Slovak University of Technology Bratislava
Faculty of Informatics and Information Technologies

Ing. Peter Bakonyi

Dissertation Thesis Abstract

Data centered network enhancement

to obtain the Academic Title of philosophiae doctor (PhD.)

Degree course: Applied Informatics
Field of study: Applied Informatics
Form of study: Internal
Workplace: Institute of Computer Engineering and Applied Informatics,
FIIT STU Bratislava

Bratislava 2023

Dissertation Thesis has been developed at the Institute of Computer Engineering and Applied Informatics, Faculty of Informatics and Information Technologies, Slovak University of Technology in Bratislava.

Submitter:	Ing. Peter Bakonyi
Location:	Institute of Computer Engineering and Applied Informatics Faculty of Informatics and Information Technologies Slovak University of Technology in Bratislava
Supervizor:	prof. Ing. Ivan Kotuliak, PhD.
Reviewers:	prof. Ing. Pavel Segeč, PhD. Doc. Ing. Miroslav Michalko, PhD.

Keywords: Software defined networks, Content delivery network
Machine learning

Dissertation Thesis Abstract was sent: .....

Dissertation Thesis Defence will be held on ..... at ..... pm at the Institute of Computer Engineering and Applied Informatics, Faculty of Informatics and Information Technologies, Slovak University of Technology in Bratislava (Ilkovicova 2, Bratislava).

Prof. Ing. Ivan Kotuliak, PhD.

Dean of FIIT STU in Bratislava

Annotation

Slovak University of Technology Bratislava
Faculty of Informatics and Information Technologies
Degree Course: Applied informatics

Author: Ing. Peter Bakonyi
Dissertation Thesis: Data centered network enhancement
Supervisor: prof. Ing. Ivan Kotuliak, PhD.
July 2023

The improvement of QoS in networks is now more critical than ever. The insurance of the smooth functioning and content deliverability in the network is becoming harder with the increased load in past years. Components like maintenance, scaling, content caching, and network utilization need to be continuously monitored to ensure high QoS. We focus on creating a network where artificial intelligence takes care of it. With that, we create a zero-touch network and enhance QoS.

Anotácia

Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií
Študijný program: Aplikovaná informatika

Autor: Ing. Peter Bakonyi
Dizertačná práca: Dátovo Orientované vylepšenie počítačovej siete
Vedúci dizertačného projektu: prof. Ing. Ivan Kotuliak, PhD.
Júl 2023

Zlepšenie QoS v sieťach je teraz kritickejšie ako kedykoľvek predtým. Poistenie plynulého fungovania a dodávky obsahu v sieti sa so zvyšujúcim sa zaťažením v posledných rokoch stáva ťažším. Súčasti ako údržba, škálovanie, ukladanie obsahu do pamäte cache, využívanie siete sa musia nepretržite monitorovať, aby sa zabezpečila vysoká QoS. Zameriavame sa na vytvorenie siete, kde sa o ňu stará umelá inteligencia. Týmto vytvárame sieť s nulovým dotykom a vylepšujeme QoS.

Contents

Chapter 1

Introduction

Quality of Service (QoS) is one of the essential features of modern networks. In the past decades, the usage and popularity of computer networks increased with a high demand for quality of service. Software-defined networking (SDN) have been around for a couple of years now. They are participating in many academic papers because of their research potential. Content Content delivery network (CDN) can be implemented to cache data to bring data temporarily close to end users and to lower the load from Origin servers. We create promising research potential by combining it with SDN. Nevertheless, like many areas, adding Machine Learning (ML) opens a new dimension of possibilities altogether. Automation of networks is a concept called Zero-touch networks. The idea is to combine ML into decision-making parts of networks and let the network make its own decisions.

In this work, we discuss the state of the art in our chosen research area—innovative networking approaches combined with ML and CDN and the idea behind zero-touch networks. More precisely, load detection and load balancing in SDN networks that utilize CDN.

We aim to create a new algorithm and a management system for SDN and CDN utilizing ML. The primary focus is on dynamically distributing content requests and optimize bandwidth utlization thereby lessen bottlenecks on potential critical network nodes with the addition of a management system.

The following chapter will be the state of the in our chosen research before we where we state our problem. A clear definition of the area we want to increase the QoS is crucial. Mainly network data gathering, storage, and its proper utilization. Consequently, the definition of the hypothesis of our goal is in this thesis. The short introduction in the problematic areas sub-parts gives us an idea and a more unobstructed view of issues in networking that result in low QoS. The main focus is enhancing network utilization, monitoring and data handling to get closer to a true zero-touch network. Following the proposal and implementation, we will describe our testing scenarios and our applied ML models, which contributed vastly to enhancing the quality of SDN.

1.1 Zero touch networks solutions

One vision of the networks of tomorrow is zero-touch networks. Theoretically, they should work autonomously without needing human personal intervention[4]. It can correct itself, update, autoscale, and maintain itself based on the help of neural networks Virtualized Network Function (VNF) and SDN. With the combination of VNF and SDN orchestration, quality of service is significantly raised. As for the VNF functionality, it provides excellent help for SDN in the traffic handling and processing as it is responsible for the stateful parts of the network and the SDN for the stateless, for example, switches[6].

