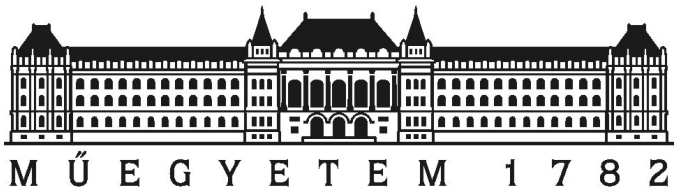

Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Hálózati Rendszerek és Szolgáltatások Tanszék

Federated learning driven network anomaly detection

Simkó Máté
O3BMRX

Konzulens:
Dr. Pekár Adrián

2022. november 1.

Kivonat

A mesterséges intelligencia alapú hálózati anomália detekció már egy ideje létezik. Az alkalmazásában a közelmúltban elért előrelépések ellenére az adathalmazok megosztása még mindig kihívást jelent, ami adatvédelmi aggályokat vet fel. Jelen kutatás e probléma kezelésével foglalkozik. Ezt a federált tanuláson keresztül teszi, amely lehetőséget biztosít arra, hogy több kliens egyszerre tanítsa a modellt, és profitáljon a folyamatosan javuló modell előnyeiből. Míg a kliensek a szoftvert a hálózati anomáliák detektálására használják, addig az adataik csak lokálisan kerülnek tárolásra. Miután elegendő adatot gyűjtöttek, megkezdik a modell tanítását, és feltöltik az új súlyokat, amelyeket átlagolásra kerülnek a többi felhasználó által beküldött súlyokkal. Minden feltöltés után az új, átlagolt súlyok egy teszthalmazon kiértékelésre kerülnek, a modell romlásának elkerülése érdekében. Ennek eredményeképpen a modellfejlesztéshez nincs szükség adatmegosztásra a különböző helyszíneken futó kliensek között. A prototípus értékelése kimutatta, hogy a helyi tanítás és a federált átlagolás életképes megközelítések az adatvédelem fenntartására.

Abstract

Artificial intelligence-based network anomaly detection has been around for a while. Despite recent advances in its application, meaningful sharing of datasets is still a challenge, raising data privacy-related concerns. This present research tackles addressing this problem. It does so via federated learning, which enables multiple clients to train the model simultaneously and benefit from the ever-improving model. While the clients use the software to detect network anomalies, their data is collected only locally. After sufficient data is collected, they start to train the model and upload the new weights, which will be averaged with the weights sent in by other users. After every upload, the new averaged weights will be evaluated on a test set to avoid model deterioration. As a result, no data sharing across anomaly detectors run at various sites is required for model development. The prototype implementation evaluation revealed that local training and federated averaging are viable approaches to maintaining data privacy and protection.

Acknowledgement

I would like to thank my supervisor Dr. Adrián Pekár for his professional and constructive comments throughout the course of the presented research. I also want to thank my colleagues, Makara Laszlo Arpad and Marcell Szoby, for their collaboration. Last but not least, my thank goes to my partner and family for their continuous support.

Contents

List of Figures

List of Tables

List of Abbreviations

FL Federated Learning

DL Distributed Learning

IoT Internet of Things

VPN Virtual Private Network

LSTM Long Short-Term Memory

GRU Gated Recurrent Units

CSV Comma-Separated Values

CUDA Compute Unified Device Architecture

GPU Graphics Processing Unit

DHCP Dynamic Host Configuration Protocol

SSL Secure Sockets Layer

TLS Transport Layer Security

HTTP HyperText Transfer Protocol

Introduction

Network anomalies have been around since the dawn of the internet. A prime example of these anomalies is cyber-attacks which are posing an ever-growing threat to both everyday users and companies. Methods to detect these attacks have been the topic of many papers in the past. The first attempts to detect these anomalies were rule-based or made use of different heuristic strategies. After machine learning got more popular it became the standard for anomaly detection. This change in methodology brought a massive improvement in the results. Anomaly detection became more reliable and scalable.

Despite the advantages offered by machine learning, it faces great difficulty obtaining large volumes of suitable training data. Publicly available datasets are hard to find, and there are only a very limited amount of them. Also, private data is becoming more and more protected by governments. It makes it even more burdensome to collect high-quality data. This research addresses this problem by creating a federated learning platform.

Federated learning (FL) makes it possible to train the model with data generated by any computer without actually sharing data with anyone. To make it work there are two required components, a server, and a client. The server stores the models and the weights, while the client uses these models and collects data locally. After enough data has been collected, the client will be ready to train. The training takes place on the client machine keeping the data private. Another advantage of this approach is that, at the same time, multiple clients can be performing training. In this case, the resulting weights will be sent to the server, where they will be averaged. This model is beneficial for both parties because the client is able to use an ever-improving model, while the creator of the model could improve its performance without the need to gather more data.

The framework proposed by this article is customizable and is capable of handling multiple models. For example, the server could store one model for anomaly detection and another for application classification. The clients could choose which of these models they would like to use, and the preprocessing will change accordingly to meet their needs.

The rest of this work is organized as follows. Chapter 1 provides a brief overview of network traffic anomaly detection, challenges in the face of network measurement, and federated learning. Chapter 2 describes the methodology for FL-driven network traffic flow anomaly detection, including the used frameworks and tools utilized to achieve this goal and the developed architecture. Chapter 3 then presents the achieved results in various experimental setups, followed by a discussion and prospects for future work. Finally, Chapter 4 concludes this work.

Chapter 1

Background

1.1 Network Traffic Flow Anomaly Detection

Network traffic consists of packets. Two packets are in the same flow if their IP addresses, port numbers, and protocol are the same. One flow contains all data from the beginning to the end of a particular connection between two computers. A flow contains multiple useful pieces of information that lets machine learning models classify them easily.

In a computer network, there is normal behavior, and everything abnormal is an anomaly. Despite this very general definition, anomalies can originate from multiple causes, for example, DDoS attacks, Port scanning, etc. These deviant behaviors are impossible to detect by humans because of the sheer volume of the network data, so multiple solutions were created using traditional reasoning based and also machine learning approaches.

1.2 Challenges

The protection of private data is a huge concern in modern societies. Since the EU introduced GDPR, data collection has become more difficult and expensive. This created a new demand for novel solutions in machine learning. This research focused on keeping the user's data on their own machines. This way, the training of the model is distributed, and every user trains on their own data. This is not only great from a privacy perspective but also reduces the resources required for the server.

The fact that everyone can use the federated learning platform is creating data quality problems. As the server does not own the data, it can not validate its quality. This is why the server validates the uploaded and averaged weights with a validation dataset. If there is a significant drop in the scores, the server will discard the new weights.

