UPPSALA
UNIVERSITET

UPTEC IT 23040
Examensarbete 30 hp
Oktober 2023

Machine Learning assisted gNodeB Data Link Layer Capacity Management

Adam Axelsson

Civilingenjörsprogrammet i informationsteknologi

UPPSALA
UNIVERSITET

Machine Learning assisted gNodeB Data Link Layer Capacity Management

Adam Axelsson

Abstract

In the uplink direction of 5G New Radio, signals are sent between Radio Units and Digital Units. The production of these signals is non-deterministic, leading to signals often being produced in bursts. These signal bursts can lead to exceeding the Data Link Layer capacity, which causes packet losses. It is possible to control the burstiness by delaying signals over time. However, excessive delays should be avoided since the processing of signals must be completed within strict time constraints. In this paper, two machine-learning-based algorithms with the objective of avoiding packet losses by introducing delays to signals were proposed. One algorithm was based on the symbol number of the signals, and the other one used a queue-based approach. Only the symbol-based algorithm was thoroughly evaluated. Visualizations of test data, as well as lab tests, showed that the symbol-based algorithm was able to efficiently delay signals in order to reduce the maximum load on the Data Link Layer.

Teknisk-naturvetenskapliga fakulteten

Uppsala universitet, Utgivningsort Uppsala

Handledare: Zaheer Ahmed Ämnesgranskare: Jens Sjölund

Examinator: Lars-Åke Nordén

Sammanfattning

I upplänksriktningen av 5G New Radio skickas signaler mellan radioenheter och digitala enheter. Produktionen av dessa signaler är icke-deterministisk, vilket leder till att signaler ofta produceras i skurar. Dessa signalskurar kan leda till att kapaciteten i datalänklagret överskrids, vilket orsakar paketförluster. Det är möjligt att kontrollera dessa skurar genom att fördröja signalerna över tid. Överdriven fördröjning bör dock undvikas eftersom bearbetningen av signalerna måste slutföras inom strikta tidsbegränsningar. I denna artikel föreslogs två maskininlärningsbaserade algoritmer med målet att undvika paketförluster genom att införa fördröjningar i signalerna. En av algoritmerna använde sig av signalernas symbolnummer och den andra använde en köbaserad metod. Endast algoritmen som använde sig av signalernas symbolnummer utvärderades ingående. Visualiseringar av testdata, samt labtester, visade att algoritmen som använde symbolnummer effektivt kunde fördröja signaler för att minska den maximala belastningen på datalänklagret.

Contents

1 Introduction

Telecommunication and machine learning are two fields that are experiencing incredible progress at the moment. 5G is currently being rolled out commercially, and research has already begun in creating the sixth generation of cellular networks. The sixth generation (6G) will achieve extremely high peak data rates[1] and researchers believe that this generation will heavily rely on machine learning and artificial intelligence for optimization and performance[2] (among many other use cases). 6G will likely have to be very dynamic and able to intelligently distribute resources in order to maximize performance as the network dynamics constantly change. ML algorithms could be trained on historical network data and find patterns and insights to create good policies for these kinds of issues[3]. The work of integrating machine learning with telecommunication has already begun in 5G. According to a study[4] conducted by Ericsson in 2019, 53% of service providers expected AI to have been incorporated into their networks already by late 2020. The study also asked the service providers in which areas they will focus on implementing AI and machine learning. Out of 11 categories, the most popular answer (64%) was ”Network capacity planning/management”. This report takes a step in the direction of integrating telecommunications systems with machine learning within the mentioned category.

The scope of this thesis was to explore if an ML model could optimize the performance of a data link in Ericsson's 5G system, based on network dynamics. This was performed by introducing varying delays to signals traversing the link to avoid congestion and packet losses that occur when the link capacity is exceeded. Another important objective was to avoid the introduction of excessive delays as this would lead to poor performance. The questions that this work aims to answer are:

- Can a machine learning model use network dynamics to predict the traffic volume?
- Can a pacing algorithm use the predictions of the machine learning model to spread signals in time so that both packet losses and unnecessary delays are avoided?

The perfect pacing algorithm can be described as one that is able to completely avoid packet losses, with the smallest possible delays. This can be seen as the long-term goal of this project. However, this project did not have the time or the resources to achieve such precise results. The goal of this paper was to explore algorithms that take a step in the direction of the final goal. The minimum criteria that these algorithms should fulfil can be described with the following bullet points:

- Significantly reduce the peak memory consumption of the switch.
- Avoid excessive delays. The minimum requirement is that packets from different slots should not overlap, which can cause new unexpected peaks.
- The algorithm should function for all possible network scenarios, both for high and low traffic.

A limitation of this thesis was that only simulated data was used. Simulated data was utilized instead of real data as it was more easily acquired and this project was under a time limit. Furthermore, the produced data was visualized to help acquire a deeper understanding of how and why packet losses occurred. These visualizations were then used to create algorithms that can delay signals in order to spread them in ways that achieve high data link utilization without exceeding the capacity and causing packet losses. Two potential signal delay strategies were created. One strategy was a queue-based solution and one other solution utilized ML to dynamically change the delays depending on several network parameters. Only the latter-mentioned algorithm was thoroughly evaluated.

The rest of this report will be structured in the following way. Section 2 presents the background of the problem and how it has been previously handled. Section 3 defines some recurring concepts that must be understood before reading the rest of the report. Section 4 presents some related literature. Section 5 contains the methods that were used in the work. In Section 6 the results are presented. Some discussions are made in Section 7, and Section 8 contains the reached conclusions from this work. Finally, in Section 9, possible future work is presented.

2 Concepts & Abbreviations

This work contains several telecommunications-specific concepts that might not be understood by all who read this report. Therefore, to improve the readability of the text, some of the concepts and abbreviations that will be used repeatedly will be defined.

2.1 Abbreviations

- **AI** Artificial Intelligence
- **ASIC** Application-Specific Integrated Circuit

- **DU** Digital Unit
- **EMCA** Ericsson Many-Core Architecture
- **gNodeB** Next Generation Node B
- **ML** Machine Learning
- **NR** New Radio
- **PUSCH** Physical Uplink Shared Channel
- **RL** Reinforcement Learning
- **RTT** Round Trip Time
- **RU** Radio Unit
- **TCP** Transmission Control Protocol
- **UE** User Equipment
- **3GPP** 3rd Generation Partnership Project

2.2 Concepts

- **PUSCH Channel**
  There exist multiple different channels that are used to allow flexible communications between the gNodeB and UEs. In this work, the focus is mainly on the PUSCH channel. PUSCH is the channel in which the UEs send data to the gNodeB (Uplink direction)[5].

- **Ericsson Signal Types.** In the uplink direction, Ericsson uses four different signal types. MeasInd, IpnInd, BetaInd and DataInd. The first three of these signal types are different kinds of measurement data that the gNodeB base station has done on the radio signal. The last one, DataInd, is the most important one for this work. DataInd holds the actual user data that is sent by the UEs. This signal type makes up at least 80% of the total transmitted data. Because of this, the existing pacing algorithm only applies delays to signals of this type.

- **Slots and Symbols.**
  The radio transmission in NR is divided into slots and symbols in the time domain[5]. A slot has a duration of $500\mu s$ (using the configurations relevant to this work) and each slot is made up of 14 symbols. This framework is a way of organizing resources and controlling data transmission.

- **EMCA.** The EMCA is Ericsson's Digital Signal Processing ASIC hardware. It consists of many Digital Signal Processing Cores and enables multiple tasks to be run simultaneously on shared hardware resources. It is designed to be able to be used on all networks, regardless of their capacity and size.

3 Background

In 5G NR, signals can travel in two directions (Figure 1). These two directions are called uplink and downlink. When signals are sent from a phone to a base station this is called uplink direction. Similarly, downlink signals are sent from the base station back to the phone. This base station is called gNodeB , and the purpose of this node can be described in short as being responsible for connecting mobile devices to the 5G network. In this report, the focus will be on the uplink direction. When a signal is sent in this direction a RU will convert this analogue signal to a digital one before it is sent to a DU. The DU will then further process the signal, for example performing demodulation on the signal. The production of these signals in the RU is non-deterministic and burst in nature. These unpredictable signal bursts can result in exceeding the data link layer capacity. When the capacity is exceeded, the data link will not be able to handle all sent signals. This will force the UEs to send the messages again to the RU when they do not receive an acknowledgement. This problem is known as packet losses[6]. Sending packets multiple times is very inefficient. The dropped packets will have increased transmission time due to having to travel between UE and RU a second time, and will also waste resources that could have been used by other packets. Initially, there was no method for dealing with this issue, meaning that signals were sent as they were produced. This was problematic as it led to a large risk of exceeding the link layer capacity. At the moment, this problem is handled with a pacing algorithm that spreads the frequent DataInd signals evenly over the entire transmission window (see Figure 2b). This solution is not optimal as it is completely static and not able to adapt the signal spreading in real-time, leading to the system not maximizing utilization of the capacity.

In this project, we will investigate if a machine-learning model would be able to spread signals in a smarter and more effective manner. The goal is to create a more dynamic algorithm that is able to avoid packet losses while also maximizing the utilization of the data link layer capacity.

