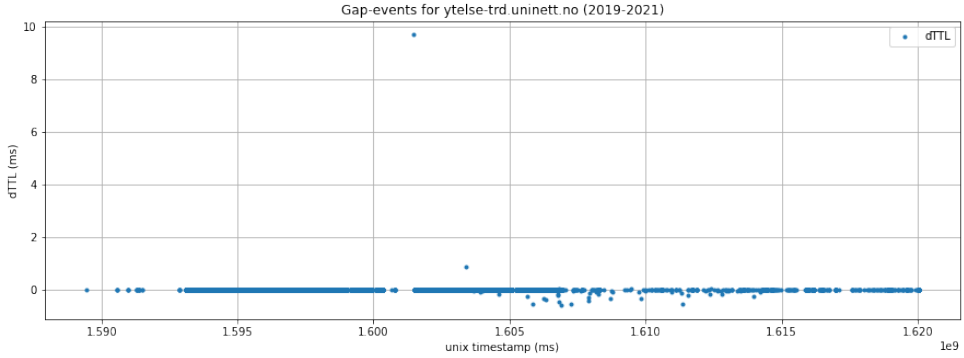



Figure E.13: Gaps timestamp/dTTL.

## E.9 t_ddelay

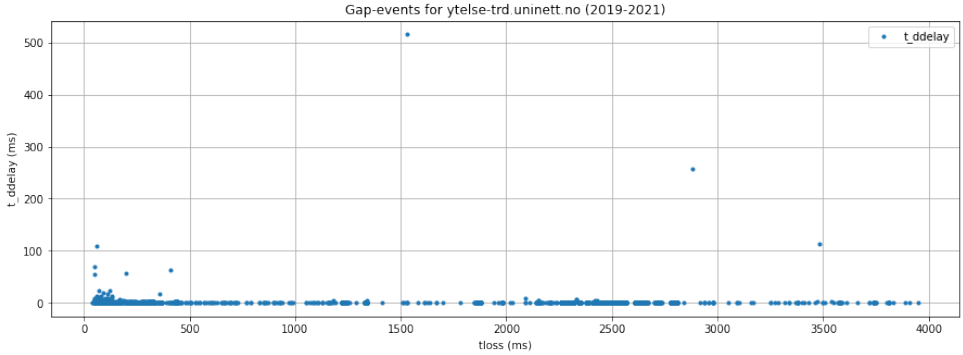



Figure E.14: Gaps tloss/t_ddelay.

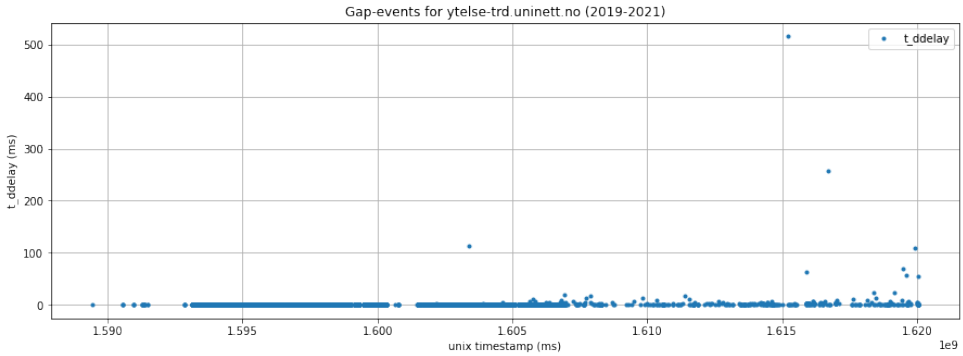



Figure E.15: Gaps timestamp/h_ddelay.

## E.10 t_delay

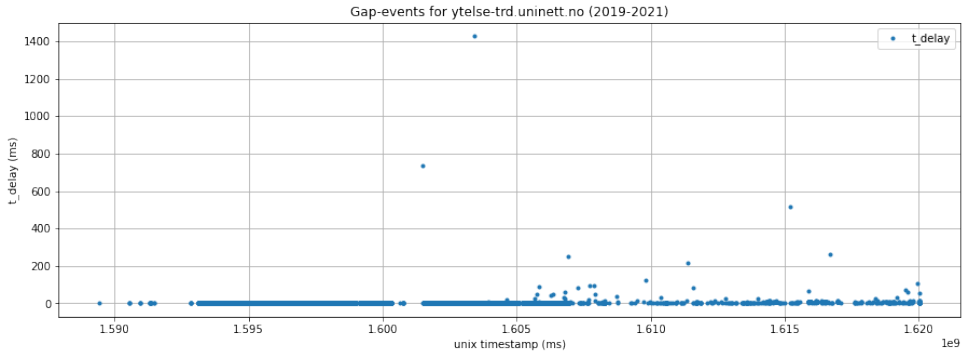



Figure E.16: Gaps timestamp/t_delay.