1.1.1 Training and accuracy

To create an efficient model, we need large amounts of high-quality training data to ensure greater accuracy in models[1]. Training and re-training of models require data and computational power. Models need to be robust and create real-time decisions; otherwise, the QoS would be decreased[1].

1.1.2 Deploying machine learning

Alongside ML solutions, data analytics play a crucial role in predicting traffic demand[1]. Zero touch network relies heavily on ML approaches to enhance QoS. Solutions enhancing SDN with ML need to be trusted and transparent, as well as the data source on which the ML solutions are trained on[1]. Explaining models is also crucial for understanding the decision process of the solutions. The earlier shallow ML approaches were easy to understand as their complexity is not as high as a deep neural network with multiple layers [1]. Therefore, it can pose as a black box in which the decision-making process is complicated.

1.2 Preemptive content caching

Optimal usage of content caching tools such as CDN is critical to ensure the QoS from the content owner's and client's view. Nowadays, it is not enough to have content cached on the nearest CDN cache node to the client, but the network load plays a crucial role in the right CDN cache node selection[2]. For example, when the nearest surrogate server to the client has a high content delivery time. Due to the lack of available bandwidth in the network node connecting the CDN cache into the network[8]. It results in the need to search another CDN cache node. To achieve a new surrogate selection, we need to select an efficient redirection method for the selection process. One popular method is a request router and Hypertext

Transfer Protocol (HTTP) redirection. As seen in the image 1.1, when a request for some content is created, the Uniform Resource Locator (URL) refers to the request router instead of a CDN cache node. The request router processes the request and, via HTTP message 302, redirects the client to a CDN cache node.[7].

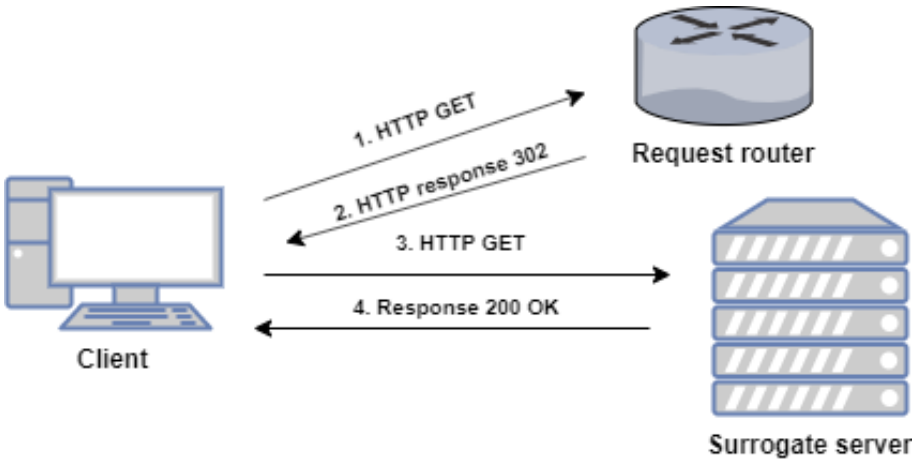

Figure 1.1: HTTP redirection[7]

Chapter 2

Problem statement

2.1 Definition of the problematic area

The goal of the thesis is to create a new SDN network management system to dynamically optimize content delivery and increase the utilization of network resources. The new network management system uses novel algorithms in data handling. This is a step towards a zero-touch network; therefore, we discuss it later on. As the previous chapter mentioned, machine learning is making its way into networking, which has contributed to creating a long leap from the legacy approaches of the last years. However, the solutions we found are solving just one issue or a specific use case. A problem with creating solutions in specific areas is that they are incompatible and ultimately need better implementation and real capabilities. Also, the data logging currently in SDN is at a reasonably low level. SDN provides the ability to extract network information, and sFlow can extract network information and create basic low-level statistics. With the overall utilization of this data, not to mention logs from CDN nodes, we can collect essential data that needs to be utilized correctly.

Incorrect utilization also goes for attempts to create zero-touch networks, which could theoretically handle maintenance, rerouting, and adaptive bandwidth utilization. As stated in the previous chapter, the idea behind zero-touch networks is complete autonomy. The software side of the networking is planned, scheduled, and executed by programs in the network. Some of our biggest issues are efficient network utilization on single controller SDN, where we want to stabilize load-balancing issues between network nodes caused by CDN content requests. A Zero touch network that could enhance service quality has yet to be created.

Chapter 3

Solution proposal

3.1 Proposed Architecture

In Figure 3.1, we can see the proposed architecture of our solution. The main idea is to build a low-cost dynamic infrastructure around an SDN network. Enabling data and log transfer to various custom APIs, which can consume/forward or process the input in a required format and as fast as possible. We will need APIs that can efficiently handle data and perform ML operations. Connectability to storage services is needed to archive gathered data and enhance the network even more in the future.