Figure 3 displays an overview of the components relevant to this project. The packets that are sent to the DU are sent from multiple individual sources. A switch in the DU is then responsible for delegating the packets to EMCAs where the packets will be processed. It is possible to introduce delays for each packet on the sender side. There

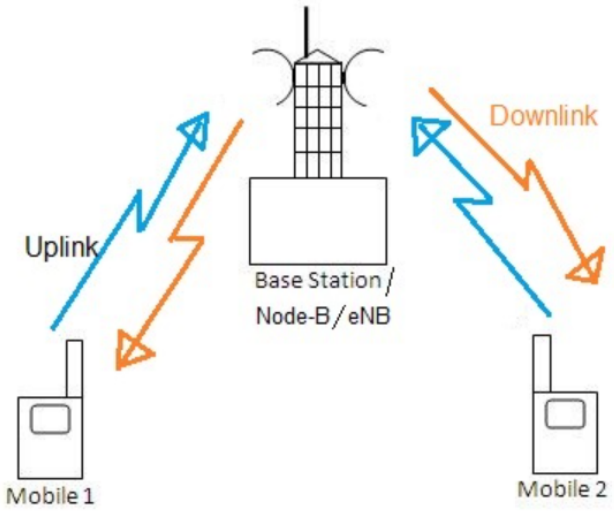
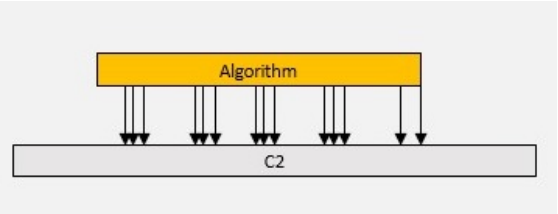
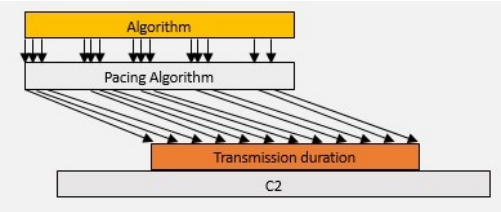


Figure 1 Illustration of uplink and downlink.

also exists an entity called BBRH (Baseband Resource Handler), which has information that can be used to estimate the number of packets originating from each sender. We believe that an ML model would be able to utilize the BBRH data to make predictions about the incoming data link traffic and based on this introduce delays to the units that are sending the data. The overlying objective of this ML model can be described as attempting to ”flatten the curve” of the sent packets. This idea is illustrated in Figure 4. In the figure, we can see a black line describing the memory capacity of the switch. We can see that when the pacing is off, we frequently exceed this capacity limit. When this capacity is exceeded, the switch's memory can not fit the packets that are sent to it. This is the cause of the unwanted packet losses. The green curve describes an example of how a good pacing algorithm could avoid packet losses. Packets are initially sent to the switch at the same pace as if there was no pacing. However, when we approach the capacity of the link, delays are introduced to utilize as much of the capacity as possible without exceeding the limit. When there are no more packets to process, the link utilization quickly goes down to zero again, avoiding unnecessary delays. This can be seen as a very efficient pacing algorithm. It is able to both avoid packet losses and excessive delays.



(a) Illustration of signals being sent as they are produced, resulting in signal bursts



(b) Current pacing algorithm, where signals are spread in time using a static algorithm.

Figure 2 Illustration comparing the current pacing algorithm versus using no pacing

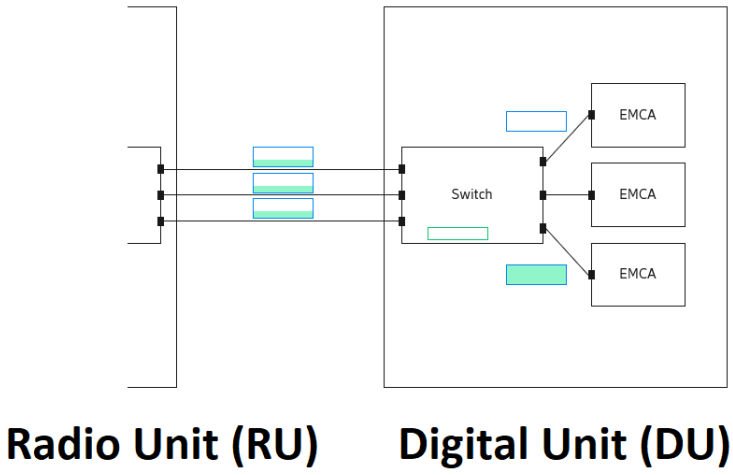


Figure 3 Overview of how the components of the data link layer are connected

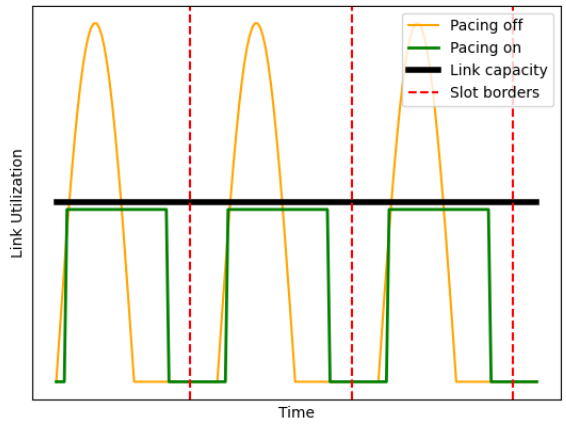


Figure 4 The objective of the ML model is to ”flatten the curve” of the sent packets.

4 Related work

During the last decade, ML methods have become increasingly popular in a multitude of different fields[7]. The telecommunication sector is no exception to this. According to one study[8], the number of publications about ML in 5G increased from around 100 in 2017 to almost 700 publications in 2021. In this section, a few of these related publications will be presented and discussed. We will also look at articles that have investigated the use of ML practices for congestion control.

4.1 Machine Learning and 5G

The launch of 5G introduces several new challenges and increased complexity in mobile communications. One approach to dealing with some of these challenges is to utilize ML. There are enormous amounts of use cases for ML in 5G[9]. Some examples of these use cases can be seen in[10]. The authors present a number of publications where ML has been used for 5G purposes, as well as discuss how the combination of these two fields can be applied in the future. They divide ML into three categories: Supervised Learning, Unsupervised Learning, and Reinforcement Learning. Each category is said to have its own suitable use cases in 5G. For example, is Supervised Learning said to be useful for predicting network demand and allocating resources. The article also discusses the biggest challenges and problems that exist in this area. It is concluded that high-quality data is essential when combining 5G and ML. Using a data set produced

from computer simulations can be sub-optimal as this data does not necessarily reflect the real world. This could potentially be a problem for the work in this report as the ML model will be trained on data from the test bench, which is only partially representative of real traffic scenarios. Another point that is discussed is that ML can increase complexity and introduce uncertainty. Because of this one must be cautious and make sure that the benefits of the ML do not hurt the interpretability of the system. The authors conclude that the goal should not be the actual ML, but how one can leverage ML to improve 5G. However, they assert that it is not a question about **if** ML will be used in future telecommunication systems, rather the question is **when** it will be fully integrated.

4.2 Congestion control using Supervised Learning

The suggested use case for supervised learning in [10] seems to correspond well with the problem that is researched in this paper. The objective of this work can be phrased as predicting network traffic and efficiently allocating the resources of the switch to avoid congestion. Controlling and avoiding congestion is a well-studied area[11], however, traditional congestion control algorithms like TCP[12] struggle to quickly adapt to the dynamic network conditions which are present in modern telecommunications systems. Lately, ML has become a popular tool for creating more dynamic congestion control algorithms. Many studies have been made trying to use ML to create dynamic TCP versions that use classifier models to detect the causes of packet losses, which can be used to increase or decrease the transmission rate[13, 14, 15]. While [13] and [14] only use RTT as input to the models, [15] believes that using only this data is not enough to create a reliable classifier, since mobile networks are very complex. Another study[16] shows that there is actually no strong evidence of a correlation between RTT and packet losses. Instead of using RTT as input, [15] uses inter-packet times and the one-way delay. These measurements are still only based on packet sending and arrival times, similar to RTT. The models that were trained for this work did not use such input. Instead, the models are fed real-time network parameters that decide the amount of produced signals.

4.3 Congestion control using Reinforcement Learning

Using RL for congestion control is a popular area of research, and many different RL algorithms have been tested[17, 18, 19, 20, 21, 22, 23]. In this article[23] the authors present a congestion control protocol based on Deep RL. In this approach, the sender of data is seen as the RL agent. This agent can act by either increasing or decreasing the

sending rate. The agent can make these decisions by looking at a number of historical network statistics, which can be seen as the states. Depending on the result of the action taken by the agent, a reward is given. This reward is based on a combination of how the action changes the throughput, latency, and loss rate. The RL agent was trained in an open-source environment that simulates network links. According to the authors, their model was able to perform as well, or even better, than the current state-of-the-art congestion control algorithms. And this is with limited training and with a not-very complex neural network. This shows that RL methods have real potential to be used for congestion control. It should be noted that this model was only evaluated in a simulated environment and it is difficult to say how it could perform when transferred to a true environment. The authors discuss that a possibility would be that if the model notices that the environment starts to diverge from what it was trained on, the system could fall back on more classic and tested congestion control algorithms. Using RL to solve our problem could be interesting as there are multiple similarities to the work done in [23]. The study has a goal similar to the one in this work, namely to maximize the performance of the network link, by minimizing loss rate and maximizing throughput. Because of this, a similar reward function could be applied to the RL agent for this problem. However, there are a few possible problems that could arise if we apply RL to solve this problem. Firstly, the data that will be used for training comes from tests that only produce signals over a time period of a few milliseconds, which is probably way too short for an agent to learn a good policy. Secondly, even if the agent would be able to learn an adequate policy, there is a good chance that the overhead of running the agent in the network would make the data link slower. Since speed is of the essence, a simpler model is most likely preferable. Also, as discussed in [10], one should be cautious in introducing too complicated ML solutions. There is a large risk that adding an RL agent to the system will exacerbate the interpretability of the system and make the code difficult to understand.

5 Method

This section contains the methods of how the work has been performed. It is divided into the subsections Data Collection, Data Visualization, Machine Learning and Pre-processing.

5.1 Data Collection

This section will be devoted to explaining how the extraction of data has been performed. First, we will discuss the Ericsson Test Benches. These are the systems that can generate the relevant data for us. After that, an explanation will be given as to what kind of data is actually interesting for this work.