## E.11 t_min_d

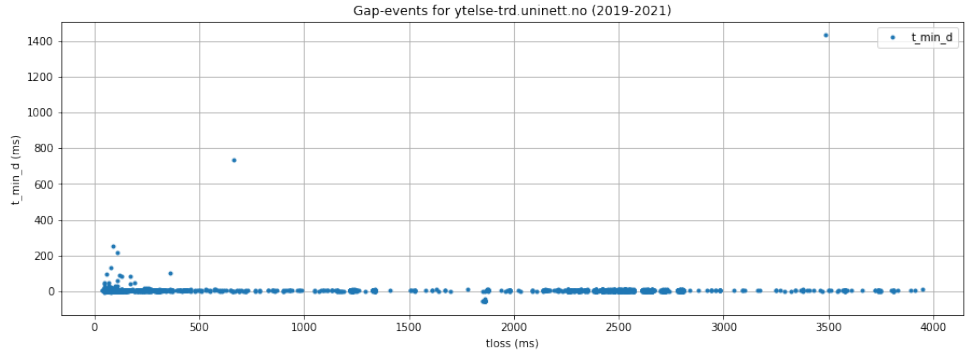



Figure E.17: Gaps tloss/t_min_d.

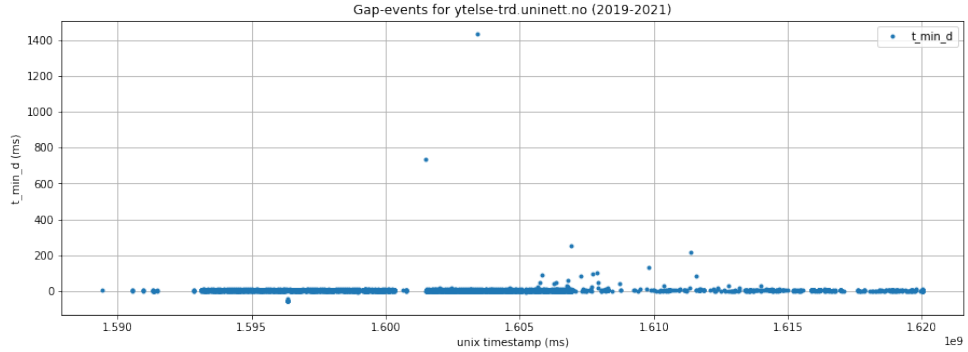



Figure E.18: Gaps timestamp/t_min_d.

## E.12 t_slope_50

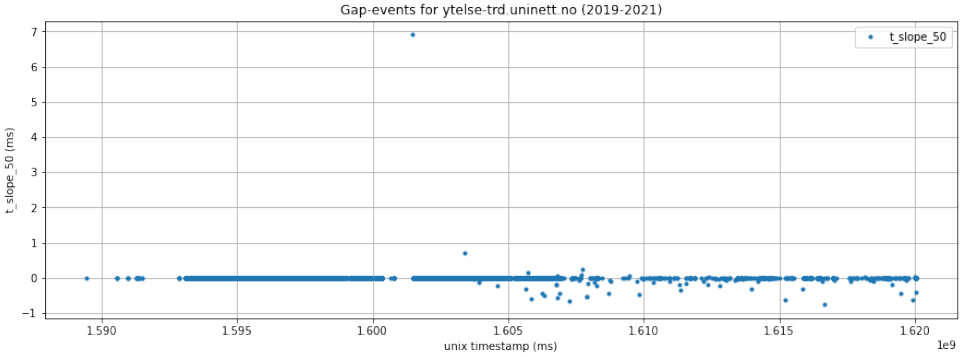



Figure E.19: Gaps timestamp/t_slope_50.

# Appendix F
# Correlation Analysis

In this appendix more interesting examples, found during cross stream correlation analysis in Chapter 8, are included for interested readers. Same as results in Section 8.2, this appendix displays correlation matrices with the underlying addends and sums used for calculations.

## F.1 Runar

Runar's figures show nearly 100% correlation of all streams. These results serve as a sanity check for our simple method of capturing simultaneous deviations from normal behaviour. Because this is an employee's personal machine, we know beforehand that all streams passes through the same infrastructure. Therefore all streams are likely to have equal behaviour and a high correlation.