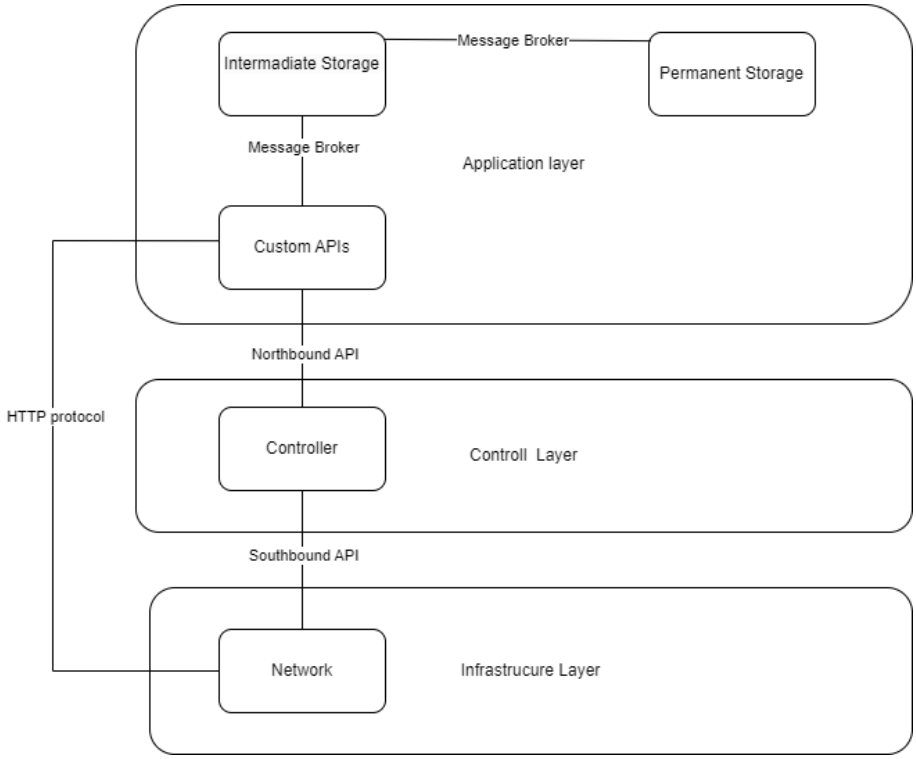

Figure 3.1: High level Architecture

3.1.1 Network

The network itself needs to be able to communicate behind its SDN borders, and it needs to communicate with external APIs and, of course, the controller. Dynamic access is key because we will do multiple testing scenarios in the network. We will also need to scale the network between testing scenarios with minimal interaction with the network configuration. We will need to emulate multiple surrogate servers for testing purposes. It will be deployed in the network part of the infrastructure layer as seen in Figure 3.1. A possible solution for the dynamic distribution of content requests is the application of a request router. We can outsource the decision-making onto the application level via HTTP requests carrying the original content request payload while returning the chosen surrogate server. Thereby, the client receives only a redirect message to a specific surrogate server available for

him for that specific request.

3.1.2 Controller

The controller needs to be maintained, which has support to this day. The ability to connect to different external APIs is needed. Otherwise, we will need the current standard functions of SDN controllers.

3.1.3 APIs

APIs will have multiple diverse functions; therefore, we must carefully consider the programming language or languages we choose. Another requirement is to choose a language that enables the application of ML models. The APIs will use servers as AI modules, data transfer, transformation services, monitoring, and other services. HTTP protocols will be used for all communication.

3.1.4 Storage

We will need two types of data storage. Firstly, short-term storage will be an intermediary and can handle large data loads during deployment. It also needs to be able to handle multiple active connections, both sending and pulling data. The retention of this storage can be a few because of fail-safe reasons. On the other hand, we will need permanent storage to store the data from the intermediate storage for longer. We chose a relational database for our solution because we can utilize complex queries on the stored data for analytical purposes. A relational database is sufficient for our solution. However, a non-relational database is still being determined.

3.2 Prepared test cases

Following the definition of the architecture, we need to define the test cases that will be tested in the network. These cases will help us to achieve our milestones. Utilizing the given network and neural networks as efficiently as possible.

As we aim to create a more efficient way to request routing with ML, we need test cases that will provide us with valuable training data. We must create a testing application to run on the hosts in the SDN. The application needs to create content requests to create a bottleneck between the surrogate servers and the nearest switch connecting it to the remaining network devices.

Expected results

During testing, we expect the content request distribution to be uneven. This means the surrogate servers with a larger portion of the host nearest to them will have more content requests; therefore, we expect the available bandwidth to be low or nonexistent. After implementing our solution, we expect the content request distribution to be evenly divided among all the surrogate servers, or there will be groups of surrogate servers with even load distribution regardless of content requests.

Chapter 4

Solution verification

4.1 Prototype Implementation

We implemented an SDN/CDN sandbox. We needed two virtual machines where the sandbox was running separately and the request router, which handles CDN redirection separately. Regarding external network connectivity, the virtual machines were interconnected with each order, and we also created a connection to the machine where the controller, Application Programming Interface (API)s, and the dockerized part will run. Following this step, we set the SDN controller applications up and ensure the connectivity of the different user environments.

4.1.1 Testing enviroment

We used a computer with the Windows operating system where the controller, controller applications, dockerized Kafka, PostgreSQL, and the ML modules were running. To ease communication, we emulated the request router and SDN sandbox on the same machine on two separate virtual machines running Ubuntu with a bridged network adapter.

4.1.2 Testing scenarios

We decided to emulate extensive network usage of clients downloading different content types. The speed and file size were scaled down due to hardware limitations. We scaled megabytes to kilobytes regarding network speed and file size to stay as close as possible to a functioning network. We created multiple testing runs on the same topology with the same setup. Content and network requests were randomized in a matter of the time the content was requested, and the content type and size were randomized.