5.1.1 The PHY Test Benches

The data that has been used both for ML as well as for data visualizations has been collected from the Ericsson PHY Test Benches. There are various number of different test benches that are used for different purposes, but the general goal of them all is to test the software for the Physical Layer. The Physical Layer is the lowest layer in the communication protocol. It is responsible for data transmission and reception over wireless medium[5]. Since the focus here is on the software, a simplified hardware setup, compared to the true product, can be used. The specific test bench that has been used in this work is the so-called ”ULPHY (Uplink Physical Layer) NR MDBF (Massive Digital Beamforming) Lower MCT (Multi-Component Test)”. This testbench is designed to allow DLPHY (Downlink Physical Layer) and ULPHY (Uplink Physical Layer) to run concurrently on the same EMCA. These test cases are known as ”Capacity Tests”. The reason why this is well fitting for the application of this work is that we want to be able to create test scenarios that put as high a load on the system as possible since the goal of the work is to investigate how new algorithms can control congestion and packet losses. It is possible to create and configure a large variety of different test cases that can be run on the test bench. These new test cases can be created quite easily. By adding a few lines of code in a new YAML file, test cases can be created with the exact desired configurations. Since the YAML format is so simple, it is possible to create a script that can generate a very large amount of test cases that test many different scenarios. This allows for creating large volumes of test cases without having to write them manually. Manually creating these test cases can be quite tedious as many of the parameters are dependent on each other, which makes it easy to make small mistakes that will result in a failing test case. By creating a script, it is possible to define the rules of how a test case should be configured and randomize a large number of test cases that comply with the 3GPP[24] standard. When the test cases have been created, the test bench can be run and data about the produced signals can be collected and saved so that they can be used for the ML model. The test bench produces a log file after it has run that describes what happened during the run and if there were any problems with any of the test cases. It is possible to add so-called traces that allow us to track certain values that we want to collect. These values are then written into the produced log file. It is then possible to

parse the log file and save all our data in a more structured way. For example as a CSV or a Python-Pickle file.

5.1.2 Collected Features

The parameters that are configured in the YAML files are the ones that affect the number of sent packets per slot. These will later be used as features to train an ML model that will be able to predict how many signals are sent and based on that spread them accordingly. Here, relevant features will be presented and a definition will be given to all of them. Also, a table describing the possible values for each feature is presented.

- **Number of PUSCH UEs**
  UE stands for User Equipment which can be any device that is connected to the 5G network. For example, it can be a smartphone, laptop or an IoT device.
- **Number of allocated PRBs.**
  PRB or Physical Resource Block[25] can be seen as the amount of radio resources given to a UE. These resources have to be distributed among all the UEs. Transmission can occur when one or more PRBs are assigned to a given user.
- **Number of layers.**
  The number of layers can be described as the number of data streams that can be transmitted by a single user[26]. Having multiple layers will lead to increased data throughput for that user.
- **Number of reference symbols.**
  Also known as DMRS (Demodulation Reference Symbol), the reference symbols are used by the gNodeB to estimate the characteristics of a radio channel, which helps the gNodeB to properly decode the data that is received[27].
- **Interpolation type.**
  This parameter is used for estimating the channel type.

Feature	Possible Values
Number of PUSCH UEs	1-16 (In the created test cases)
Number of allocated PRBs	1-273
Number of layers	1-2
Number of reference symbols	1-3
Interpolation type	Method 1, Method 2

Table 1 Table describing the possible values for each feature used for ML

In addition to the ML features, information about each sent signal was collected to be used for visualizing how they are sent over time. This will allow us to experiment with different signal-spreading techniques and visualize how well they work. This data includes:

- The slot number of the signal
- The arrival time of the signal
- The signal size in words
- The ID of the UE that sent the signal
- The signal type
- The symbol number of the signal

5.2 Data Visualizations

The data that has been collected from the test benches can be used to get a deeper understanding of how the signal bursts occur and how they lead to packet losses. The collected data can be used to visualize how different signal-spreading strategies will affect the number of packet losses and how much of the link capacity is used.

5.2.1 Simulating Packet Losses

The first and simplest plot that was created looked at how the signals are sent over time (Figure 5. The same plot can be created for each individual signal type (Figure 6). In this plot, it is very clear that the majority of sent signals are of the type DataInd. This plot can be extended by including the amount of data in words that each packet contains (Figure 7). This tells us how much data the switch must process over time. Figure 8 is the same plot except that it shows only the data produced by DataInd signals. The two plots (Figures 7, 8) look very similar since the majority of the sent signals are of the type DataInd. This is the reason why the previous pacing algorithm only introduces delays to signals of this specific type.

Figure 7 can be considered as a visualization of how much memory would be consumed in the switch at each time point if it was able to process each packet instantly. In reality, this is not the case. The switch requires a certain amount of time to process the data. If we assume that the switch holds the packets for a certain time in order to

process them, it is possible to make an estimation of how much of the switch's memory is consumed at each time point. This idea is visualized in Figure 9. In that figure, it is clear that the holding time greatly affects the memory consumption. In reality, there is no static time that the switch is able to process a packet but for the purpose of these visualizations, 5 microseconds was assumed to be a reasonable estimation.

Based on this switch holding time, many test cases cause the estimated switch memory consumption to reach around 8000 words. The test case that caused the highest peak created a peak of 8428 words (Figure 10). For experimenting purposes, a capacity limit was set to 7000 words, which means that all packets that cause the memory consumption to go above this value can be considered dropped packets. This value is very arbitrary, but it would mean that a good pacing algorithm will have to be able to reduce the maximum consumption by about 17%. These visualizations should not be considered an accurate description of the true memory consumption of the switch. It should rather be seen as a tool that can help us estimate if a certain signal pacing strategy will increase or decrease the memory consumption on the switch.