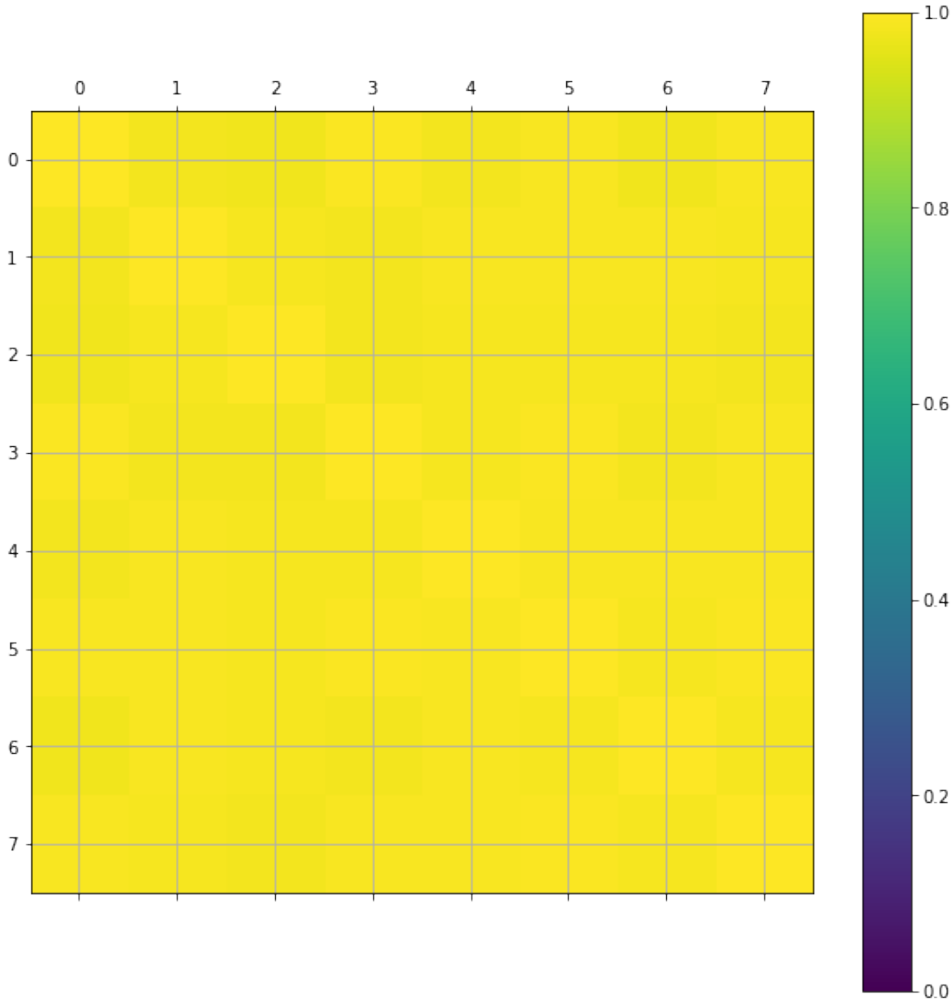



Figure F.1: Plot of correlation-scores (Runar, 2021.03.10, BINSIZE=1000).

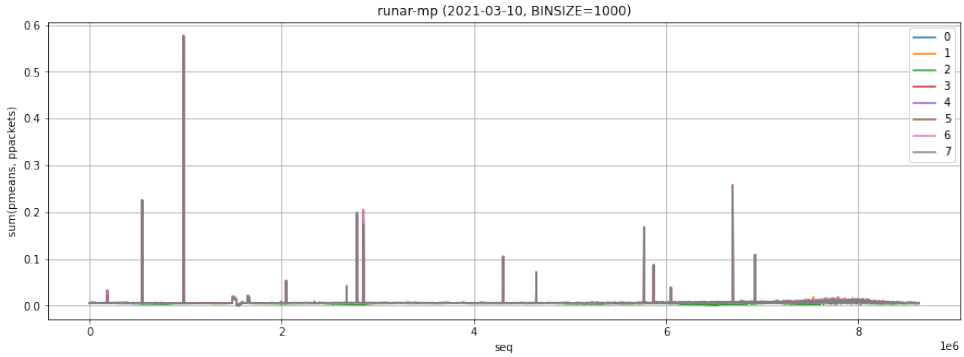



Figure F.2: Plot of sum(pmeans, ppackets) (Runar, 2021.03.10, BINSIZE=1000.)

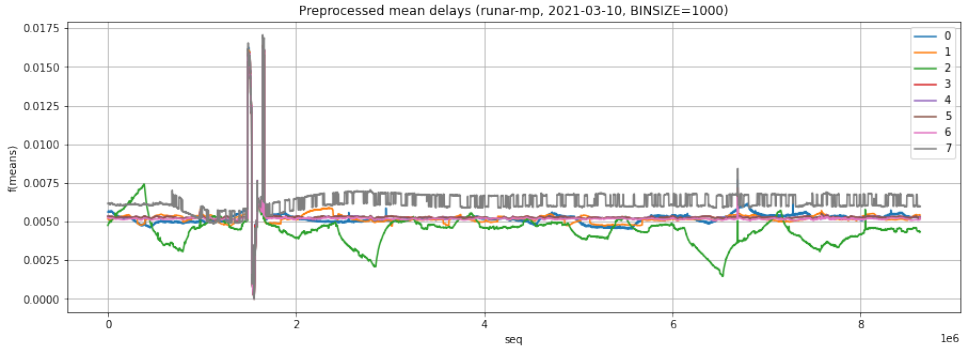



Figure F.3: Plot of packet mean-delays (Runar, 2021.03.10, BINSIZE=1000).