- Data collection from the network during testing runs
- Heavy network load with content demands from clients
- Data logging

To ensure the correctness of our solution, we manually evaluated all the decisions created and the effect they would have on the network.

4.1.3 Networking setup

The mininet address range was 10.11.0.1-45. We had designated Internet Protocol (IP) addresses for the request router, origin server, surrogates, and client devices. The controller was outside of the virtual machines where the testing took place. Therefore, in the network setup, we connected the network layer via the remote controller connection to the local host of the computer where the VMs were running. As the mininet network and request router were on a different VM, we created a bridged adapter where we statically assigned an IP address for the request router and port via which it connected to mininet. The application layer applications and storage service were running on the same device as the controller.

4.2 Colected Data

We had two data collection channels. The primary data source was collected in a Ryu application, which selects surrogate servers in a CSV format for content delivery. The secondary data source was via REST API, which Ryu controller can expose so requests can be made and response payloads are stored in JSON format. As for the form of the data, we collected the Received and transmitted bytes. We also increased the size of the network testing environment in two stages. The reason was that with the originally proposed network, which contained only a few end devices, the initial analysis showed promising results.

First increment We increased the number of surrogate servers to eight. With the increase, we wanted to monitor the surrogate selection and change in network utilization, leaving the original number of end devices the same.

Second increment The final scaling of the network was done by increasing the number of end devices to 35 and 49 virtual switches. Gaining more versatile data for further analysis and processing.

Chapter 5

Result evaluation

5.1 Data storage and containerization

To prove the concept of temporal storage and provide the environment for data analysis, We utilized a dockerized Kafka client with Zookeeper. We used an existing image to prove that we can use existing images from [9]. The basis of the image was Kafka, Zookeeper, KSQL, and Grafana for load monitoring of its utilization. The setup of the topics was a single partition with a single broker since it was sufficient for our testing purposes. It is entirely possible to use multiple partitions and brokers. If we wanted to deploy the solution, we would emphasize parallel applications, which would need more computing resources. As for the message setup, we used a simple JSON structure. We did not implement a key structure for messages since the structure was not dependent on data point order. Other parameters, such as topic retention, were defaulted for seven days. We implemented Python-based consumers and producers, which performed data transformations and storage into PostgreSQL tables for more long-lasting data storage.

Source	Target	Port	Transmitted bytes	Received bytes	Bandwidth
2	10.11.0.9	2	753739.0	701164.0	254968.727273
4	10.11.0.11	6	621942.0	25437.0	9690.285714
4	10.11.0.12	7	15252.0	10155.0	3868.571429
4	10.11.0.13	8	94894.0	7110.0	2708.571429
5	10.11.0.14	1	91667.0	4762.0	1814.095238

Table 5.1: Topology data sample

5.1.1 Topology data

We utilized NetworkX to search for the selection of shortest routing paths and created a rest API from which we collected data in a JSON format, which represented the network in a Digraph format. Enable us to replicate the network link utilization as it was monitored regularly. The data is processed into a format as seen in Table 5.1 for the content request data. We logged information about the request itself, the requester IP address, surrogates sorted by location, the available bandwidth of those surrogates, the request file name, and the time when it was requested.

We decided to classify the network load per our defined bandwidth to create the machine learning-based network load classification requirements. The measurements are broken down into three groups: Link Not Used, Link Used, and Heavy Load.

- Link Not Used if the bandwidth was under the five quantiles of measurement group.
- Heavy load if the bandwidth was over 85 quantile of the measurement group
- Link Used if the bandwidth was between the other two classes.

Client IP address	Filepath	Size of the requested file
10.11.0.45	/movies/movie170.avi	53
10.11.0.45	/movies/movie158.avi	79
10.11.0.38	/videos/video102.avi	23
10.11.0.45	/movies/movie117.avi	56
10.11.0.38	/movies/movie251.avi	63

Table 5.2: Collected content request sample

5.1.2 Surrogate server data

Regarding the data needed for surrogate selection, we decided to log more data for analytical purposes and important features for machine learning. Data such as the IP address of the end users (clients in the network), the file the surrogate server delivered, and the size and size class can be used in further analysis to enhance content type caching on surrogate servers to avoid cache misses. In this case, the attributes of size and size class of files only enhance the network. We can see some example data logs in Table 5.2 in the format and form they were collected during testing.

As for the more relevant data for the machine learning process, we chose to collect three relevant features of the network:

- Client source Internet Protocol version 4 (IPV4) address used in the network
- all the surrogate servers orderer from nearest to farthest from the client based on shortest path algorithm
- The bandwidth of the network link before the surrogate CDN server, which represents the load on them

The bandwidth calculation was the same as with the topology data. We displayed only the first 3 in the example below for extensive reasons. The surrogates were logged in the form of IP addresses in the network and the bandwidth in bytes as

Surrogate 1	Surrogate 2	Surrogate 3	Bandwidth 1	Bandwidth 2	Bandwidth 3
10.11.0.10	10.11.0.9	10.11.0.8	1453.369863	1407.783784	1498.057143
10.11.0.10	10.11.0.9	10.11.0.8	1453.369863	1407.783784	1498.057143
10.11.0.3	10.11.0.6	10.11.0.2	1478.888889	1438.197183	1428.056338
10.11.0.10	10.11.0.9	10.11.0.8	1453.369863	1407.783784	1498.057143
10.11.0.3	10.11.0.6	10.11.0.2	1478.888889	1438.197183	1428.056338

Table 5.3: Collected bandwidth and surrogate information sample

can be seen in Table 5.3.