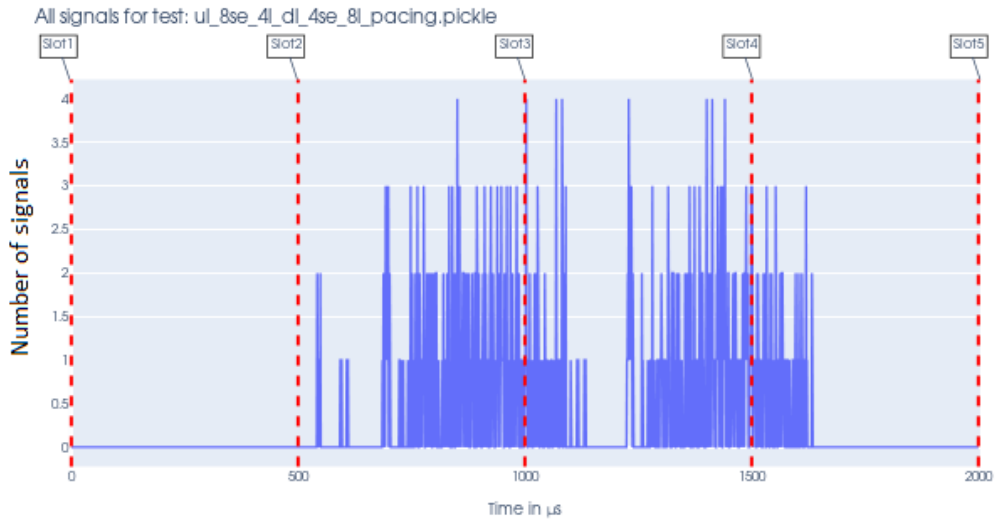


Figure 5 The number of signals sent at each time point

Figure 6 The number of signals sent at each time point for each signal type

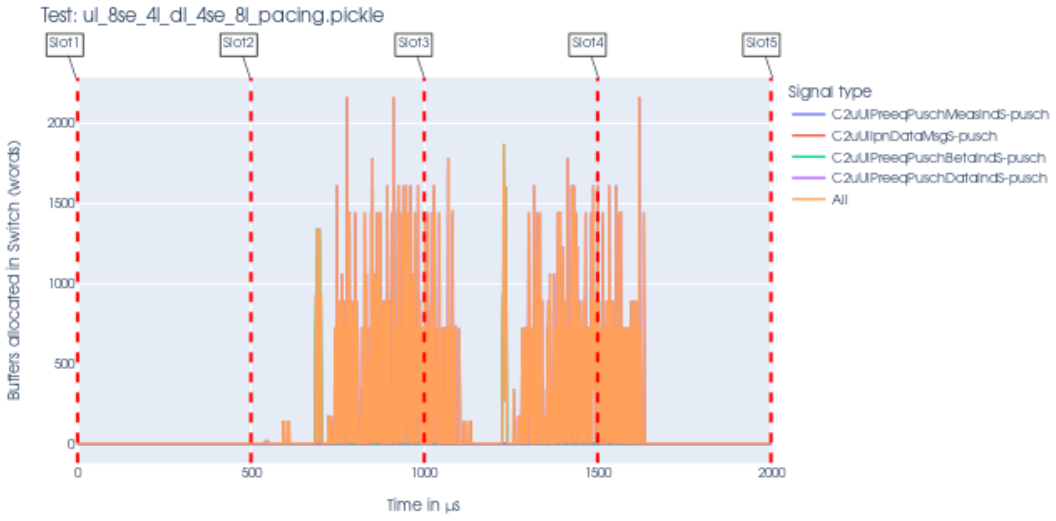


Figure 7 The amount of data sent at each time point

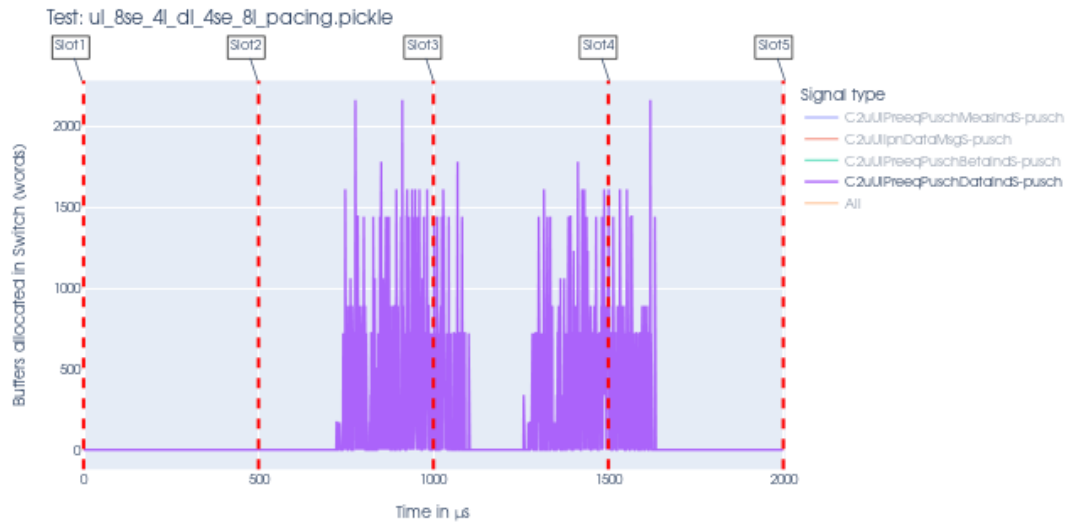


Figure 8 The amount of data sent by DataInd signals at each time point

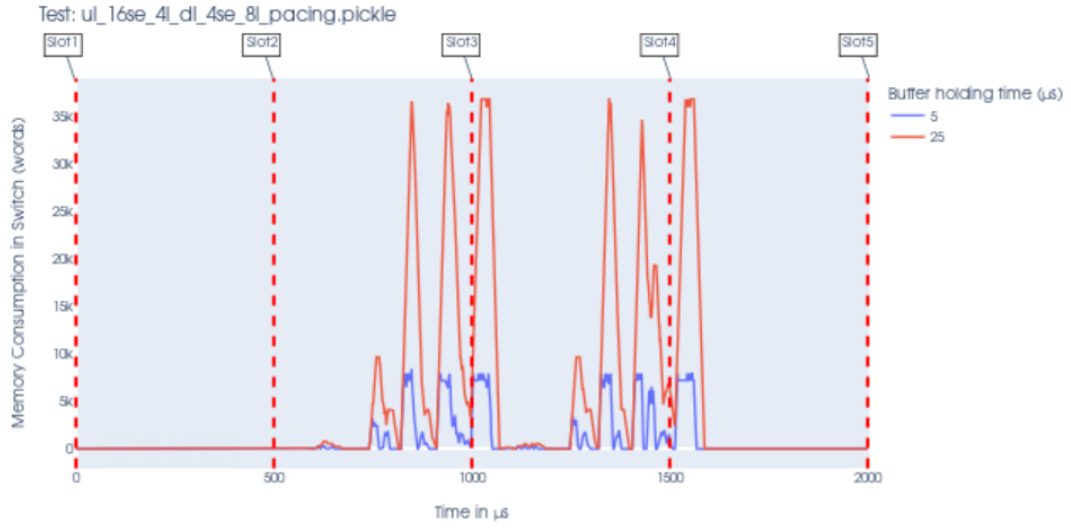


Figure 9 Estimated memory consumption in switch depending on the holding time for each packet

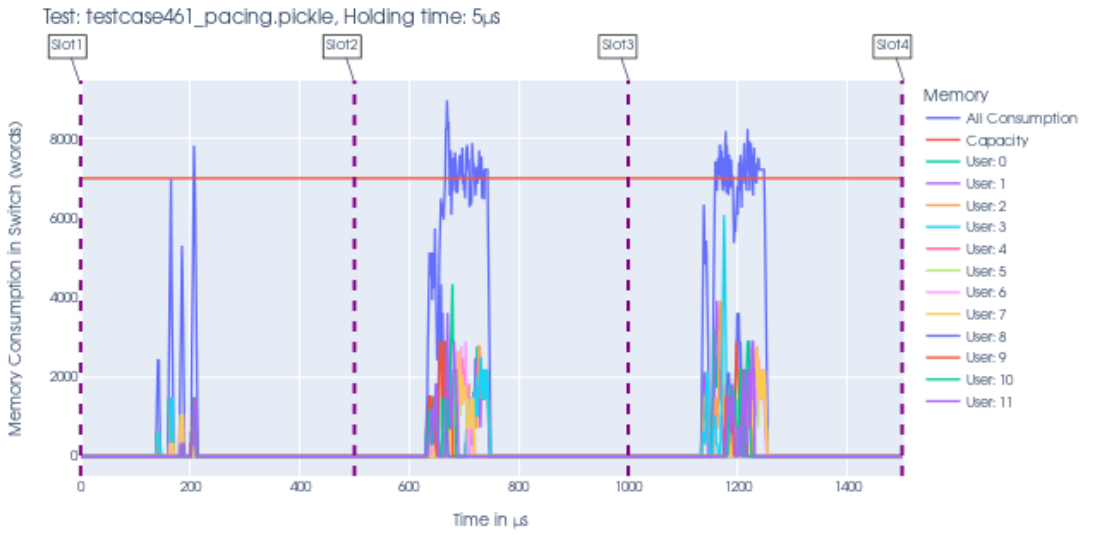


Figure 10 Memory consumption in switch exceeding the hypothetical memory capacity

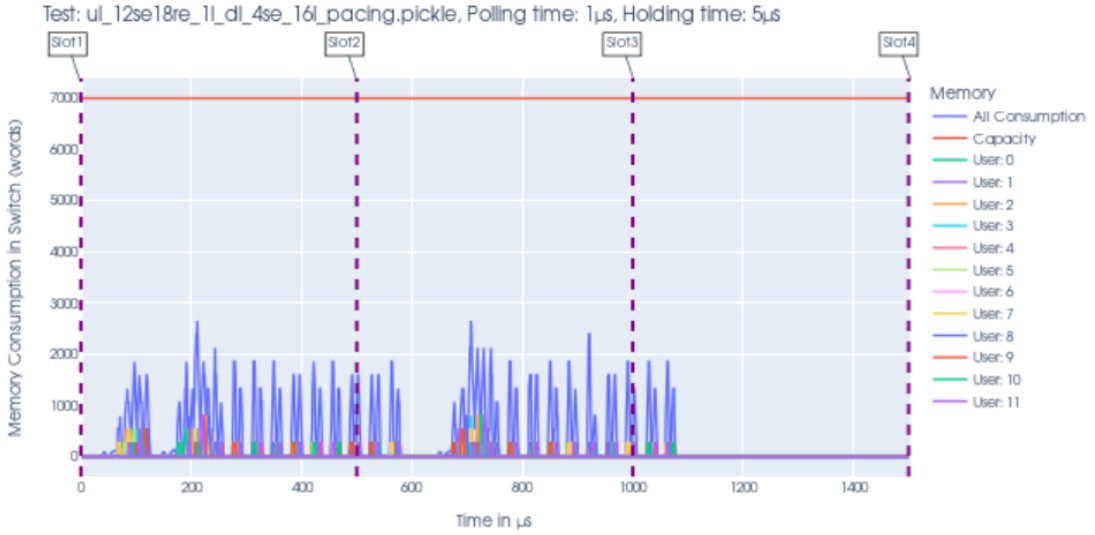


Figure 11 Test case where the hypothetical memory capacity is not exceeded

5.2.2 Spreading the Signals

The visualizations can be utilized to investigate how efficient different delay strategies are. The goal is to find a strategy that is able to lower the peak memory consumption and by that minimize packet losses as much as possible, due to the memory limitations in

the switch. However, it is also important to consider that signals should not be delayed more than is necessary. Some cases do not need any delays at all (Figure 11). Since the processing is real-time sensitive and must be finished within strict deadlines, excessive delays must be avoided. There is a trade-off that must be considered: packet losses should be avoided, but we also do not want to increase the transmission delays due to spreading in time too much.

Algorithm 1: Symbol Based Delays A possible approach to the problem is to base the delays on the symbol number of each signal. We could utilize these values to, in a sense, stretch out each slot to a longer transmission time, which could potentially aid in lowering the peak switch memory consumption and by that the packet losses. One potential delay strategy is to delay according to the function:

$$D = (S - 1) \cdot G + R \tag{1}$$

where:

D = total delay introduced to the packet
S = symbol number of the packet
G = delay gradient of the packet
R = random bonus delay introduced to the packet
$0 \leq R \leq G$

The general idea of this algorithm is to first separate the signals of different symbol numbers from each other by giving the signals a delay of their symbol number multiplied by a number *delay gradient*. A bonus random delay can then be added in order to fill the gaps that will be created between the separations. The reason for the random bonus delay being in the range specified in Equation 5.2.2 is that the goal is to evenly spread the signals in the largest possible time span without spreading them into the following symbol group. Spreading the symbol groups into each other can potentially create new peaks which is the opposite of what we want to achieve with the algorithm. One potential downside of this approach is that the randomness could introduce new peaks if we by chance schedule many signals at the same time. Another factor to keep in mind is that the *delay gradient* variable can not be too big as this could cause signals from one slot to interfere with the signals in the following slot. This can also produce new, unwanted peaks. The reason the algorithm subtracts 1 from the symbol number is to avoid introducing delays to the signals in the first symbol as these should be the first

signals sent in each slot. Similarly to the existing pacing algorithm, in this algorithm, we have decided to only look at the DataInd signals. As we are working on new territory, we wanted to simplify the problem. If it turns out this approach is viable, it would be interesting to also include delays to the other signal types.

Algorithm 2: Queue-based delays Another possible pacing algorithm is a "Queue-based" algorithm. The general idea of this algorithm is to set a limit on how much data is allowed to be sent to the switch. DU can communicate link utilization limits to every RU and data is only sent when the utilization is below a certain threshold. Before signals are sent, they are put in a queue. When the switch has processed enough data to fit another packet, the first packet in the queue will be sent to the switch. This algorithm requires that we assume a certain static time that the switch is able to process a packet in, as well as a switch memory capacity.