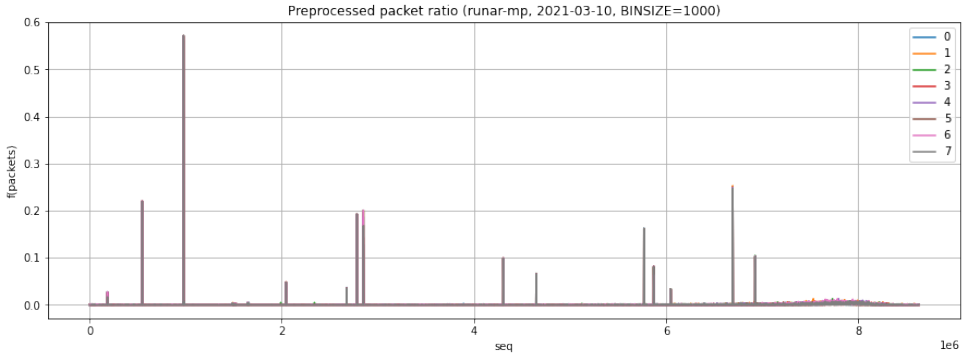



Figure F.4: Plot of packet-loss (Runar, 2021.03.10, BINSIZE=1000).

## F.2 Madrid

Correlation matrix for Madrid shows numerous correlated streams. We see several pairs of high correlation. The aggregated results show a fairly steady day with few disruptions. There is a common spike at the very beginning of the day that likely causes the correlation.

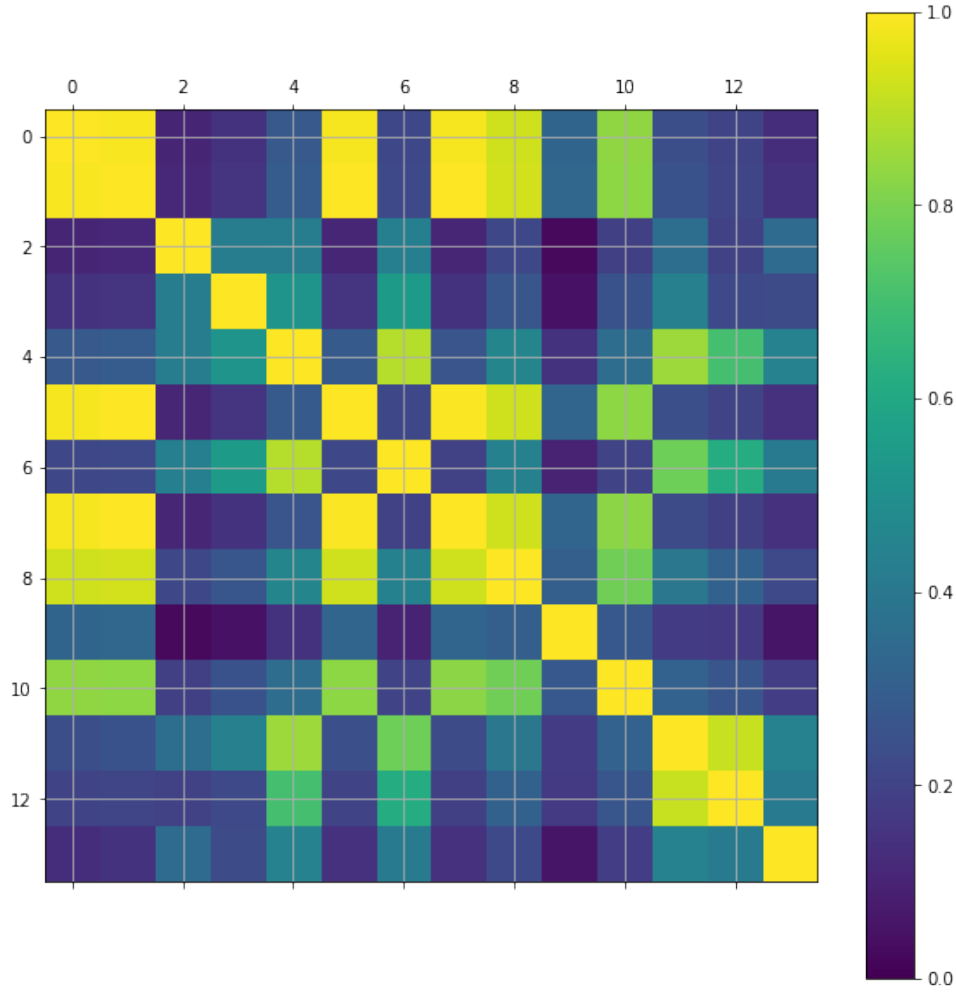



Figure F.5: Plot of correlation-scores (Madrid, 2021.03.10, BINSIZE=1000).

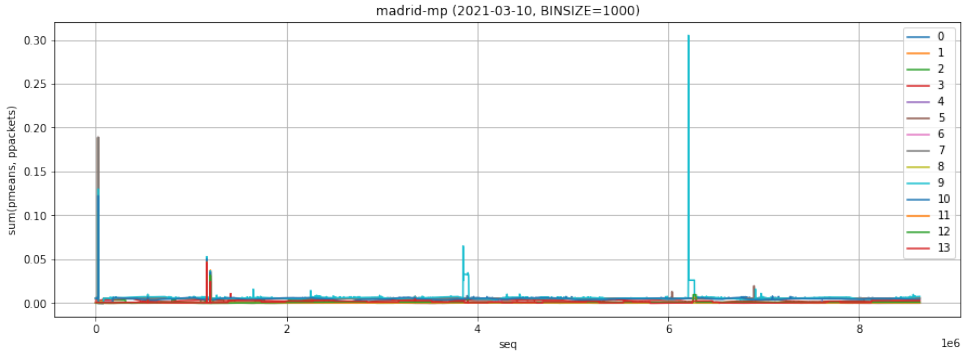



Figure F.6: Plot of sum(pmeans, ppackets) (Madrid, 2021.03.10, BINSIZE=1000).