5.2 Apllication of machine learning

We decided to use the same machine learning approach for the prediction of network load and for surrogate selection, which was multiclass classification. The traditional approach does not provide the flexibility and the ability to learn as efficiently as neural networks.

5.2.1 Network load classification

We applied several shallow classification algorithms to verify our data extraction, cleaning, and possible network load. As a best practice when applying machine learning to a new use case dictates, we needed to test the classification potential of the data and our common goal. Therefore, we chose neighbor, Gaussian, Regression, and tree-based classification algorithms. As we can see, the result shows great potential in our approach, and at this point, we had high hopes for achieving our goal. The settings on the classifiers were the default ones, as we only wanted to test the potential of the data. Our foresight in the data preparation paid off, and the normalized data were a good choice since the GaussianNB classifier also returned according to the accuracy metric above 98%

With this newly gained confidence, we tried to apply a more complex approach

Layer	**Architecture**
Input	8
Hidden Layer 1 (IL)	$8 \rightarrow 64$
Hidden Layer 2 (HL1)	$64 \rightarrow 128$
Hidden Layer 3 (HL2)	$128 \rightarrow 64$
Output Layer (OL)	$64 \rightarrow 3$
Activation Function	ReLU (for all)

Table 5.4: Neural network architecture for network load classification

to tackle this issue. We have chosen a relatively small dataset with around 1 million data points. This amount is sufficient to create a neural network with a few hidden layers. Therefore, the approach is more commonly known as the application of deep learning. To create a network that has the potential to serve accurate real-time predictions about network status, we implemented and tested Recurrent Neural Network (RNN) and Artificial Neural Network (ANN) networks. After initial testing, we chose to rely on an ANN network, and we attempted to optimize the hyperparameters of the network. From the knowledge gained from the previous network, we chose to use four linear layers with backpropagation. The complexity of this use case was relatively smaller than 5.2.2. According to that, we had a shorter learning period of just under five thousand iterations.

We also added the same overfitting prevention methods for this network and stopped the training after the tolerance limit was reached, which was 16. In this figure, we can see that it seems that the model hit the local minimum when training, and for a few epochs, it stopped improving, but in the end, we got the loss function value down to 0.07.

Our final neural network can be seen in Table 5.4. It contains four hidden layers, where two are hidden. The activation functions used were Rectified Linear Unit (ReLU) in all layers.

5.2.2 Surrogate server selection

We already knew that the application of machine learning as in our previous work [2] has potential. Therefore, we went straight for deep learning approaches, which could enhance the network load balancing even more. Our chosen metric was to minimize bandwidth utilization on the nodes connecting the surrogate servers to the rest of the network. To be more precise, the goal of our dynamic surrogate selection approach was to select the surrogate server with lower bandwidth utilization than the nearest surrogate server. Mitigating the risk of a bottleneck near the surrogate servers. We utilized deep neural networks to achieve our enhancement. To be more precise, an Artificial neural network is more commonly known as ANN. ANN was the final result of our rigorous tests. We hoped that recurrent neural networks or networks such as Long short-term memory networks would provide the best results; however, ANN took the spotlight regarding accuracy. The main challenge after selecting the ANN approach was to select the appropriate layers, activation functions, batch sizes, and other hyperparameters for the network's training process.

The best criterion was the cross entropy loss commonly used in linear problems. To ensure a stable learning process, we also needed to include an optimizer for our model, such as Stochastic gradient descent or the many variations of the so-called Adaptive moment estimation (Adam) optimizer family. After testing, we chose the learning rate 0.0001 and Adam's newest variation, the Adaptive moment estimationl with weight decay (AdamW). It is based on Decoupled Weight Decay Regularization[5]. In simple words, it is a type of L2 regularisation method. The final step in the training process was to choose the training batch sizes. With the batch size, we determined the amount of training data points flowing into the training per iteration in a given epoch. The number was sixteen, resulting

in the best training results as seen in the wandb Figures ?? and ??; the loss decreased after the half-million iteration mark. That was the moment we applied the mentioned changes to the model. Resulting in a final training loss of 0.35 and 99 percent accuracy on the testing data.

Being satisfied with the network, we had to add contingencies to prevent an overfitting model at the end of the training process. Therefore, we added two methods to achieve it. The first method was the application of dropout, which temporarily removed some nodes in the network layers. It is a common method to see how the network copes with information loss. The second method was the addition of early stopping in the training cycle. The tolerance was set to fifteen, which means if the network itself does not improve in loss during 15 epochs, the training process is stopped, and we contribute to overfitting prevention.