In a scenario where we know the exact switch holding time, this algorithm can be seen as the perfect pacing. We can keep utilizing the entire memory of the switch without running the risk of going above the threshold. In Figure 12 we can visualize an example of how the algorithm is able to significantly reduce the peak memory consumption of a test case while barely increasing the transmission time. One problem with this algorithm is that in real life, there is no static holding time in the switch. However, if we make an educated estimation of this number, this algorithm should be able to reduce the high peaks in memory consumption. The estimated holding time could be tuned up or down depending on whether we want more or less delays on the signals. ML could also be used to predict the holding times such that there will be the least possible amount of packet losses on the DU.

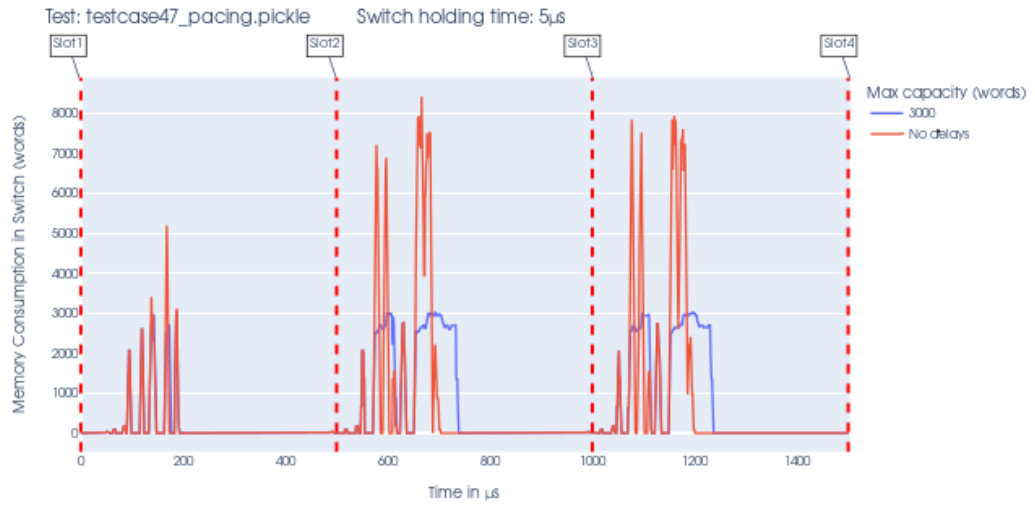


Figure 12 A certain test case without delays and the same test case with delays according to Algorithm 2

This algorithm has not been investigated further than the created plots shown in this section. The reason for this is that the overhead that this algorithm would cause is likely too great to be implemented as software. However, we believe that this algorithm has potential if it is implemented with a hardware accelerator, rather than as pure software. A hardware accelerator[28] is a hardware component that is designed for a specialized purpose. Hardware accelerators excel at what they are designed for, but they can not perform any other general processing. This means that there is always a trade-off using hardware accelerators, as while the specific function that we want to accelerate will perform better, we will have less space for other general processors.

5.3 Machine Learning

One important question that needs to be investigated is how ML should be utilized to implement Algorithm 1. The first question is whether a classifier or a regressor would be more suitable for the problem. Both approaches were investigated. In the optimal case, a regressor would be the better fit for this problem, as it would be able to find a perfect value of *delay gradient* for each scenario. The reasoning behind using a classifier is that while a regressor tries to find the optimal delay value for each test case, a classifier predicts an interval in which the optimal delay value is. While this is not the optimal solution, it might still perform in a satisfactory manner. For the classification, the test cases were separated into four different classes, with different values of *delay gradient*.

The following classes were created: 0, 6, 12, and 20. In total, the data set is made up of 550 test cases, and the respective classes are made up of 91, 158, 148, and 153 samples. The test cases were labelled according to the histogram in Figure 14. For example, the test cases that make up bin 0 are assigned to class 0, and bins 1 to 6 are assigned to class 6. The reasoning behind choosing these specific classes was that firstly, one class is needed for the maximum needed delay and one class for no delays. Secondly, the three non-zero classes were chosen so that these classes would have roughly the same number of samples. Class 0 likely does not need as many samples as the other classes, since the samples in this class should be very similar. For the regression, each test case was labelled according to Figure 14 without modifications. Three different classification algorithms were explored; Decision Tree, Random Forest, and Logistic Regression. Random Forest was also used for regression, which was the only regression algorithm explored. To create and test the ML, the Python module Scikit-learn[29] was used. Scikit-learn is an open-source module that makes it easy to create ML models.

5.3.1 Machine Learning Algorithms

Decision Tree A decision tree[30] is a classification algorithm that as the name suggests, uses a hierarchical tree-like structure to classify a data point. A decision tree is made up of three types of nodes. A root node, internal nodes, and leaf nodes. All classification starts in the root node. Based on one of the input features of the data point, the data point will be directed to an internal node either down left or down right. This process will continue until we have reached a leaf node. In the leaf nodes, we decide which class a data point should be assigned to. A simple visualization of this idea can be seen in Figure 13.

But how is the tree created? To find the optimal splits, we use a ”splitting criteria”. These techniques help us evaluate how ”pure” or ”impure” a node is. A pure node is a node that only contains samples from one single class. A maximally impure node is a node that contains an equal amount of samples from all classes. We use the splitting criteria techniques to find the purest splits, to separate the classes as much as possible. We have investigated two different splitting criteria: Entropy and Gini Index. The impurity of the splits is calculated according to:

$$H(n) = -\sum_{i=1}^{c} p(i|n) \log_2 p(i|n) \tag{2}$$

Equation 2: Formula for Entropy impurity

$$G(n) = 1 - \sum_{i=1}^{c} p(i|n)^2 \tag{3}$$

Equation 3: Formula for Gini Index impurity

Both Entropy and Gini Index return 0 for a completely pure node and 1 for a maximally impure node.

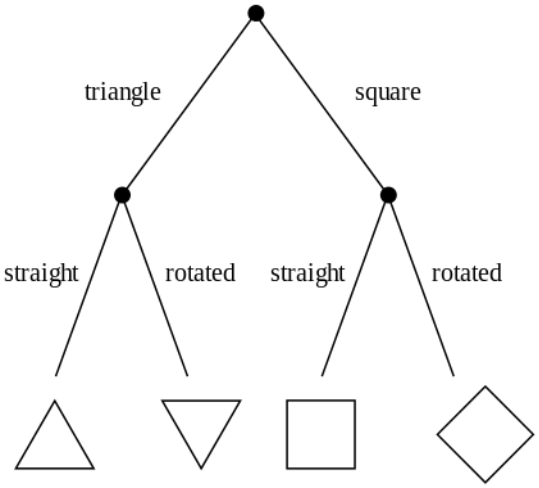


Figure 13 Simple Decision Tree

Random Forest Random Forest[31] is a so-called ensemble technique. The idea of ensemble techniques is to combine several weak classifiers to create one strong classifier. In the case of Random Forest, the weak classifiers are Decision Trees. Each of these classifiers votes on which class they think a sample belongs to, and the sample is assigned to belong to the popular vote. In the regression version, the average value is taken instead. Another difference is that the regression version does not use Entropy and Gini Index as splitting criteria. Instead, Squared Error and Absolute Error are used. Random Forest uses Bagging[30] (Bootstrap Aggregating). This means that each of the weaker classifiers is trained on different subsets of the original dataset. The subsets are randomly created, which could mean that some samples will not be used at all in training, while others are used several times.

Logistic Regression Logistic Regression[32] classifies a sample by calculating the class probabilities.

$$p(y = m|\boldsymbol{x}) \tag{4}$$

$$g(\boldsymbol{x}) = 1 - \frac{e^{\theta^T \boldsymbol{x}}}{1 + e^{\theta^T \boldsymbol{x}}} \tag{5}$$

$$\hat{\boldsymbol{\theta}} = \underset{\theta}{\operatorname{argmin}} \frac{1}{n} \sum_{i=0}^{n} \ln(1 + e^{-y_i \boldsymbol{\theta}^T \boldsymbol{x}_i}) \tag{6}$$

Equation 4 can be described in words as the probability that a sample **x** belongs to class **m**. To calculate this probability Logistic Regression uses the Sigmoid Function (Equation 5). **x** is the input feature vector, and θ is an unknown parameter vector that must be calculated. This vector is found by solving Equation 6. The Sigmoid Function is only used for binary classification problems. The return value of the function signifies the probability of the input sample belonging to one of the classes. However, if we have a problem with more than two classes we have to use a different function. In these cases, we have to use the Softmax Function (Equation 7), where we need a weight vector θ for each class.

$$p(y = m|\boldsymbol{x}) = \frac{e^{\boldsymbol{x}^T \boldsymbol{\theta}_m}}{\sum_{k=1}^{K} e^{\boldsymbol{x}^T \boldsymbol{\theta}_k}} \tag{7}$$

5.4 Pre-processing

Because of the way that Random Forest and Decision Trees use Gini Index and Entropy to classify data points, these two algorithms do not require any specific pre-processing of the training data. Logistic Regression on the other hand does. Two preprocessing techniques were utilized before training the Logistic Regression model, Normalization and One-Hot-Encoding[32]. Normalization is the practice of rescaling features to all be in the same range (0-1). This is to avoid features of higher magnitude dominating over features of lower magnitude, causing the trained model to think that the features of higher magnitudes are more important. The other technique, One-Hot-Encoding is used for categorical classes. The developed ML models in this work use some features that are nominal categorical data (for example the Interpolation type) that are described by numbers. This can be misinterpreted as ordinal data by the ML model which can be problematic. As there is no numeric relationship between the categories within the class, we need another way of describing the class. The way One-Hot-Encoding handles this is by splitting each category into multiple binary classes, thereby removing the possibility of interpreting the class as ordinal.

6 Results

The results of this work can be divided into three parts. Firstly the performance of Algorithm 1 will be presented. After that, the created ML models will be discussed. Also, some experiments were done trying to investigate the performance of Algorithm 1 on real hardware, rather than on test bench data.