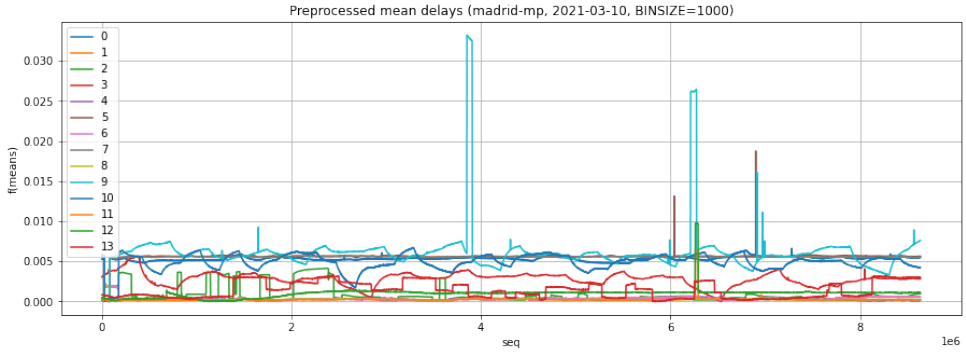



Figure F.7: Plot of packet mean-delays (Madrid, 2021.03.10, BINSIZE=1000).

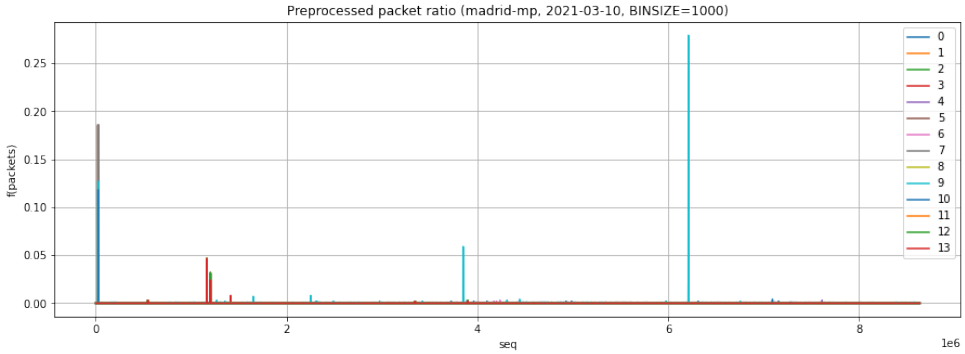



Figure F.8: Plot of packet-loss (Madrid, 2021.03.10, BINSIZE=1000).

## F.3 São Paulo

São Paulo has only a single pair (4-7) of strong correlation. The aggregated values from Figure F.10 does not show many common spikes either.