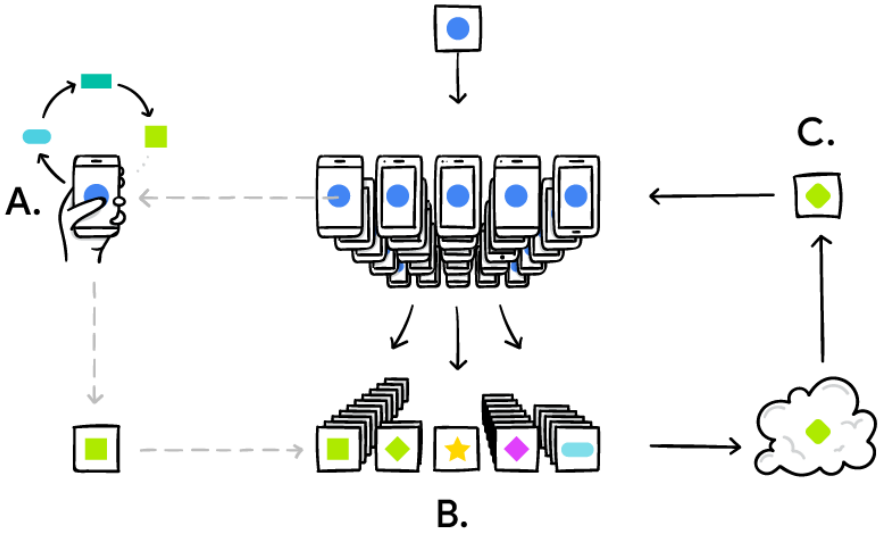

Figure 1.1: Federated Learning [1]

1.3 Federated Learning

Federated Learning is a new technique in machine learning which can take advantage of decentralized data collection and decentralized computing. Using the traditional method, data scientists needed to obtain large amounts of data to create a good model. More and more data is required to create better models, and with more data, the demand for computing power also increases. This trend has been going on for decades, and this is making this process more expensive. Also, it is challenging to obtain large amounts of data. Publicly accessible datasets are too small, and the companies who are collecting large amounts of data are keeping them for their own research purposes. Federated Learning not only makes training models cheaper but also gives smaller, innovative companies opportunities. A high-level view of this training strategy can be seen in Figure 1.1. It shows how the process works, which starts with the clients separately generating data and training the model. These weights will be averaged by the server, and a new, improved model will be created. This advanced model will be downloaded by all clients, so they can all benefit from the better-performing new model. This process repeats each time new data is gathered by a client in a cyclical nature.

Horizontal federated learning is one of the most popular federated learning types. It is suitable if all of the data on the client machines have the same characteristics, as shown in Figure 1.2. The architecture proposed by this research also falls into this category. It uses NFStream to export the network traffic flows. This ensures that every client will produce data with the same basic structure. Using this method it is possible to train supervised models.

In the federated learning architecture, there are N clients $\{C_1, \ldots, C_N\}$, and all of them have collected enough data to begin training the model. After each of them finishes the training, their weights will be averaged and validated on the server. This validation will produce an accuracy

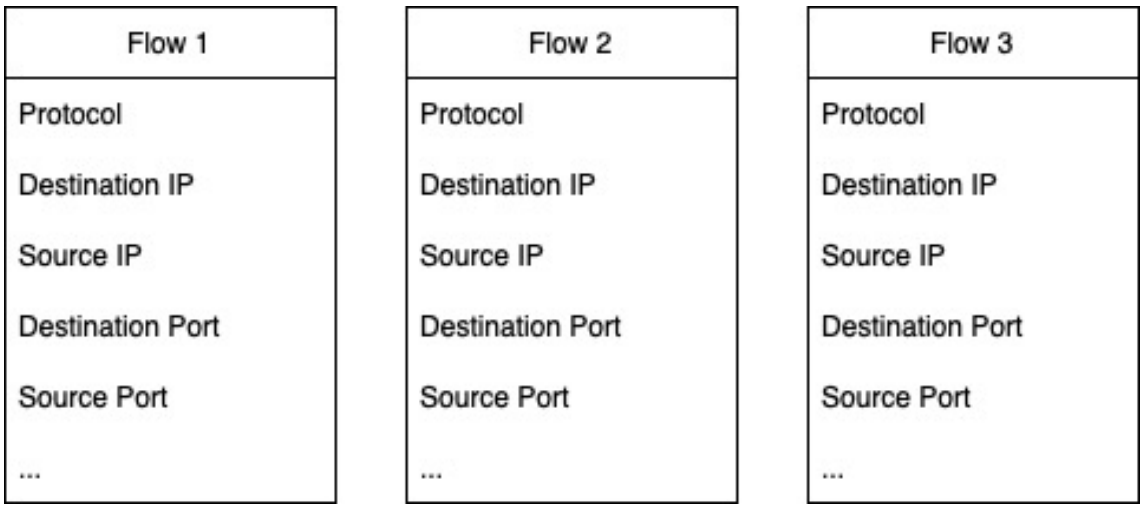

Figure 1.2: Possible data structure Horizontal Federated Learning

score A_{FED}. Traditionally all of the data would be stored in one set and trained on one computer. This training can also be validated and would produce an accuracy score $A_{TRADITIONAL}$. The effectiveness of federated learning is described by its accuracy loss, which can be calculated in the following manner:
$Q = |A_{TRADITIONAL} - A_{FED}|$ where Q is the accuracy-loss of the federated training.

1.4 Federated Learning vs. Distributed Learning

Distributed Learning (DL) is a machine learning technique that enables training a model with multiple nodes at the same time. The data is cut into smaller subsets, and every node trains on a portion of the dataset. This offers a method to carry out trainings that otherwise would not be viable because the complexity outmatches the limits of a single computer.

While distributed learning focuses on training models on an enormous dataset, federated learning does not necessarily work with comparable volumes of data. In the case of DL, the data is already obtained, while in FL, it is constantly created by the clients. Federated learning does not enable clients to share data with each other, while distributed learning has no restriction on this. This is because of the different use cases. An FL client can be any computer anywhere in the world, while DL nodes are usually located in the exact physical location and owned by the same company. Federated learning offers a solution for difficulties regarding data, such as privacy issues or obtaining data over extended periods of time, whereas distributed learning seeks to overcome computational limitations.

1.5 Related Work

Mothukuri *et al.* [2] proposed a similar framework, but this paper's main focus was on Internet of Things (IoT) use. Each IoT device functions as a client node, which collects data

and eventually trains the model. The model weights are also uploaded to a server and averaged. To detect the network flow anomalies Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRU) neural networks were used.

Sater *et al.* [3] proposed a framework in which they aim to lower the training time of the long short-time memory (LSTM) model by proposing a new privacy-by-design federated learning model using a stacked LSTM model. The main goal of their research is to optimize energy usage in a smart building environment. The primary function of their model also uses a centralized server node and sensors as clients.

Pei *et al.* [4] proposed a framework in which the clients collect and train data, and the server aggregates the gradients and creates a new model. However, the local models are personalized by fine-tuning. They used an LSTM-autoencoder model.

Gupta *et al.* [5] proposed a hierarchical FL that enables collaboration among various organizations using different levels of aggregation. Their main goal was to lower the delay in healthcare applications, such as in a digital twin environment, by detecting anomalies between the sensors and the cloudlet server.