6.1 Performance of Algorithm 1

To judge how well this algorithm performs, investigations were made for each test case to see what was the minimum value of *delay gradient* that is able to bring the switch memory consumption down below the capacity limit of 7000 words. It was shown that the algorithm was able to achieve this for all test cases with a maximum *delay gradient* value of 20µs (Figure 14).

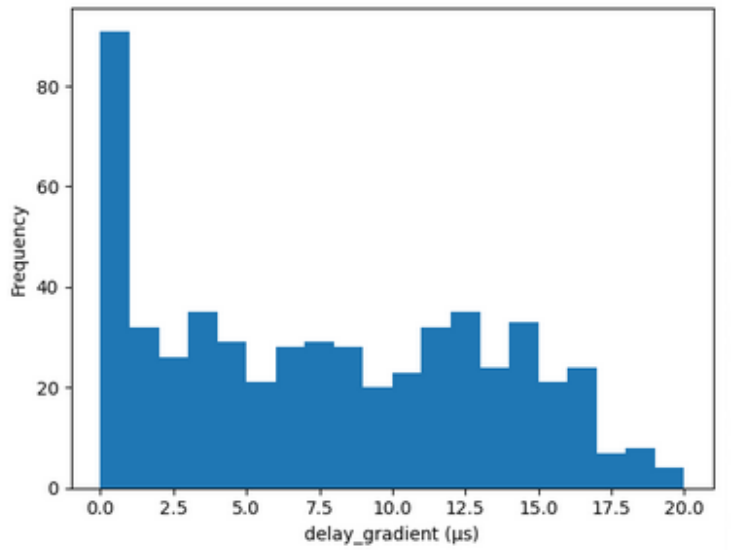


Figure 14 The minimum value of *delay gradient* to reduce the switch memory consumption below 7000 words.

If we assume that 20µs will be the largest number that *delay gradient* can be assigned, it means that the maximum delay that can be applied to an individual signal is $(14 - 1) \cdot 20 + 20 = 280$µs. For the test case with the highest peak in memory consumption (Figure 10), applying delays according to equation 1 with a *delay gradient* value of 20, it was possible to reduce the peak from 8428 words to 5842 (Figure 15).

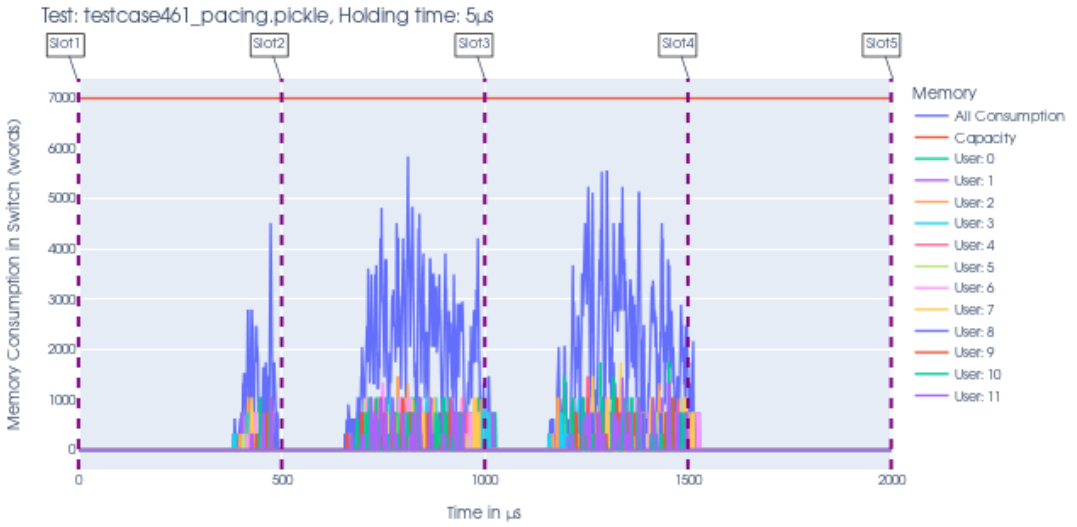


Figure 15 The effect of applying delays according to Algorithm 1 on the test case in Figure 10.

This is a reduction of around 31%. Even for test cases such as this one, where longer delays are required, we can see that the signals from different slots are separated. This is exactly what we want, as this will avoid creating new unwanted peaks in memory consumption. An ML model should be able to predict what value of *delay gradient* is suitable for each test case to reduce its memory consumption below a certain threshold.

6.2 Performance of machine learning models

For each of the ML algorithms that were presented in 5.3, one model was created. To find the optimal hyperparameters for each algorithm, the Scikit-learn[29] method GridSearchCV was utilized. GridSearchCV takes a grid of parameters and fits a model for each combination of these parameters. It then returns the model that had the best Cross Validation score. This resulted in the following choice of parameters:

- Decision Tree:
  {'criterion': 'gini',
  'max_depth': 8,
  'min_samples_leaf': 3,
  'min_samples_split': 3}

- Random Forest Classification:
  {'criterion': 'entropy',
  'max_depth': 6,
  'min_samples_leaf': 2,
  'min_samples_split': 4,
  'n_estimators': 50}
- Random Forest Regression
  {'criterion': 'squared_error',
  'max_depth': 10,
  'min_samples_leaf': 1,
  'min_samples_split': 4,
  'n_estimators': 100}
- Logistic Regression:
  {'max_iter': 100,
  'penalty': 'l1',
  'solver': 'liblinear'}

The classification models were evaluated based on two factors: Cross Validation score, as well as by looking at confusion matrices. Cross Validation[33], works by splitting the data set into multiple subsets. We iterate over each subset and use it as a test set while training a model on the remainder of the data. When all subsets have been used as test data, the average accuracy is calculated, which is the Cross Validation score. The benefit of Cross Validation is that it can help with avoiding over-fitting. If we only have a single training set, and try to tweak the model to perform as well as possible on that small set of data it is likely over-fitting will occur. Confusion Matrices[32] can be seen as a summary of the performance of an ML model. It describes the predicted classes versus the true classes. The Confusion Matrices are important because we can potentially tolerate test cases that are misclassified, as long as they are classified in a neighbouring class. This is because some test cases can be seen as edge cases between classes, meaning they can fit in both classes. Even for test cases that are not necessarily edge cases, misclassifying to the neighbouring class is always better than misclassifying to a distant class. If all incorrectly classified test cases are classified as an adjacent class, we can say that the model is able to see a pattern in the data, even though it may not perform perfectly. To create the confusion matrices, the data set was split into a training and test set with a ratio of 80% to 20%. The numbers in the matrices come from the models making predictions on the test set. For the Cross Validation, a fold of 5 was applied. The scores of each model can be seen in Table 2. In this table, we also look at "Misclassified Adjacent Class (%)". To phrase this more clearly: Out of the misclassified test cases, what percentage were classified to an adjacent class?

The Decision Tree model and the Random Forest model performed significantly better than the Logistic Regression. The Logistic Regression was only able to correctly classify about two-thirds of the test cases and was spectacularly poor at recognizing the class for *delay gradient* $= 12$. As can be seen in Figure 17c, this model is decent at identifying the classes 0 and 20 but is lacking for the two middle classes. The Random Forest and Decision Tree models had a similar Cross Validation score and their confusion matrices also look quite similar (Figures 17a and 17b). Both of the models perform well on classes 0, 6 and 20, but struggle with class 12, especially the Random Forest model. The Decision Tree model outperforms the Random Forest model for class 12, while the Random Forest model is perhaps performing slightly better for class 20. It is possible that the reason for the Logistic Regression model not performing as well as the other two models is because of the chosen features. Logistic Regression is sensitive to irrelevant features and easily overfit. Adding more training data, maybe in combination with more aggressive regularization would likely lead to a better Logistic Regression model[34].

The Random Forest and Decision Tree models do show promise, even if they do not perform perfectly. Even though we probably would like to have a slightly higher accuracy than 72%, the fact that the 28% that are misclassified are always (almost) classified as an adjacent class, seems to support this solution has potential. If the model's bad performances for class 12 could be improved, the model would actually have very good accuracy. Some possible ways to try and achieve this would be to either create more training data for this class or to experiment with a different *delay gradient* value for this class.

The Random Forest Regression model was evaluated using Mean Absolute Error. 100 different training and test set splits of sizes 80/20 were used for training the model, and the average Mean Absolute Error from these splits was an impressively low 1.54. A summary of the errors from one of these splits can be seen in Figure 16. The model is able to achieve an error of 2 or less for 70% of the test cases, which is quite impressive. In table 3 an approximate comparison between the regressor model and the classifiers is made. The centre value of each bin predicted by the models is compared to the centre value of the bin of the true value. The comparison between these values can be used to calculate an approximate Mean Absolute Error for each of the classifier models. According to these approximations, the Decision Tree and the Random Forest model have a similar performance and a lower error than the other two models.