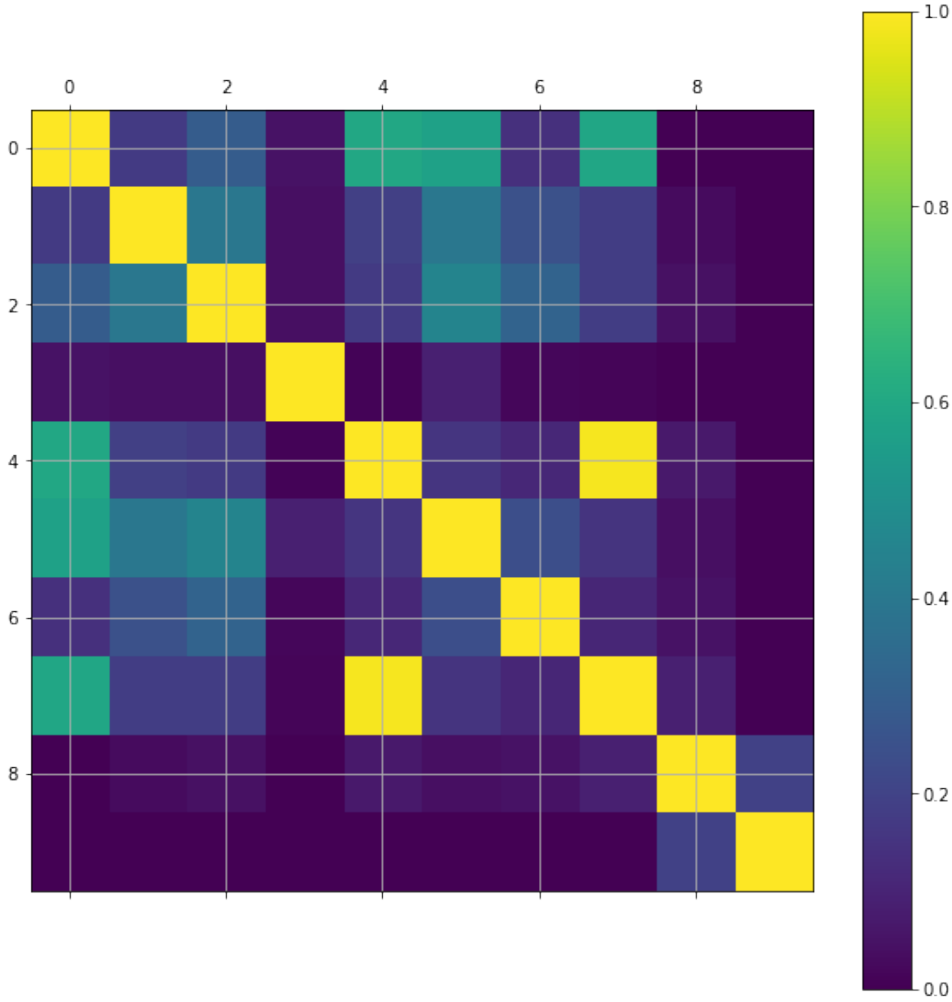



Figure F.9: Plot of correlation-scores (São Paulo, 2021.03.10, BINSIZE=1000).

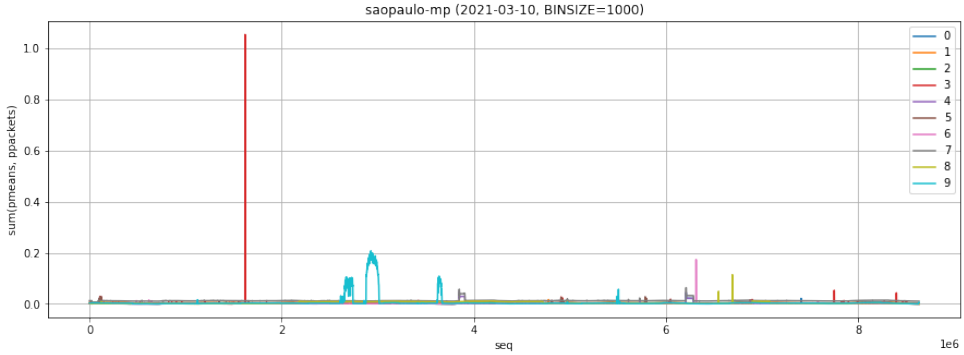



Figure F.10: Plot of sum(pmeans, ppackets) (São Paulo, 2021.03.10, BINSIZE=1000).

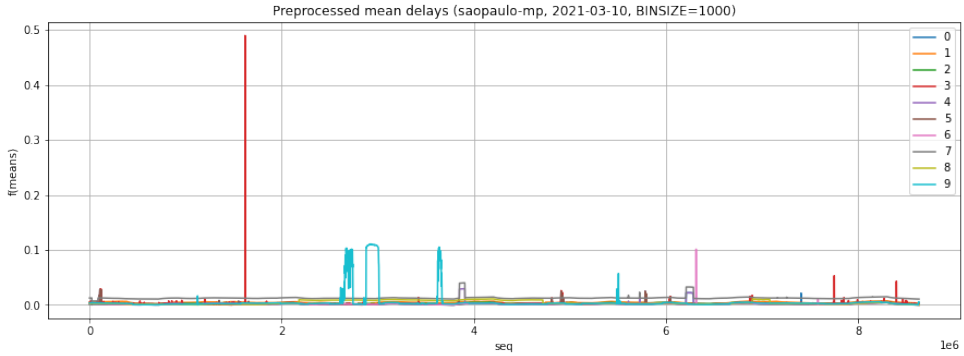



Figure F.11: Plot of packet mean-delays (São Paulo, 2021.03.10, BINSIZE=1000).

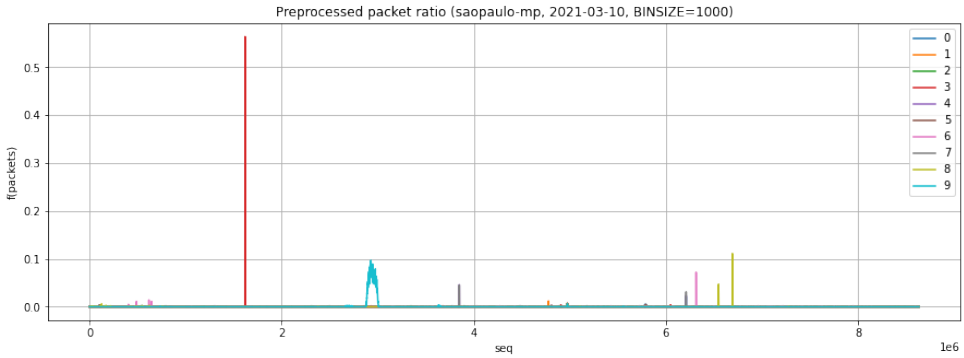



Figure F.12: Plot of packet-loss (São Paulo, 2021.03.10, BINSIZE=1000).