Cui *et al.* [6] integrated blockchain into their proposal and designed a decentralized IoT anomaly detection framework, which does not use a central server for storing the model weights. Instead, they are stored in a P2P network. They used DP-GAN models for anomaly detection.

Agrawal *et al.* [7] aimed at presenting an extensive and exhaustive review on the use of FL in intrusion detection systems. While they do not provide a framework or an exact solution to the anomaly detection problem, they introduce ideas and real-world statistics about the anomalies that this paper deals with.

Huong *et al.* [8] proposed a solution that uses federated learning to reduce bandwidth consumption in the link between edges and the cloud. In their solution, they are using a variational autoencoder (VAE) and LSTM. The training and data gathering happens in the edge devices, and then the "Cloud" computes the weight of the aggregated global model.

Anaissi *et al.* [9] propose a federated learning approach to overcome the challenges of machine learning mechanisms in centralized and distributed settings. They named this algorithm PC-FedAvg. Their structure is similar to the one that is the focus of this paper. They take steps to solve problems in federated learning environments, such as the continuous communication between several clients and the server, by mitigating the case when weights from low-performing clients are included.

Zhao *et al.* [10] propose a multi-task deep-neural network in federated learning (MT-DNN-FL) to perform an anomaly detection task, Virtual Private Network (VPN) traffic recognition task, and traffic classification task simultaneously. Their goal is similar to this paper's purpose. They seek an anomaly detection solution in a federated learning environment while classifying the network flow as well.

Chapter 2

Methodology

2.1 Flow Measurement Framework

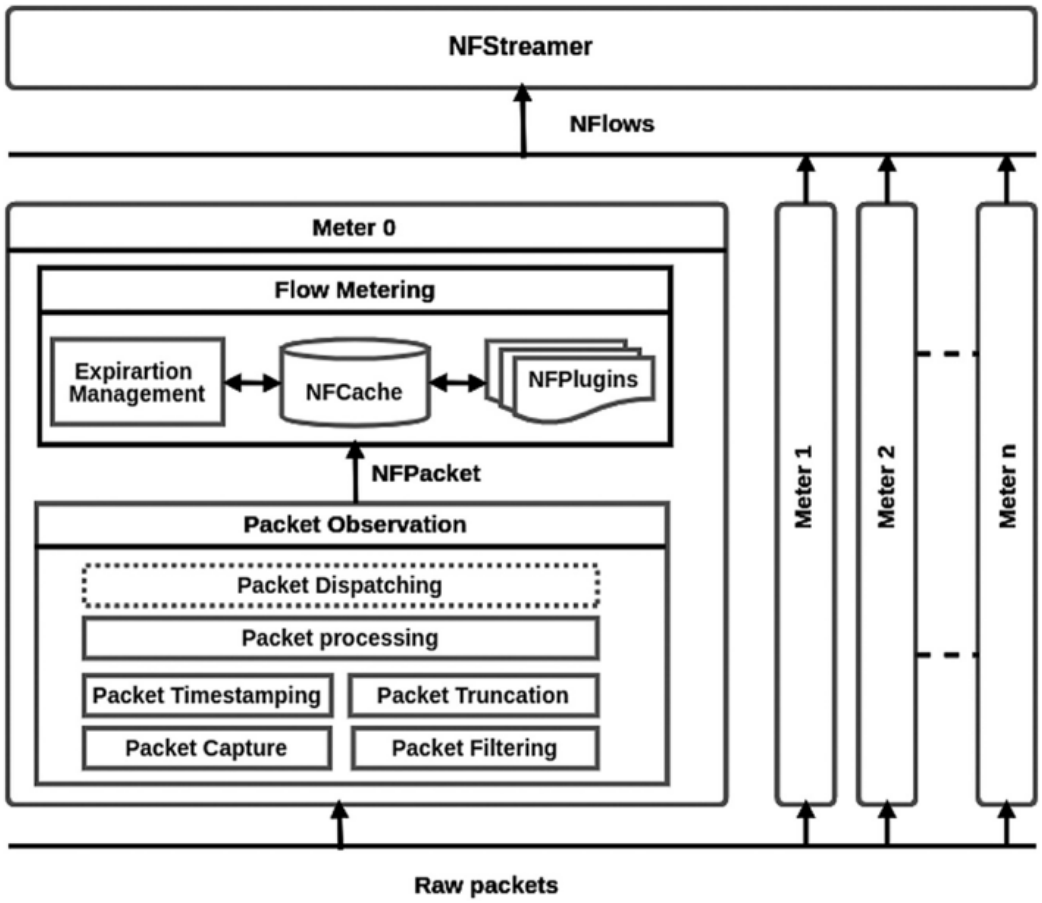

Figure 2.1: NFStream [11]

NFStream is a platform-independent, open-source network tool that is capable of sniffing the network traffic on a selected interface and exporting the captured flows. It calculates a wild variety of statistics from the flow, which can be used for machine learning purposes. With the help of the nDPI library, NFStream is also capable of determining the application type of the flow.

As it is shown in Figure 2.1 NFStream consists of multiple essential modules in a meter. Packet capture is a low-level functionality that is implemented using the libpcap library. With packet truncation, lower CPU load and bus bandwidth is required. The packet filtering module enables developers to use packet filtering expressions based on Berkeley Packet Filter syntax.

The flow metering layer is responsible for the flow management logic. NFCache organizes the packets into flows. The expiration management module has multiple flow termination logics. If any of these consider the flow terminated, the modul ends it. NFPlugins are user-defined extensions for NFStream. They allow developers to create their own methods, which will be called at the creation, update, and expiration of the flow. This feature can come in handy when combining NFStream with machine learning.

2.2 Flower

Flower is an open-source federated learning framework that provides the base functionalities to developers and enables them to create their own solutions building on the components supplied by it. It provides tools for handling multiple clients, sending the training results to the server, and validating them.

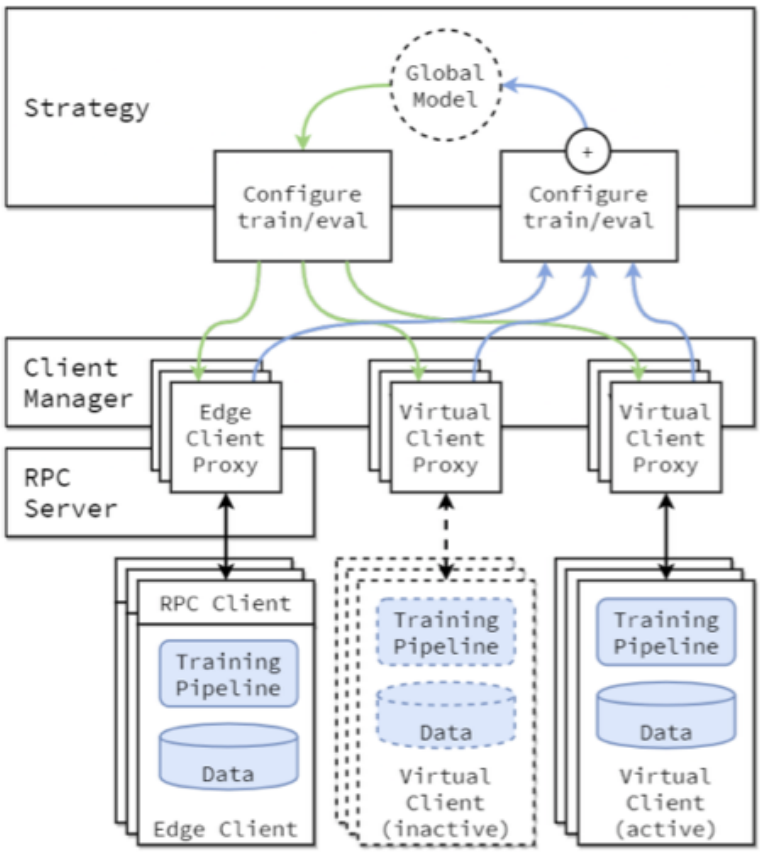

Figure 2.2: Flower architecture [12]