5.2.3 Result verification

To summarize our results, we achieved network load classification, a crucial step for future network enhancements with the help of machine learning. Furthermore, we can apply deep learning to load balance content delivery. for easier uderstanding of the surrogate servers we will call them CDN1-8. To validate our approach, look at Figure 5.1, where the current approach of nearest surrogate selection is applied if we can see that the bandwidth utilization on most of the nearest CDN surrogates ranges from 400 to 800 KB from with the limit being 1MB, except surrogate CDN7. The reason was that it had the fewest end devices in its proximity; therefore, it had a relatively small number of content requests than the others.

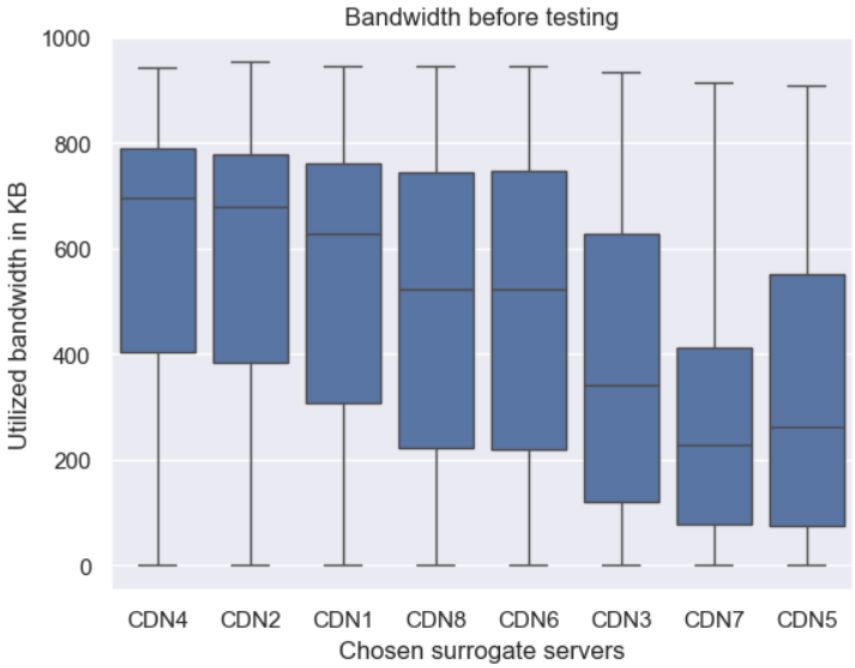

Figure 5.1: Chosen surrogate CDN server before testing

On the other hand, if we take a close look at Figure 5.2, we can see that we decreased the overall utilization of surrogates when selecting the or, more specifically, we chose the surrogate servers, which had visibly lower loads at the time when the selection process was taking place. Therefore, there was a considerable increase in selection efficiency and content delivery.

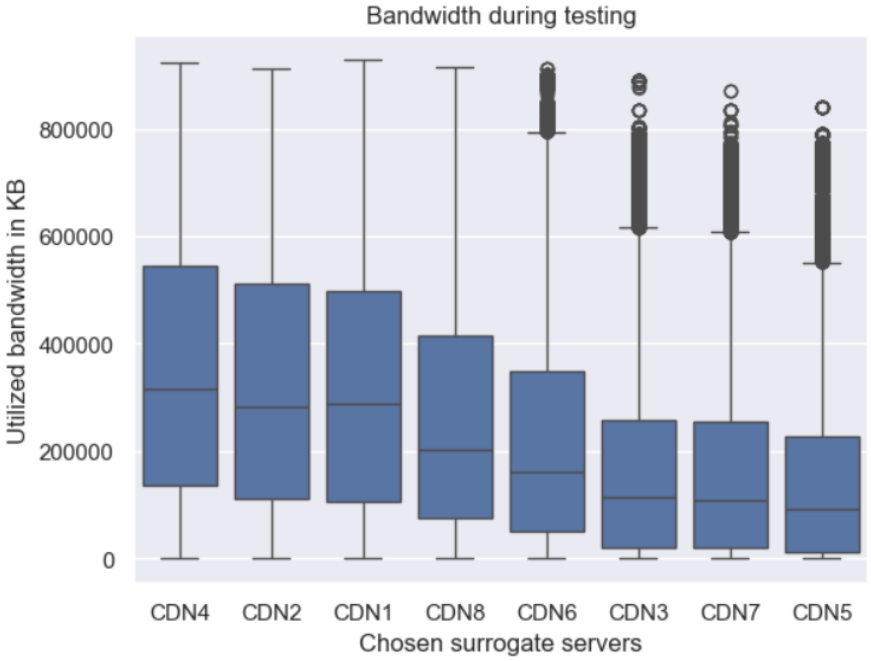

Figure 5.2: Chosen surrogate CDN server after testing

For a clearer picture, we plotted a comparison of the host with the id 35 with the nearest surrogate selection and our dynamic approach in figure 5.3. We took the sample from the middle of the testing scenario. We can see that the nearest surrogate to the client has mostly higher utilization than the dynamic selection process. The overlay between the lines on the line plot is when the nearest surrogate was selected in our approach, which means its utilization was lower than that of other surrogates.