# Appendix G
# Streams

This appendix consists of the exact IP addresses used for cross stream correlation analysis in Chapter 8. The streams are presented as dictionaries where the numeric keys represent the numbers on the matrix-figures, and the string values are ”SRC—DST” addresses.

## G.1 Streams used for BINSIZE comparisons

See Figures 8.2, 8.3, 8.4

```
{
    0: '128.39.65.26---138.44.131.98',
    1: '13.82.53.167---138.44.131.98',
    2: '130.216.51.132---138.44.131.98',
    3: '192.148.201.15---138.44.131.98',
    4: '195.113.144.236---138.44.131.98',
    5: '200.133.192.133---138.44.131.98',
    6: '54.202.174.174---138.44.131.98',
    7: '128.39.65.26---18.195.175.203',
    8: '13.79.144.22---18.195.175.203',
    9: '192.148.201.15---18.195.175.203',
    10: '200.133.192.133---18.195.175.203',
    11: '34.246.185.57---18.195.175.203',
    12: '35.246.8.199---18.195.175.203',
    13: '128.39.65.26---34.246.185.57',
    14: '13.53.187.135---34.246.185.57',
    15: '13.79.144.22---34.246.185.57',
    16: '18.195.175.203---34.246.185.57',
    17: '192.148.201.15---34.246.185.57',
    18: '194.68.13.71---34.246.185.57',
    19: '35.228.220.215---34.246.185.57',
    20: '35.246.8.199---34.246.185.57',
    21: '128.39.65.26---13.53.187.135',
```

```
    22: '13.79.144.22---13.53.187.135',
    23: '158.39.1.126---13.53.187.135',
    24: '158.39.1.90---13.53.187.135',
    25: '158.39.1.94---13.53.187.135',
    26: '158.39.1.98---13.53.187.135',
    27: '185.71.209.4---13.53.187.135',
    28: '192.148.201.15---13.53.187.135',
    29: '194.68.13.71---13.53.187.135',
    30: '34.246.185.57---13.53.187.135',
    31: '35.246.8.199---13.53.187.135',
    32: '128.39.65.26---54.202.174.174',
    33: '13.82.53.167---54.202.174.174',
    34: '130.216.51.132---54.202.174.174',
    35: '138.44.131.98---54.202.174.174',
    36: '18.195.175.203---54.202.174.174',
    37: '192.148.201.15---54.202.174.174',
    38: '194.68.13.71---54.202.174.174',
    39: '200.133.192.133---54.202.174.174',
    40: '35.246.8.199---54.202.174.174',
    41: '128.39.65.26---130.216.51.132',
    42: '13.79.144.22---130.216.51.132',
    43: '13.82.53.167---130.216.51.132',
    44: '130.59.35.214---130.216.51.132',
    45: '138.44.131.98---130.216.51.132',
    46: '200.133.192.133---130.216.51.132',
    47: '35.246.8.199---130.216.51.132',
    48: '54.202.174.174---130.216.51.132',
    49: '128.39.65.26---13.82.53.167',
    50: '13.79.144.22---13.82.53.167',
    51: '130.216.51.132---13.82.53.167',
    52: '138.44.131.98---13.82.53.167',
    53: '192.148.201.15---13.82.53.167',
    54: '194.68.13.71---13.82.53.167',
    55: '200.133.192.133---13.82.53.167',
    56: '35.246.8.199---13.82.53.167',
    57: '54.202.174.174---13.82.53.167',
    58: '128.39.65.26---13.79.144.22',
    59: '13.53.187.135---13.79.144.22',
    60: '13.82.53.167---13.79.144.22',
    61: '130.216.51.132---13.79.144.22',
    62: '158.39.1.126---13.79.144.22',
    63: '158.39.1.90---13.79.144.22',
    64: '158.39.1.94---13.79.144.22',
    65: '158.39.1.98---13.79.144.22',
    66: '18.195.175.203---13.79.144.22',
```

```
    67: '185.71.209.4---13.79.144.22',
    68: '192.148.201.15---13.79.144.22',
    69: '194.68.13.71---13.79.144.22',
    70: '200.133.192.133---13.79.144.22',
    71: '34.246.185.57---13.79.144.22',
    72: '35.228.220.215---13.79.144.22',
    73: '35.246.8.199---13.79.144.22',
    74: '46.249.255.10---13.79.144.22',
    75: '128.39.65.26---109.105.116.52',
    76: '130.59.35.214---109.105.116.52',
    77: '158.39.1.126---109.105.116.52',
    78: '158.39.1.90---109.105.116.52',
    79: '158.39.1.94---109.105.116.52',
    80: '158.39.1.98---109.105.116.52',
    81: '185.71.209.4---109.105.116.52',
    82: '192.148.201.15---109.105.116.52',
    83: '194.68.13.71---109.105.116.52',
    84: '195.113.144.236---109.105.116.52',
    85: '46.249.255.10---109.105.116.52',
    86: '89.45.232.192---109.105.116.52',
    87: '128.39.65.26---35.246.8.199',
    88: '13.53.187.135---35.246.8.199',
    89: '13.79.144.22---35.246.8.199',
    90: '13.82.53.167---35.246.8.199',
    91: '130.216.51.132---35.246.8.199',
    92: '138.44.131.98---35.246.8.199',
    93: '158.39.1.126---35.246.8.199',
    94: '158.39.1.90---35.246.8.199',
    95: '158.39.1.94---35.246.8.199',
    96: '158.39.1.98---35.246.8.199',
    97: '18.195.175.203---35.246.8.199',
    98: '185.71.209.4---35.246.8.199',
    99: '192.148.201.15---35.246.8.199',
   100: '194.68.13.71---35.246.8.199',
   101: '200.133.192.133---35.246.8.199',
   102: '34.246.185.57---35.246.8.199',
   103: '35.228.220.215---35.246.8.199',
   104: '54.202.174.174---35.246.8.199',
   105: '128.39.65.26---35.228.220.215',
   106: '13.79.144.22---35.228.220.215',
   107: '192.148.201.15---35.228.220.215',
   108: '194.68.13.71---35.228.220.215',
   109: '34.246.185.57---35.228.220.215',
   110: '35.246.8.199---35.228.220.215',
   111: '109.105.116.52---192.148.201.15',
```

```
    112: '128.39.65.26---192.148.201.15',
    113: '13.53.187.135---192.148.201.15',
    114: '13.79.144.22---192.148.201.15',
    115: '13.82.53.167---192.148.201.15',
    116: '138.44.131.98---192.148.201.15',
    117: '18.195.175.203---192.148.201.15',
    118: '194.68.13.71---192.148.201.15',
    119: '195.113.144.236---192.148.201.15',
    120: '200.133.192.133---192.148.201.15',
    121: '34.246.185.57---192.148.201.15',
    122: '35.228.220.215---192.148.201.15',
    123: '35.246.8.199---192.148.201.15',
    124: '54.202.174.174---192.148.201.15',
    125: '109.105.116.52---195.113.144.236',
    126: '128.39.65.26---195.113.144.236',
    127: '138.44.131.98---195.113.144.236',
    128: '192.148.201.15---195.113.144.236',
    129: '194.68.13.71---195.113.144.236',
    130: '109.105.116.52---185.71.209.4',
    131: '13.53.187.135---185.71.209.4',
    132: '13.79.144.22---185.71.209.4',
    133: '158.39.1.90---185.71.209.4',
    134: '158.39.1.94---185.71.209.4',
    135: '158.39.1.98---185.71.209.4',
    136: '194.68.13.71---185.71.209.4',
    137: '35.246.8.199---185.71.209.4',
    138: '128.39.65.26---200.133.192.133',
    139: '13.79.144.22---200.133.192.133',
    140: '13.82.53.167---200.133.192.133',
    141: '130.216.51.132---200.133.192.133',
    142: '130.59.35.214---200.133.192.133',
    143: '138.44.131.98---200.133.192.133',
    144: '18.195.175.203---200.133.192.133',
    145: '192.148.201.15---200.133.192.133',
    146: '35.246.8.199---200.133.192.133',
    147: '54.202.174.174---200.133.192.133'
}
```