Figure 2.2 displays the underlying architecture of the framework, showcasing its key components. The architecture clearly divides the responsibilities of the server and the clients. While

the training process is managed by the server, the data is kept locally. The most important part of the architecture are the strategies. There are premade strategies such as FedAvg or FedProx, but custom strategies can be implemented too. These are responsible for the training and evaluation process. The client manager keeps track of the clients and communicates with them through Flower Protocol messages. The FL loop is the component orchestrating the federated learning process based on the strategy.

2.3 Architecture

The framework proposed by this article has two main components. One for the server and one for the client side. The client is able to choose a model. It was designed to be easily customizable. The models are stored on the server, and the clients can choose to use any of them. Also, new models can be added to the server whenever. However, currently only keras models work with the framework. The structure of the framework proposed in this paper can be seen in Figure 2.3.

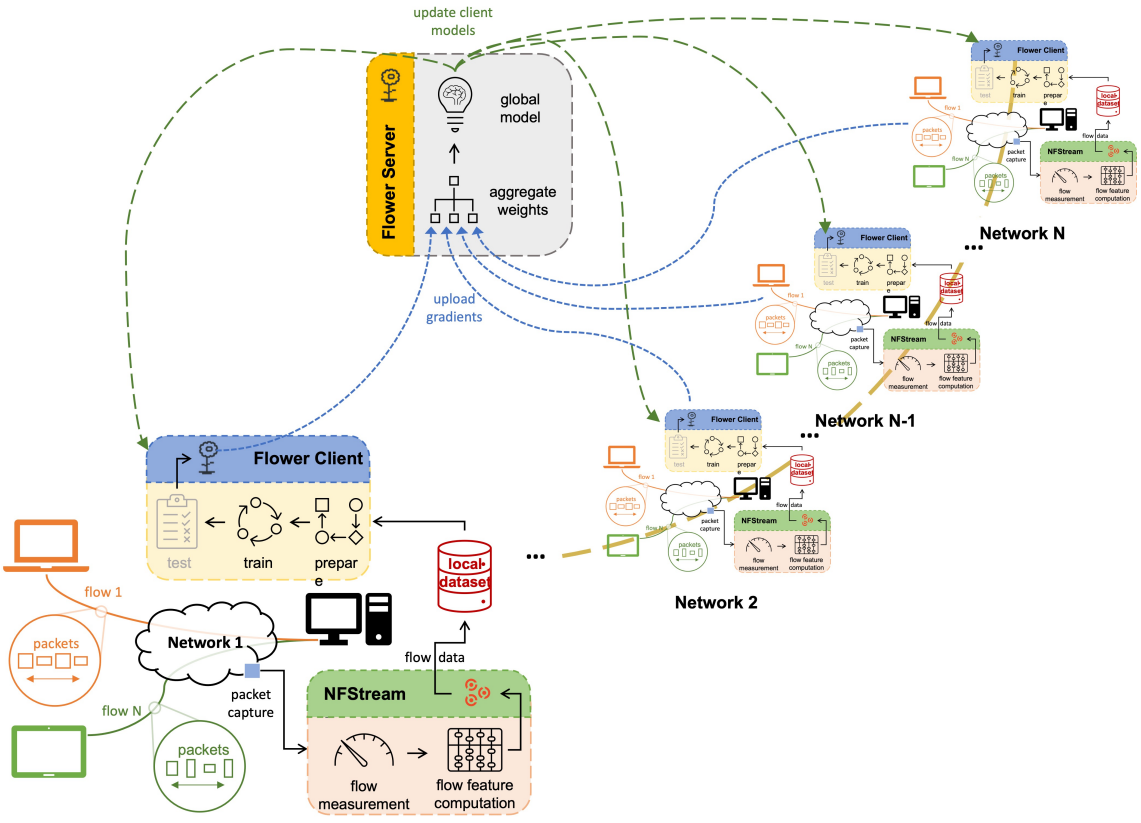

Figure 2.3: Federated learning architecture

2.3.1 Server-side API

The server-side component implements a Flower server that can handle the clients and manage the federated learning processes based on the strategy. The server is capable of storing mul-

tiple models. Each model has its own python file with the keras implementation, and all models can have their own preprocess steps. The model weights are stored in a NoSQL database. The training process is initiated by the client by stating that it has enough data. After that, the server waits for other clients to be able to train too. If enough clients are ready, the server notifies the clients to start training. When the clients finish the training, their weights will be uploaded to the server and averaged.

2.3.2 Client-side API

The client-side component implements a Flower client that lets the user connect to the server. The user can select a model from the server, and in case they have not been loaded yet, the model with the weights and the preprocessing steps will be loaded. After the initialization, an NFStream instance will be loaded. NFStream constantly siffes the flows, and the model can give a prediction after every terminated flow. These flows are also stored in a local file. After enough data has been collected, the client lets the server know that it is ready to train. Before the training, the client must download the newest model weights if they have not downloaded them already. The training takes place on the client's computer and is managed by the server. The locally generated data will never be sent to the server. Only the weights will be uploaded.

Network

The network card of a computer consists of multiple buffers. They are managed by the kernel. In the case of almost every computer, the packets are not processed in real-time and are often dropped. However, there are some specialized network modules that are capable of handling the packets in real-time. The best predictions can be achieved with these hardware modules.

NFStream

NFStram constantly sniffs the incoming packets and monitors active flows. With the help of an NFPlugin, when a flow terminates, the model will give a prediction.

Local Dataset

The data generated by the clients will always stay on the same computer. Every time a flow terminates, an NFPlugin will be called that appends the new data to a CSV file. After the training threshold is met and enough clients are ready, the training will start, and the weights will be sent to the server. After this, all data will be deleted.

2.3.3 Flower Strategy

A custom flower strategy has been implemented. After the clients finish the training, the uploaded weights will be averaged and saved to a NoSQL database. This way, all the previous versions of the model will be stored on the server, but the clients always have to download the most recent one.