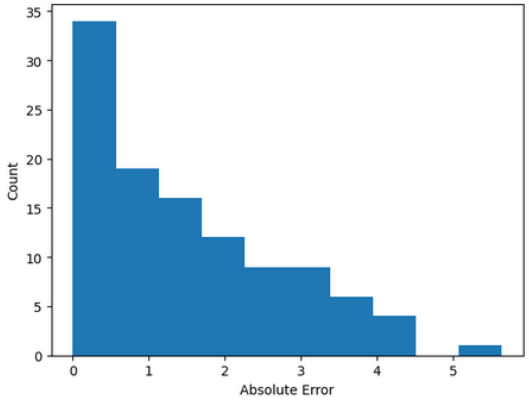


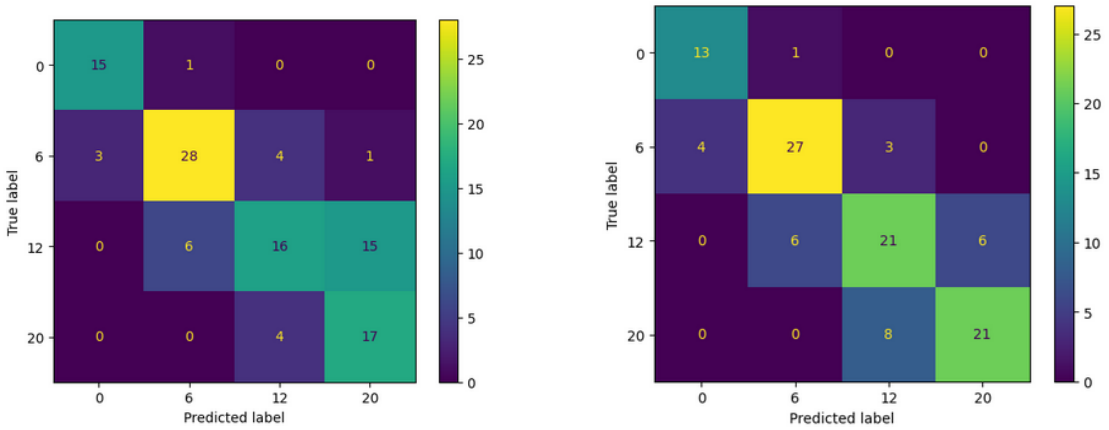
Figure 16 Summary of the Mean Errors from the Random Forest regression model

Algorithm	Average CV-Score	Misclassified Adjacent Class (%)
Decision Tree	71.27%	100%
Random Forest	72.00%	99.09%
Logistic Regression	63.27%	93.64%

Table 2 The performance of the classifier models

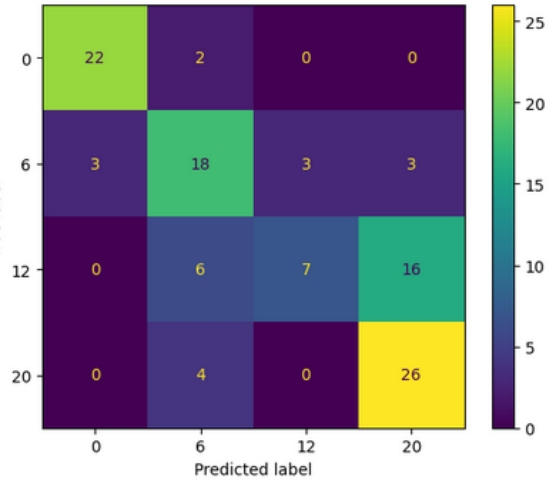
Algorithm	Approximated Mean Absolute Error
Random Forest Classifier	2.00
Decision Tree	1.54
Logistic Regression	2.50

Table 3 Approximations of Mean Absolute Errors for the classifier models



(a) Confusion Matrix for the Random Forest model

(b) Confusion Matrix for the Decision Tree model



(c) Confusion Matrix for the Logistic Regression model

Figure 17 Confusion Matrices for the ML models

6.3 Lab Testing

The ML model was never fully implemented into the product, due to the work being under a time restriction. However, as a first test to investigate how Algorithm 1 functions on true hardware, some lab tests were done. Lab tests allow us to conduct more realistic tests than the Test Benches since they use a more realistic hardware setup. The goal of the tests was to find out how effective Algorithm 1 is at reducing packet losses. This was done by setting up high-traffic test runs where all signals were delayed according

to Algorithm 1 with *delay gradient* 20. This test run was then compared with a run with the current pacing solution. Results showed that both algorithms were able to achieve 0 packet losses with the run configuration. This seems to support that the algorithm is solid and that we have achieved one of the objectives: reducing the peak load on the switch and thereby avoiding packet losses. The second objective, maximizing the data link's utilization, is the ML model's responsibility. The next step in testing would be to implement one of the ML models into the product and set up a test run with the model in use.

7 Discussion

The findings in this work seem to support the notion that an ML model could be capable of managing the data link capacity. The visualizations from Section 6.1 showed that Algorithm 1 significantly reduces the peak memory consumption of the switch, while still managing to maintain a satisfactory transmission time. However, until more lab tests are made, we can not conclude that the algorithm will function as well in reality as the visualizations show. As the writers concluded in [10] high-quality data is a very important factor when developing ML models for this area. The data that has been collected for this work originates from the test benches whose purpose is to test software and not to accurately describe signal streams. This could mean that even though the models show promising results on our collected data, it is unclear how the models will perform in the true environment, as it could be very different from what our models have been trained on. Another potential issue with this work is how the memory consumption of the switch has been modelled. The buffer holding time of the switch is a vital component in how much of the switch memory is being consumed at each point in time, as shown in Figure 9. It is very likely that the numbers in the memory consumption visualizations are not very accurate. However, this is not the objective of these plots. The important factor is whether or not they are able to accurately capture the overall trend of the curve to describe when consumption increases or decreases. It is difficult to be sure if the visualizations accomplish this, but the lab test results seem to suggest that this is the case.

If it turns out that the machine-learning models created are not able to manage the data link capacity in a satisfactory manner, this does not mean that the findings in this paper should be dismissed. Rather, it means that we need to create more realistic data and a more realistic description of the switch memory consumption. This would significantly increase the chances to create models of higher quality. Since this project was conducted under a restricted time period, some simplifications and assumptions regarding these parameters had to be made. Because of this, this work should be seen as a

first investigation of potential ways to attack this problem, rather than a finished product.

In this work, both classifier and regressor models were explored. Regression models performed better than the classifiers using a limited data set and it is likely that this difference will increase as more/better data is accumulated. This is because regressors should have a higher potential for this specific problem, as they would be capable of finding the optimal delay for each combination of input features.

8 Conclusion

This work suggested two potential ML-based algorithms to be used for managing the data link layer capacity. Algorithm 1 was based on the signals symbol numbers and Algorithm 2 used a queue-based strategy. Algorithm 2 was not thoroughly evaluated due to the belief that the algorithm needed hardware accelerators to perform satisfactorily. Since this project was under a time restriction, and implementing hardware accelerators would likely take a lot of time, this algorithm was not prioritized. Four machine-learning models to be used for Algorithm 1 were created and trained on data collected from the Ericsson test benches. Three of them showed promising results. One model was a Random Forest Classifier model, one was a Random Forest Regression and the last one was a Decision Tree model. These classifier models were able to achieve an average cross-validation score of 72.00% and 71.27% respectively, while the regression model had a Mean Absolute Error of 1.54. While a higher cross-validation score could be desired for these classifier models, their very high Misclassified Adjacent Class scores seem to suggest that the models have a certain degree of proficiency in identifying the traffic volume. The performance of the regression model also showed potential. Comparisons between the classifiers and the regression models showed that it could perform on par or better than the classifier models. More training data is likely necessary to create very proficient models, but the findings of this work seem to support the notion that ML models can indeed use network dynamics to predict traffic volume.

One of the goals of the developed algorithms was to significantly significantly reduce the peak memory consumption of the switch in order to avoid packet losses. Lab tests showed that Algorithm 1 was able to, similarly to the currently used algorithm, completely avoid packet losses in a high-traffic test setup. The second goal of the algorithm has not been as thoroughly evaluated. Since the setup of the lab tests only used a static version of the algorithm, without any ML, it is not possible to use the results of these tests to evaluate the algorithm's ability to avoid excessive delays. The created visualizations from the test bench data seem to suggest that the algorithm is able to achieve the minimum stated requirement that packets from different slots should not overlap with

each other. However, since these visualizations rely on simplifications and estimations, these results should not be used to draw any strong conclusions. The third and final goal, that the algorithm should be able to adapt to both high and low traffic, is completely dependent on the ML model's ability to predict the traffic volume. It is clear from the lab tests that Algorithm 1 is able to handle high traffic. Also, because of the way the algorithm is designed, if the ML model predicts very low traffic, the algorithm will not introduce any delays, meaning that the algorithm is able to properly handle low traffic properly as well. While it is unclear what the optimal *delay gradient* values are for traffic volumes in between the two extremes, Algorithm 1 should be able to adapt to these traffic scenarios as well, assuming that the ML model performs satisfactory and the range for the *delay gradient* values is properly tuned.

9 Future work

The immediate next step for this work would be to fully implement one of the machine-learning models into the product and to conduct lab tests. This would allow for more accurate conclusions about the performance of the solution. If the results are promising, then the next step would be to collect more data to further improve the models. This would likely lead to even better performance. However, if the results disappoint, we likely do not need more data, but better data. It would then be necessary to try and collect real data, instead of the currently used test bench data. The way of doing this is not clear and is currently an open question. It would also be interesting to collect test cases from specific high-traffic situations. For example, sports events or concerts where a lot of people are gathered in a small space.

Algorithm 1 appears to function quite well, but it could likely be improved upon. For most test cases (as in figure 15), we can see that this delay strategy causes the memory consumption to gradually increase, reach a peak, and then gradually decrease down to 0 again. This creates a bell shape in the plots. This seems to suggest that the signals in the first and the last symbols are delayed more than they need to. An idea would be to instead of linearly increasing the delays for each symbol, introduce some sort of logarithmic delays (Figure 18). This means that we gradually increase the delays for the first few symbols, and then let the increases in delays pan out towards the last few symbols. With this approach, we could possibly achieve a more narrow, rectangle-shaped plot while keeping the same maximum peak. A successful implementation of this strategy could lead to a lower transmission time, however, no such algorithm has been thoroughly explored. More lab tests should also be conducted experimenting with the *delay gradient* value. The maximum value of 20 that was used for this work is likely not the true maximum. It should be interesting to conduct tests with both a lower and

a higher *delay gradient* value and see if there is a difference in the amount of packet losses.