## G.2 Streams used for Madrid

See Figures [F.5, F.6, F.7, F.8]

```
{
    0: '128.39.65.26---54.202.174.174',
    1: '13.82.53.167---54.202.174.174',
    2: '130.216.51.132---54.202.174.174',
    3: '138.44.131.98---54.202.174.174',
    4: '18.195.175.203---54.202.174.174',
    5: '192.148.201.15---54.202.174.174',
    6: '194.68.13.71---54.202.174.174',
    7: '200.133.192.133---54.202.174.174',
    8: '35.246.8.199---54.202.174.174'
}
```

## G.3 Streams used for Amazon

See Figures [8.6, 8.7, 8.8, 8.9]

```
{
    0: '128.39.65.26---54.202.174.174',
    1: '13.82.53.167---54.202.174.174',
    2: '130.216.51.132---54.202.174.174',
    3: '138.44.131.98---54.202.174.174',
    4: '18.195.175.203---54.202.174.174',
    5: '192.148.201.15---54.202.174.174',
    6: '194.68.13.71---54.202.174.174',
    7: '200.133.192.133---54.202.174.174',
    8: '35.246.8.199---54.202.174.174'
}
```

## G.4 Streams used for Runar

See Figures [F.1, F.2, F.3, F.4]

```
{
    0: '109.105.116.52---185.71.209.4',
    1: '13.53.187.135---185.71.209.4',
    2: '13.79.144.22---185.71.209.4',
    3: '158.39.1.90---185.71.209.4',
    4: '158.39.1.94---185.71.209.4',
    5: '158.39.1.98---185.71.209.4',
    6: '194.68.13.71---185.71.209.4',
    7: '35.246.8.199---185.71.209.4'
}
```

Emil Telstad and Mats Ove Sannes

# Machine Learning for Multi-Source Analysis

NTNU
Norwegian University of
Science and Technology