2.3.4 Anomaly Detection Model

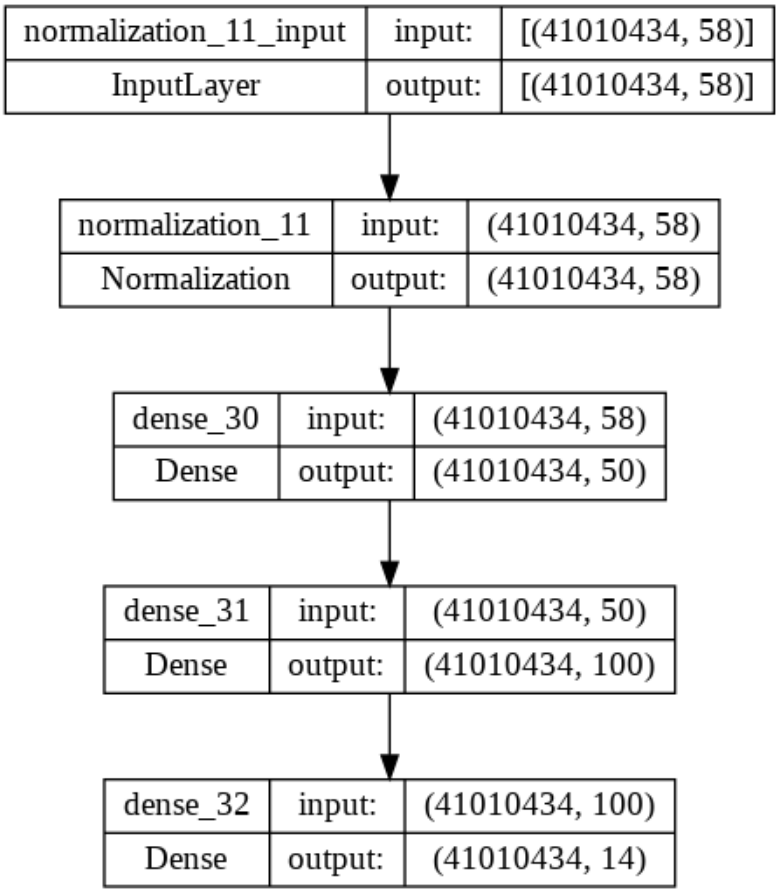

Figure 2.4: Layers of the dense model

To test the framework a simple model was created. The model contains a normalizer and three dense layers, as can be seen in Figure 2.4. To be able to determine the differences between traditional and federated training, the model was first trained on the whole dataset, with a 20% validation split. The validation dataset was stratified, which means the distribution of the anomalies in the validation set matches the distribution in the training dataset. The model was trained for 15 epochs with the batch size of 1024. The batching was achieved by implementing a keras sequences for managing the data. The first layer of the model is a normalization layer that transforms the numerical features of the data. It does not change the dimension of the data. The second layer is a fully connected dense layer with 50 units. The next layer is also a fully connected dense layer which widens the network by taking an input of (,50) and having an out-

put of (,100). Both of these layers use relu function as the activation. The final layer is also a dense layer with the same number of units as there are classes in the dataset. In this case, it is 14: 13 different types of anomalies and one category for not anomalous behavior. This layer uses softmax function as the activation because this layer is used for the classification.

Chapter 3

Evaluation

3.1 Dataset

For evaluating the results, the CIC-DDoS2019 dataset was used. It contains more than 50 million network traffic flows. The dataset contains simulated attacks against the network. The dataset is available in both PCAP and CSV formats. However, the labels are included only in the CSV files. The federated learning framework proposed by this paper was designed to work with NFStream, but the CSV files were created by CICFlowMeter. Because of this, a new CSV file had to be created from the PCAPs with NFStream.

3.1.1 Data Preprocessing

NFStream is capable of extracting flows from PCAP files. CIC included the precise times of each anomaly in the form of start and end timestamps, so the labeling could be done using the data based on the flows bidirectional_first_seen and bidirectional_last_seen features. The dataset contains multiple data types such as numerical, categorical, binary, and text data. Most of the text data (e.g., IP/MAC addresses) was dropped from the dataset because they do not carry much information regarding anomaly detection. The categorical data has been one-hot-encoded so the model can process it. For federated learning, the data was split into ten chunks. One of these is the validation subset, which has the same label distribution as the original dataset. The rest of the data was split chronologically to model real word use.

3.1.2 Characteristics

The 50 million flows contained in this dataset belong to 14 categories. The distribution of the anomalies in the dataset is listed in Table 3.1.

Each flow has 39 features extracted to the CSV file, which are the following:

- id: Unique identifier of the flow.

Table 3.1: Number of records in each class

Anomaly type	No. of flows
PortMap	1 268
NetBIOS	3 240 741
LDAP	1 773 081
MSSQL	5 751 368
UDP	8 469 048
UDP-Lag	880 927
SYN	4 942 597
NTP	1 159
DNS	1 186
SNMP	1 737 334
SSDP	1 052 640
WebDDoS	2 869 290
TFTP	10 614 585
Not anomaly	9 294 942

- expiration_id: This feature can have three values and represents the reason of the expiration.
- src_ip: The IP address of the source machine.
- src_mac: The MAC address of the source machine.
- src_oui: The Organizationally Unique Identifier of the source machine.
- src_port: The source port.
- dst_ip: The IP address of the destination machine.
- dst_mac: The MAC address of the destination machine.
- dst_oui: The Organizationally Unique Identifier of the destination machine.
- dst_port: The destination port.
- protocol: The communication protocol, represented as a number.
- ip_version: IPv4 or IPv6.
- vlan_id: VLAN identifier.
- bidirectional_first_seen_ms: The timestamp of the arrival of the first bidirectional packet in the flow in milliseconds.
- bidirectional_last_seen_ms: The timestamp of the arrival of the last bidirectional packet in the flow in milliseconds.

- bidirectional_duration_ms: The elapsed time between the first and last bidirectional packets in milliseconds.
- bidirectional_packets: The number of bidirectional packets sent in the flow.
- src2dst_bytes: The number of bytes sent from the source to the server in the flow.
- src2dst_first_seen_ms: The timestamp of the arrival of the first packet sent from the source to the server in the flow in milliseconds.
- src2dst_last_seen_ms: The timestamp of the arrival of the last packet sent from the source to the server in the flow in milliseconds.
- src2dst_duration_ms: The elapsed time between the first and last packets sent from the source to the server in milliseconds.
- src2dst_packets: The number of packets sent from the source to the server in the flow.
- src2dst_bytes: The number of bytes sent from the source to the server in the flow.
- dst2src_bytes: The number of bytes sent from the server to the source in the flow.
- dst2src_first_seen_ms: The timestamp of the arrival of the first packet sent from the server to the source in the flow in milliseconds.
- dst2src_last_seen_ms: The timestamp of the arrival of the last packet sent from the server to the source in the flow in milliseconds.
- dst2src_duration_ms: The elapsed time between the first and last packets sent from the server to the source in milliseconds.
- dst2src_packets: The number of packets sent from the server to the source sent in the flow.
- dst2src_bytes: The number of bytes sent from the server to the source in the flow.
- tunnel_id: Identifier of the tunnel.
- application_name: The name of the layer 7 application.
- application_category_name: The category of the layer 7 application.
- application_is_guessed: Marks whether the application type is guessed or not.
- application_confidence: Represents how trustable the data regarding the application is.
- requested_server_name: The name of the requested server.
- client_fingerprint: Dynamic Host Configuration Protocol (DHCP) fingerprint.