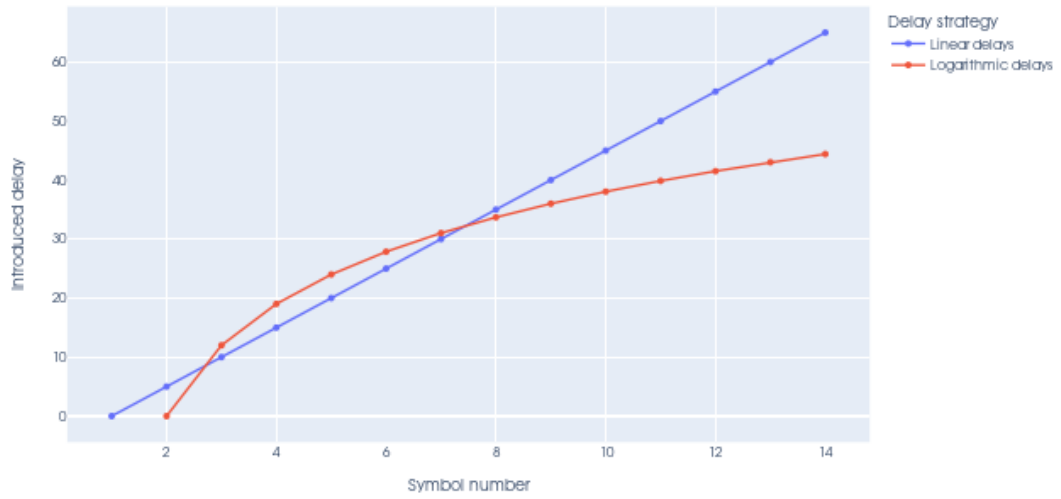


Figure 18 Comparing linear and logarithmic delay strategies.

Algorithm 2 has not been explored beyond the plots discussed in 5.2.4. This was due to the need for hardware implementations which was not the scope of this project. However, it should be interesting to conduct further experiments on this algorithm. The fact that it is able to create a rectangle-shaped plot, which Algorithm 1 struggles with at the moment, suggests that it has potential. Another good thing about the algorithm is that it should be easy to increase or decrease the rate at which signals are allowed to be sent to the switch by simply changing the assumed switch holding time and capacity limit.

10 Acknowledgments

In closing, I want to express my gratitude to all the people who helped me during this project. Thank you to my supervisors Nameen Abeyratne and Göran Hammentun, as well as my manager Zaheer Ahmed. Big thanks to Deepsantosh Sail and Debasis Jana for your assistance. I also want to give a big thank you to my subject reviewer Jens Sjölund for helping me with the writing process.

References

[1] T. Nakamura, "5g evolution and 6g," in *2020 IEEE Symposium on VLSI Technology*, 2020, pp. 1–5.

[2] K. B. Letaief, W. Chen, Y. Shi, J. Zhang, and Y.-J. A. Zhang, "The roadmap to 6g: Ai empowered wireless networks," *IEEE Communications Magazine*, vol. 57, no. 8, pp. 84–90, 2019.

[3] V. P. Rekkas, S. Sotiroudis, P. Sarigiannidis, S. Wan, G. K. Karagiannidis, and S. K. Goudos, "Machine learning in beyond 5g/6g networks—state-of-the-art and future trends," *Electronics*, vol. 10, no. 22, 2021. [Online]. Tillgänglig: https://www.mdpi.com/2079-9292/10/22/2786

[4] Ericsson, "Employing ai techniques to enhance returns on 5g network investments," 2019.

[5] A. Zaidi, *5G physical layer : principles, models and technology components*. London, United Kingdom: Academic Press, an imprint of Elsevier, 2018.

[6] D. R. Bhadra, C. A. Joshi, P. R. Soni, N. P. Vyas, and R. H. Jhaveri, "Packet loss probability in wireless networks: A survey," in *2015 International Conference on Communications and Signal Processing (ICCSP)*, 2015, pp. 1348–1354.

[7] Y. Kumar, K. Kaur, and G. Singh, "Machine learning aspects and its applications towards different research areas," in *2020 International Conference on Computation, Automation and Knowledge Management (ICCAKM)*, 2020, pp. 150–156.

[8] I. Turker and S. O. Tan, "Machine learning vs. deep learning in 5g networks – a comparison of scientific impact," 2022. [Online]. Tillgänglig: https://arxiv.org/abs/2210.07327

[9] H. Fourati, R. Maaloul, and L. Chaari, "A survey of 5g network systems: challenges and machine learning approaches," *International journal of machine learning and cybernetics*, vol. 12, no. 2, pp. 385–431, 2021.

[10] M. E. Morocho-Cayamcela, H. Lee, and W. Lim, "Machine learning for 5g/b5g mobile and wireless communications: Potential, limitations, and future directions," *IEEE Access*, vol. 7, pp. 137 184–137 206, 2019.

[11] H. Haile, K.-J. Grinnemo, S. Ferlin, P. Hurtig, and A. Brunstrom, "End-to-end congestion control approaches for high throughput and low delay in 4g/5g cellular networks," *Computer Networks*, vol. 186, p. 107692, 2021. [Online]. Tillgänglig: https://www.sciencedirect.com/science/article/pii/S1389128620312974

[12] P. Sarolahti and A. Kuznetsov, "Congestion control in linux tcp." in *USENIX Annual Technical Conference, FREENIX Track*, 2002, pp. 49–62.

[13] I. El Khayat, P. Geurts, and G. Leduc, "Improving tcp in wireless networks with an adaptive machine-learnt classifier of packet loss causes," in *NETWORKING 2005. Networking Technologies, Services, and Protocols; Performance of Computer and Communication Networks; Mobile and Wireless Communications Systems*, R. Boutaba, K. Almeroth, R. Puigjaner, S. Shen, and J. P. Black, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 549–560.

[14] M. C. Jun Liu, Ibrahim Matta, "End-to-end inference of loss nature in a hybrid wired/wireless environment," *WiOpt'03: Modeling and Optimization in Mobile, Ad Hoc and Wireless Network*, 2003. [Online]. Tillgänglig: https://inria.hal.science/inria-00466774

[15] N. Fonseca and M. Crovella, "Bayesian packet loss detection for tcp," in *Proceedings IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies.*, vol. 3, 2005, pp. 1826–1837 vol. 3.

[16] S. Biaz and N. Vaidya, "Distinguishing congestion losses from wireless transmission losses: a negative result," in *Proceedings 7th International Conference on Computer Communications and Networks (Cat. No.98EX226)*, 1998, pp. 722–731.

[17] N. Jay, N. Rotman, B. Godfrey, M. Schapira, and A. Tamar, "A deep reinforcement learning perspective on internet congestion control," in *Proceedings of the 36th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. PMLR, 09–15 Jun 2019, pp. 3050–3059. [Online]. Tillgänglig: https://proceedings.mlr.press/v97/jay19a.html

[18] J. Fang, M. Ellis, B. Li, S. Liu, Y. Hosseinkashi, M. Revow, A. Sadovnikov, Z. Liu, P. Cheng, S. Ashok, D. Zhao, R. Cutler, Y. Lu, and J. Gehrke, "Reinforcement learning for bandwidth estimation and congestion control in real-time communications," 2019.

[19] V. Sivakumar, O. Delalleau, T. Rocktäschel, A. H. Miller, H. Küttler, N. Nardelli, M. Rabbat, J. Pineau, and S. Riedel, "Mvfst-rl: An asynchronous rl framework for congestion control with delayed actions," 2021.

[20] S. Emara, B. Li, and Y. Chen, "Eagle: Refining congestion control by learning from the experts," in *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*, 2020, pp. 676–685.

[21] Z. Xu, J. Tang, C. Yin, Y. Wang, and G. Xue, "Experience-driven congestion control: When multi-path tcp meets deep reinforcement learning," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 6, pp. 1325–1336, 2019.

[22] S. Abbasloo, C.-Y. Yen, and H. J. Chao, "Classic meets modern: A pragmatic learning-based congestion control for the internet," in *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication*, ser. SIGCOMM '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 632–647. [Online]. Tillgänglig: https://doi.org/10.1145/3387514.3405892

[23] N. Jay, N. Rotman, P. Godfrey, M. Schapira, and A. Tamar, "Internet congestion control via deep reinforcement learning," 10 2018.

[24] 3rd Generation Partnership Project, "About 3gpp." [Online]. Tillgänglig: https://www.3gpp.org/about-3gpp

[25] S. P. Federica Rinaldi, Alessandro Raschella, "5g nr system design: a concise survey of key features and capabilities," *Springer*, pp. 5178–5180, 2021.

[26] F. G. H. K. L. Von Butovitsch, Astely, "Massive mimo for 5g networks," 2017.

[27] M. H. C. Garcia, A. Molina-Galan, M. Boban, J. Gozalvez, B. Coll-Perales, T. Şahin, and A. Kousaridas, "A tutorial on 5g nr v2x communications," *IEEE Communications Surveys Tutorials*, vol. 23, no. 3, pp. 1972–2026, 2021.

[28] D. Serpanos and T. Wolf, "Chapter 11 - specialized hardware components," in *Architecture of Network Systems*, ser. The Morgan Kaufmann Series in Computer Architecture and Design, D. Serpanos and T. Wolf, Eds. Boston: Morgan Kaufmann, 2011, pp. 211–227. [Online]. Tillgänglig: https://www.sciencedirect.com/science/article/pii/B9780123744944000116

[29] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[30] J. Rogel-Salazar, *Data Science and Analytics with Pytho.)n (1st ed.* Chapman and Hall/CRC, 2017. [Online]. Tillgänglig: https://doi-org.ezproxy.its.uu.se/10.1201/9781315151670

[31] L. Breiman, "Random forests," *Springer*, 2001. [Online]. Tillgänglig: https://link.springer.com/article/10.1023/A:1010933404324

[32] A. Lindholm, N. Wahlström, F. Lindsten, and T. B. Schön, *Machine Learning - A First Course for Engineers and Scientists*. Cambridge University Press, 2022. [Online]. Tillgänglig: https://smlbook.org

[33] C. Schaffer, "Selecting a classification method by cross-validation," *Springer*, 1993. [Online]. Tillgänglig: https://link.springer.com/article/10.1007/BF00993106

[34] A. Ng, "Feature selection, l 1 vs. l 2 regularization, and rotational invariance," *Proceedings of the Twenty-First International Conference on Machine Learning*, 09 2004.