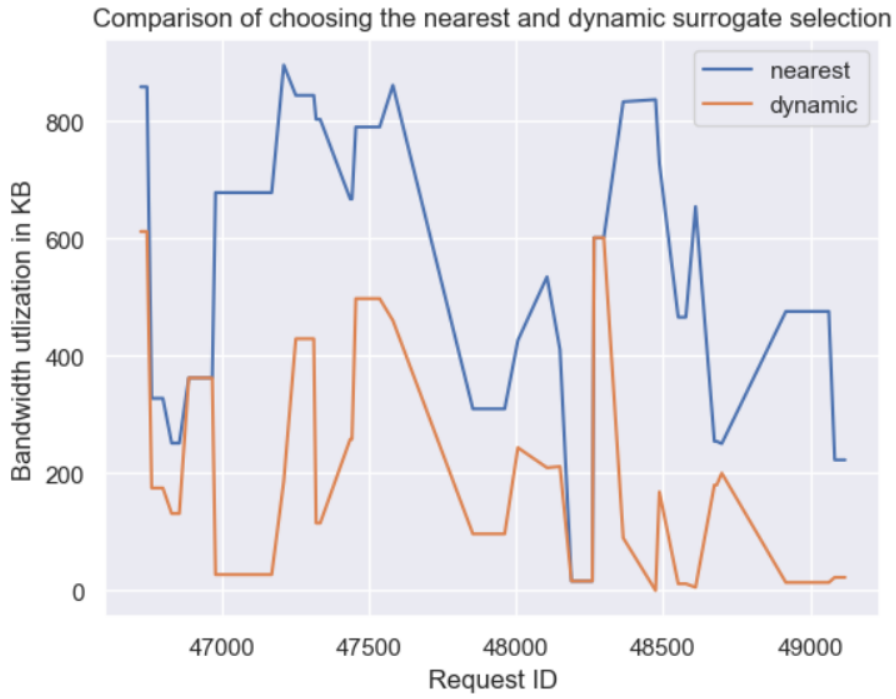

Figure 5.3: Nearest surrogate selection and our approach bandwidth utilization comparison

To be more precise, if we look at the count plot in figure 5.4, we can see that the CDN3 is the nearest surrogate sever for the client with the id 35 in the testbed. This means that when applying the nearest surrogate selection approach, all requests from the client go CDN3. On the other hand, with our dynamic selection process, the client's requests were served from four different surrogate servers during testing. This means that the model successfully selected a surrogate server with lower bandwidth utilization, and when the nearest was selected, it had low utilization.

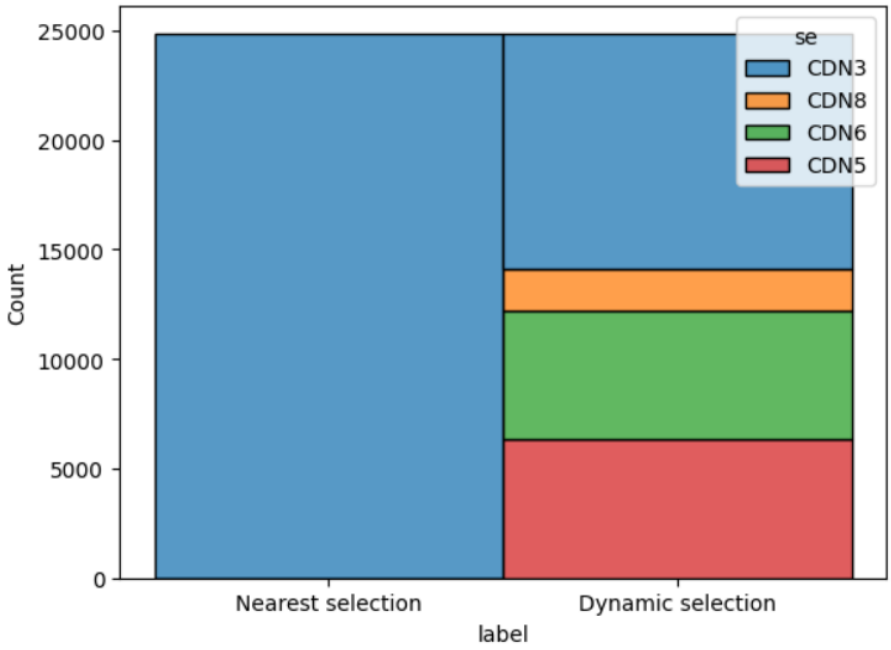

Figure 5.4: Redirection distribution in the tested scenarios

With the comparison of the results in Figure 5.1, where the load of the surrogate servers is not evenly distributed, and Figure 5.5, where we can see the result comparison of our proposed solution and the nearest surrogate selection approach, we can see that the distribution of the surrogates is even within three subgroups reasoning the number of clients in the proximity of the surrogate serves. The subgroups are:

- CDN4, CDN2 and CDN1
- CDN8 and CDN6
- CDN3, CDN7 and CDN5

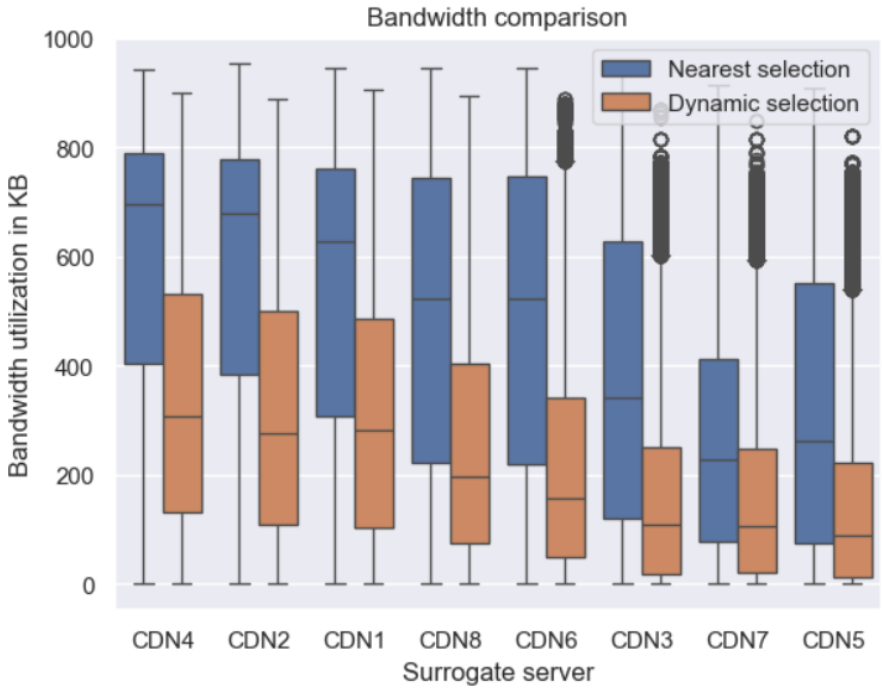

Figure 5.5: Nearest surrogate selection and our approach bandwidth utilization comparison

Each of the surrogate subgroups is almost even distributed between surrogate serves. Therefore, we successfully distributed content delivery between multiple surrogate servers where multiple servers were considered. Therefore, we have proven our proposed solution valid and decreased the load on the critical nodes of the network.

5.2.4 Neural network verification

Both our neural networks were tested via the Z3 theorem verifier which is a Satisfiability Modulo Theories (SMT) solvers[3], also used in the formal verification of neural networks. The input is the neural network and constraints to be verified. Our neural networks successfully passed the given constrains.

Chapter 6

Conclusion

With our solution, we could dynamically distribute the load off surrogate servers, optimizing bandwidth utilization. We also created a new network management system to utilize ML and different monitoring tools to ensure network quality. Therefore, we achieved a big step towards a zero touch network.

To be more precise, we focused on the progress of QoS in the mentioned areas. Most notably, the utilization of ML to increase QoS in areas like bandwidth utilization, strategical content delivery, network resource classification, network cost optimization, network scaling, and existing ideas of the structure of zero touch networks. Afterward, we defined our problematic fields related to the QoS in computer networks. We chose the approach of a zero-touch network proposition which could have the potential to improve the QoS in our chosen area significantly.

‘

References

[1] Imran Ashraf et al. "Zero touch networks to realize virtualization: Opportunities, challenges, and future prospects". In: *IEEE Network* 36.6 (2022), pp. 251–259.

[2] Peter Bakonyi, Tomáš Boros, and Ivan Kotuliak. "Classification Based Load Balancing in Content Delivery Networks". In: *2020 43rd International Conference on Telecommunications and Signal Processing (TSP)*. IEEE. 2020, pp. 621–626.

[3] Leonardo De Moura and Nikolaj Bjørner. "Z3: An efficient SMT solver". In: *International conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer. 2008, pp. 337–340.

[4] David M Gutierrez-Estevez et al. "Artificial intelligence for elastic management and orchestration of 5G networks". In: *IEEE Wireless Communications* 26.5 (2019), pp. 134–141.

[5] Ilya Loshchilov and Frank Hutter. *Decoupled Weight Decay Regularization*. 2019. arXiv: `1711.05101 [cs.LG]`.

[6] Jon Matias et al. "Toward an SDN-enabled NFV architecture". In: *IEEE Communications Magazine* 53.4 (2015), pp. 187–193.

[7] Gang Peng. "CDN: Content distribution network". In: *arXiv preprint cs/0411069* (2004).

[8] Jihoon Sung et al. “Efficient content replacement in wireless content delivery network with cooperative caching”. In: *2016 15th IEEE International Conference on Machine Learning and Applications (ICMLA)*. IEEE. 2016, pp. 547–552.

[9] Marcin Zabłocki. *Setup apache kafka in Docker on windows*. 2017. URL: `https://zablo.net/blog/post/setup-apache-kafka-in-docker-on-windows/`.

Appendix A

Publications

- P. Bakonyi and I. Kotuliak, "Clustered collaborative c-learning," 2022 IEEE Zooming Innovation in Consumer Technologies Conference (ZINC), Novi Sad, Serbia, 2022, pp. 63-66, doi: 10.1109/ZINC55034.2022.9840740.
- P. Bakonyi, M. Vančo, I. Kotuliak (2022) INNOVATIVE DOCKERIZED WEBSITE BUILDER, ICERI2022 Proceedings, pp. 3013-3016.
- P. Bakonyi, T. Boros and I. Kotuliak, "Classification Based Load Balancing in Content Delivery Networks," 2020 43rd International Conference on Telecommunications and Signal Processing (TSP), Milan, Italy, 2020, pp. 621-626, doi: 10.1109/TSP49548.2020.9163470.