- server_fingerprint: Server fingerprint based on Secure Sockets Layer (SSL) or Transport Layer Security (TLS).
- user_agent: HyperText Transfer Protocol (HTTP) user agent.
- content_type: HTTP content type.

3.1.3 Evaluation Metrics

A confusion matrix is a usual tool to evaluate the performance of a model. It can be created for every classification model, and it contains four values, which are the following:

- True positive (TP): The number of cases where the model predicted the sample to be a member of a given class, and it actually belongs to it.
- False positive (FP): The number of cases where the model predicted the sample to be a member of a given class, but it actually does not belong to it.
- True negative (TN): The number of cases where the model predicted the sample not being a member of a given class, and it actually does not belong to it.
- False negative (FN): The number of cases where the model predicted the sample not being a member of a given class, but it actually belongs to it.

The performance of the models was determined by these four metrics:

- Accuracy: It measures how often the model gives the correct prediction.

  $Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$
- Precision: It measures how many of the predictions for a single class were actually correct. It shows how trustable the prediction of the model is.

  $Precision = \frac{TP}{TP+FP}$
- Recall: It measures how many samples of a single class were correctly predicted. It shows how good the model is at recognizing the given class.

  $Recall = \frac{TP}{TP+FN}$
- F1 score: It is the harmonic mean of the precision and recall.

  $F1\ score = \frac{2*(Recall*Precision)}{Recall+Precision}$

3.2 Results

3.2.1 Experiment Setup

For this research paper, three experiments were performed. Firstly the rudimentary model was trained on the whole dataset in a traditional manner. After this, two different topologies were tested using the federated learning framework on the same network. This section details the environment and resources used as well as the execution of the experiments.

For the traditional training, Microsoft Azure was used as a training platform. The code ran in a jupyter notebook. The machine used for the experience had a Compute Unified Device Architecture (CUDA) compatible Graphics Processing Unit (GPU) to accelerate the training.

The experiments with federated learning ran on four different computers. One of them functioned as a server, and the others as clients. The dataset was split into ten parts. One was used for evaluation and one for the initial training on the server. The remaining eight were all allocated to a client. So overall, there were eight clients and a server in the experiment simulating real-world race conditions.

To evaluate the solution multiple tests were run. First, the model was trained on the whole dataset. This could be used as a reference for the efficiency of the federated training in further experiments.

The federated training was done in two different ways. In the first test, the clients trained in pairs. After two clients were connected to the server, they started to train, and their weights were averaged. In the second test, the clients trained in triplets.

3.2.2 Traditional Training

For the traditional training, the whole dataset was used with a 20% validation split. When separating the data into training and validation sets, stratification was used. This means that the distribution of the anomalies in the training and the validation sets were approximately the same. The model was trained for 15 epochs with a batch size of 1024. The results of this experiment are listed in Table 3.2.

Table 3.2: Metrics of the traditional training

Metric	Score
Accuracy	58,58%
Precision	55,14%
Recall	59,49%
F1 score	54,30%

3.2.3 Federated Training in Pairs

In this experiment, the federated training happened in pairs, meaning at any given moment, two clients are performing training. The batch size was set to 1024, and the number of epochs was 15. The training ran three times, and the one with the best results was saved by the server. In the experiment, eight clients participated, so four models were trained by the clients. The changes in accuracy, precision, recall, and f1 score can be seen in Figures 3.1 and 3.2.

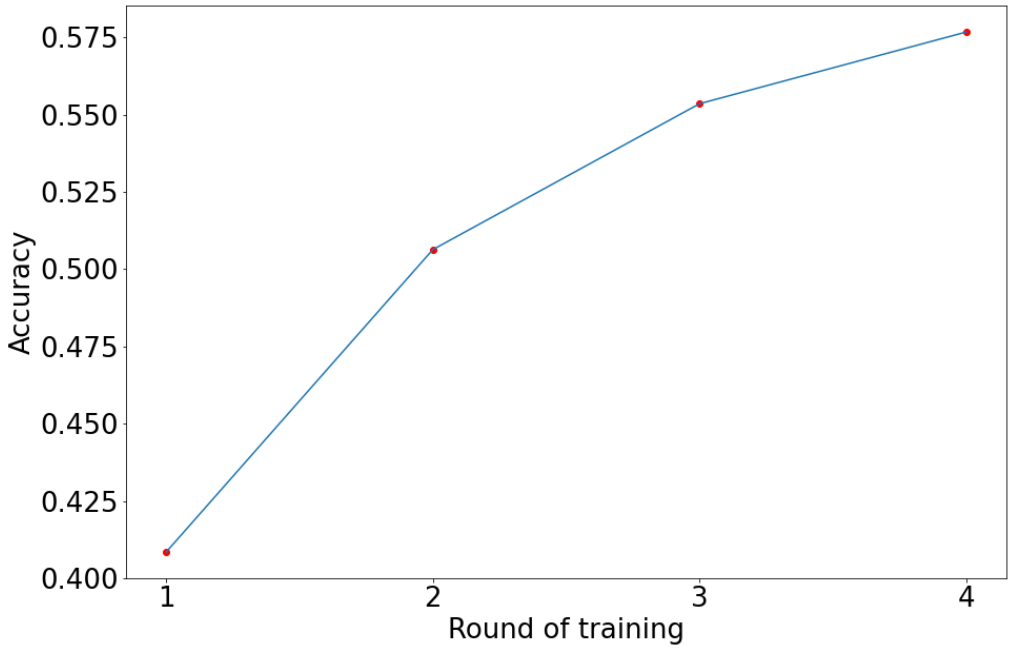

Figure 3.1: Federated training in pairs accuracy

3.2.4 Federated Training in Triplets

In this experiment, the federated training happened in triplets. The batch size was set to 1024, and the number of epochs was 15. The training ran three times, and the one with the best results was saved by the server. In the experience, eight clients participated. In the first two federated training, 3-3 clients took part, and the last training was done with the remaining two clients. The changes in accuracy, precision, recall, and f1 score can be seen on Figures 3.3 and 3.4.

3.3 Discussion

The experiment has not produced a powerful anomaly detection model, but the creation of such is outside of this paper's scope, which is to simply demonstrate the effectiveness of the federated learning framework. The relatively low scores are partly due to the simple model and

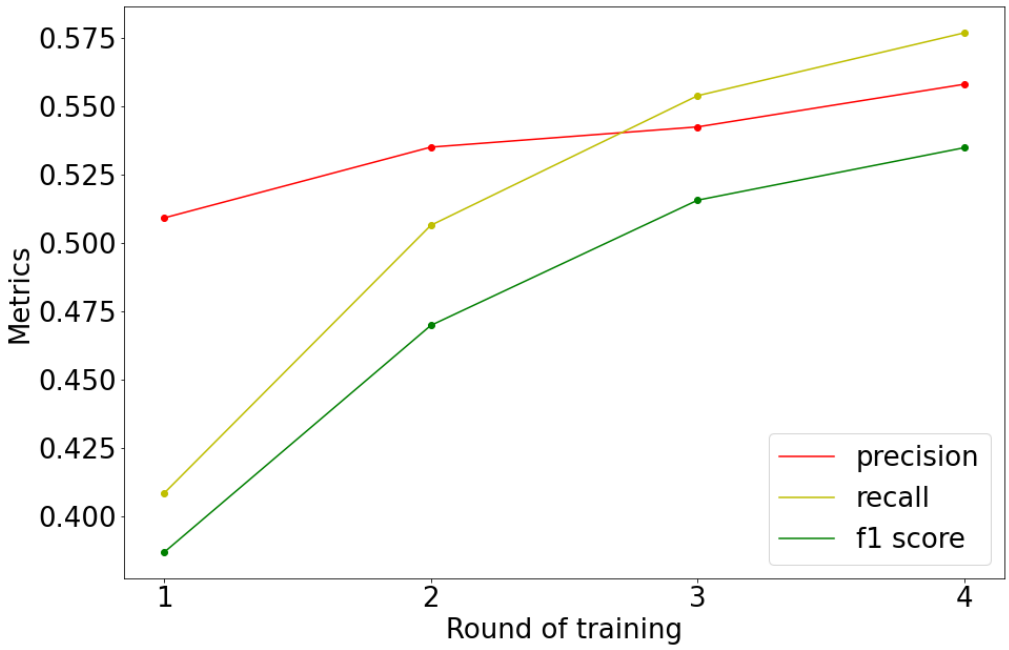

Figure 3.2: Federated training in pairs

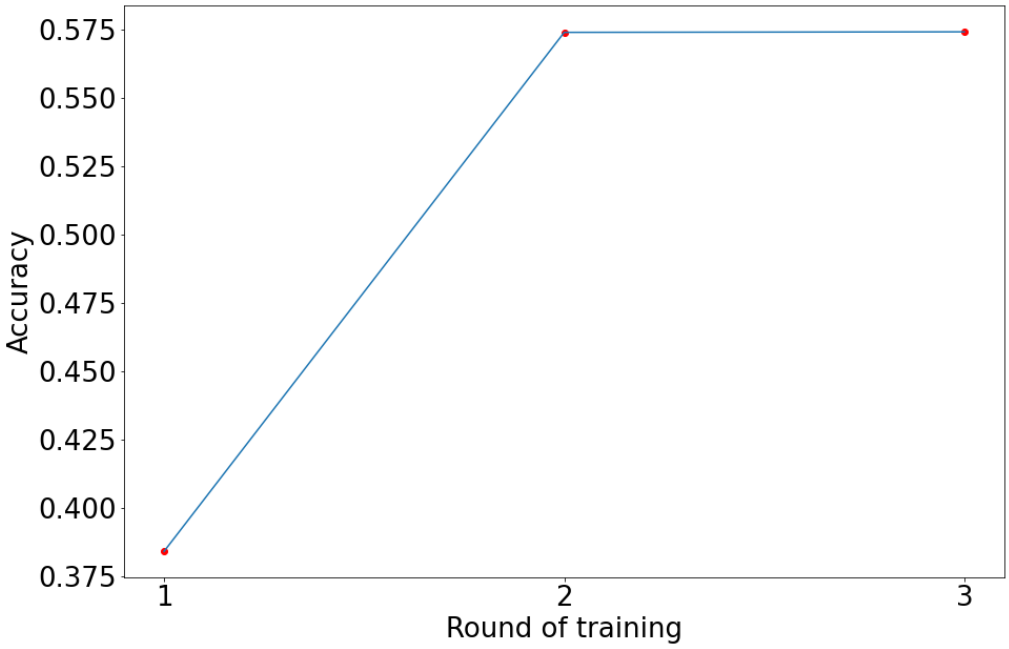

Figure 3.3: Federated training in triplets accuracy

the imbalanced dataset as well as the lack of computing capacity, although in the evaluation of the federated learning framework, it does not matter much. It is worth noting that even this simple model vastly outperforms random prediction. However, with more training rounds, smaller batch sizes, and more epochs, these scores could be increased.

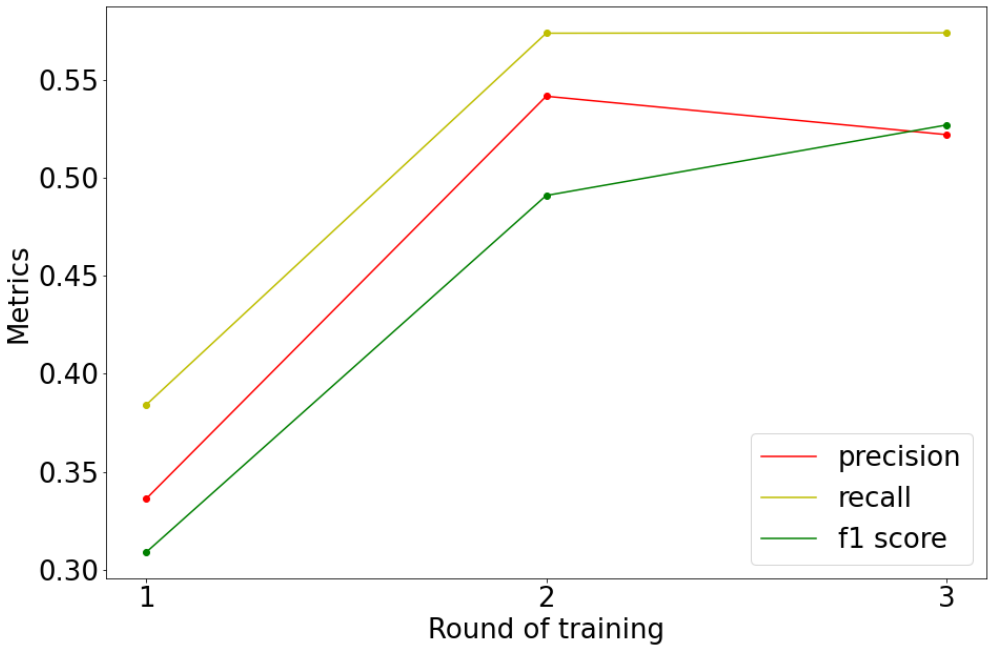

Figure 3.4: Federated training in triplets metrics

The results of the first experiment are used as a reference for the following experiments. With its results, it is possible to determine how well the federated learning framework performs. The training runs with the exact same parameters as the further experiments.

In the second experiment, where the federated training took place with two clients, the results show that there was growth after every training, as can be seen in Figures 3.1 and 3.2. The rise in accuracy slowed down training after training. The amount of increase had approximately halved after each training. The growth of the recall showed very similar behavior to the accuracy. However, the precision did not change a lot. There was a slight rise in the score after every training, but it was not significant.

The third experiment was similar to the previous one, the only difference being that the number of clients varied. Three trainings took place in this experiment. In the first two rounds, 3-3 clients were training, and in the third round, the remaining two. As it can be seen in Figure 3.3, in this case, the accuracy rose to the same level as it did at the end of the previous experiment. However, it did it after the second round of training, and it did not improve further after the third round. As it can be seen in Figure 3.4 the recall behaved very similarly. It rose to the same level as it did in the previous experiment, but it also achieved this after the second round and did not improve further after the third training. The precision, unlike in the second experiment, did show improvement after the second round. However, it did decrease after the third one.

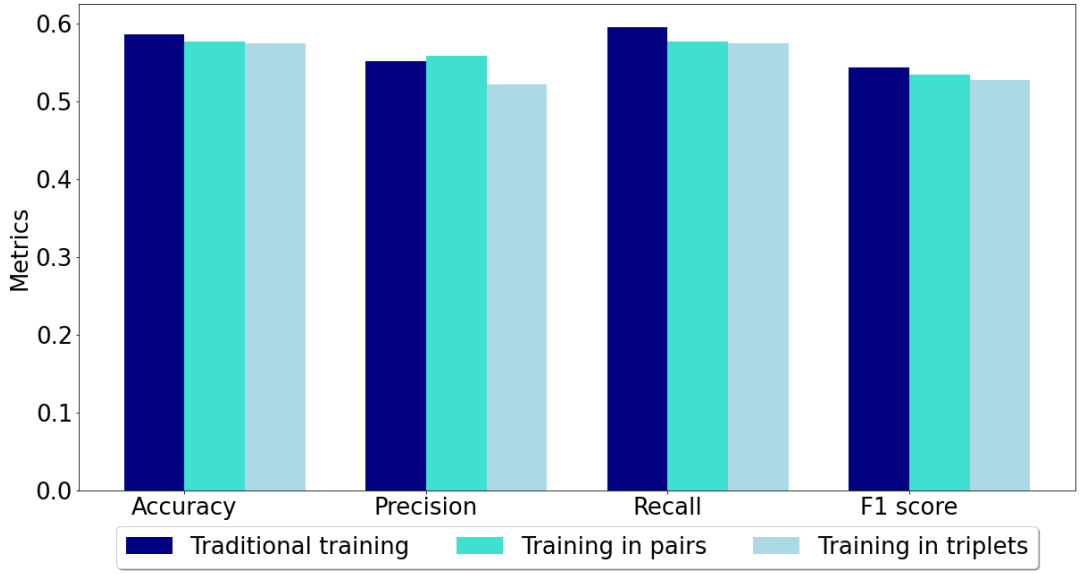

Figure 3.5: Comparison between the results of the different experiments

3.4 Future Work

This topic could be further investigated by testing the framework's behavior with multiple different kinds of models. The framework is designed the accommodate a multitude of different networks that have not yet been tested but should be effective in different situations.

There is also room for further research in testing the framework's scalability by increasing the number of simultaneous clients. As this paper showed, the number of training rounds and clients have an effect on the model's performance. Therefore, it is worth exploring how training over an extended period with large amounts of data from many different clients would affect the efficiency of the central model.

One key area that should be explored in the future is the vulnerabilities inherent to federated learning and how well the framework deals with them. As the whole point of this learning strategy is that anyone can train on their private data without sharing it, the server can not perform checks on whether the data used is trustable. Malicious actors could try to exploit this by intentionally sending damaging weights to the server in order to corrupt the model and decrease its performance. The framework's reaction to such attacks requires further investigation.

Chapter 4

Conclusion

In conclusion, this paper has demonstrated that federated learning is a viable solution for anomaly detection models. These results clearly show that there is only a very slight loss of performance when using federated learning compared to traditional training. In addition to all of this, only 10% of the data was used for the initial training of the model on the server. The rest of the data was located on the client machines, and only their weights were shared with the server. This is especially promising because this means that a continuously growing and changing pool of clients can improve the model, all while keeping their data private and benefiting from its increasingly precise predictions.

Bibliography

[1] Brendan McMahan and Daniel Ramage, *Federated Learning: Collaborative Machine Learning without Centralized Training Data*, 2017. [Online]. Available: https://ai.googleblog.com/2017/04/federated-learning-collaborative.html.

[2] V. Mothukuri, P. Khare, R. Parizi, S. Pouriyeh, A. Dehghantanha, and G. Srivastava, "Federated-learning-based anomaly detection for iot security attacks," *IEEE Internet of Things Journal*, vol. 9, pp. 2327–4662, May 2021.

[3] R. A. Sater and A. B. Hamza, *A federated learning approach to anomaly detection in smart buildings*, 2020. [Online]. Available: https://arxiv.org/abs/2010.10293.

[4] J. Pei, K. Zhong, M. A. Jan, and J. Li, "Personalized federated learning framework for network traffic anomaly detection," *Comput. Netw.*, vol. 209, no. C, May 2022. [Online]. Available: https://doi.org/10.1016/j.comnet.2022.108906.

[5] D. Gupta, O. Kayode, S. Bhatt, M. Gupta, and A. S. Tosun, *Hierarchical federated learning based anomaly detection using digital twins for smart healthcare*, 2021. [Online]. Available: https://arxiv.org/abs/2111.12241.

[6] L. Cui, Y. Qu, G. Xie, D. Zeng, R. Li, S. Shen, and S. Yu, "Security and privacy-enhanced federated learning for anomaly detection in iot infrastructures," *IEEE Transactions on Industrial Informatics*, vol. 18, no. 5, pp. 3492–3500, 2022.

[7] S. Agrawal, S. Sarkar, O. Aouedi, G. Yenduri, K. Piamrat, S. Bhattacharya, P. K. R. Maddikunta, and T. R. Gadekallu, *Federated learning for intrusion detection system: Concepts, challenges and future directions*, 2021. [Online]. Available: https://arxiv.org/abs/2106.09527.

[8] T. T. Huong, T. P. Bac, D. M. Long, T. D. Luong, N. M. Dan, L. A. Quang, L. T. Cong, B. D. Thang, and K. P. Tran, "Detecting cyberattacks using anomaly detection in industrial control systems: A federated learning approach," *Computers in Industry*, vol. 132, p. 103 509, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0166361521001160.

[9] A. Anaissi and B. Suleiman, *A personalized federated learning algorithm: An application in anomaly detection*, 2021. [Online]. Available: https://arxiv.org/abs/2111.02627.

[10] Y. Zhao, J. Chen, D. Wu, J. Teng, and S. Yu, "Multi-task network anomaly detection using federated learning," Dec. 2019, pp. 273–279.

[11] Z. Aouini and A. Pekar, "Nfstream: A flexible network data analysis framework," *Computer Networks*, vol. 204, p. 108 719, 2022. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/S1389128621005739`.

[12] D. J. Beutel, T. Topal, A. Mathur, X. Qiu, T. Parcollet, and N. D. Lane, "Flower: A friendly federated learning research framework," *arXiv preprint arXiv:2007.14390*, 2020.