Future Generation Computer Systems 166 (2025) 107641

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Flow timeout matters: Investigating the impact of active and idle timeouts on the performance of machine learning models in detecting security threats

Meryem Janati Idrissi [a,*], Hamza Alami [b], Abdelkader El Mahdaouy [a], Abdelhak Bouayad [a], Zakaria Yartaoui [c,d], Ismail Berrada [a]

[a] College of Computing, Mohammed VI Polytechnic University, Ben Guerir, 43150, Morocco
[b] LISAC Laboratory, Faculty of Sciences Dhar EL Mehraz (FSDM), Sidi Mohamed Ben Abdellah University (USMBA), Fes, Morocco
[c] National Moroccan Computer Emergency and Response Team (maCert), Morocco
[d] Vanguard Center, Mohammed VI Polytechnic University, Ben Guerir, Morocco

ARTICLE INFO

Keywords:
Network security and privacy
Network intrusion detection
Flow timeout
Machine learning models

ABSTRACT

In the era of high-speed networks and massive data, several network security technologies are shifting focus from payload-based to flow-based methods. This has led to the incorporation of Machine Learning (ML) models in network security systems, where high-quality network flow features are of paramount importance. However, limited attention has been dedicated to studying the impact of the flow metering hyperparameters, specifically idle and active timeouts, on ML models' performance. This paper, therefore aims to address this gap by designing a series of experiments related to flow features and learning models in the case of Network Intrusion Detection Systems (NIDS). Our experiments investigate the impact idle and active timeouts have on the quality of the extracted features from network data and their subsequent impact on the performance of ML models. For this end, we consider three flow exporters for feature extraction (NFStream, Zeek, and Argus), three ML models, and different feature sets. We conducted extensive experiments with public datasets including, USTC-TFC2016, CICIDS2017, UNSW-NB15, and CUPID. The results show that the difference between best and worst timeout combinations may reach up to 8.77% in terms of macro F1-score. They also unveil varying sensitivity to changes in timeouts among different models and feature sets. Finally, we propose a distributed learning approach based on federated learning. The latter showcased potential in handling multiple NIDS with different timeout configurations. The code is available at https://github.com/meryemJanatiIdrissi/Flow-timeout-matters.

1. Introduction

The exponential growth in internet usage has become a defining characteristic of the digital age, driven by the widespread adoption of online services, the proliferation of connected devices, and the growing importance of digital communication. This surge in usage has led to a significant uptick in internet traffic. To meet the rising demand for data transmission, there have been substantial improvements in network infrastructure, resulting in notable enhancements in line speeds and bandwidth capacities. However, this surge in internet usage has also attracted malicious actors, leading to a growing number of cyberattacks.[1] For instance, during COVID-19 pandemic and due to the increased internet usage and virtual existence [1], cybercrime increased by 400% according to data from the Federal Bureau of Investigation (FBI).[2] These incidents present an ongoing threat to individuals, organizations, and critical digital ecosystems. The increasing frequency of cyber incidents underscores the need for heightened cybersecurity measures and a deeper understanding of the evolving threat landscape.

Within this context, various network security systems have become essential tools for enhancing cybersecurity measures and mitigating the impact of cyberattacks. Noteworthy examples include Network Security Monitoring (NSM) [2], Security Information and Event Management (SIEM) [3], and Network Intrusion Detection Systems (NIDS) [4,5]. These tools typically operate by analyzing network data to identify potential security threats. Nevertheless, the substantial volume of network traffic makes the examination of each packet impractical and

* Corresponding author.
E-mail addresses: Meryem.Janati@um6p.ma (M. Janati Idrissi), hamza.alami@um6p.ma (H. Alami), abdelkader.mahdaouy@um6p.ma (A. El Mahdaouy), abdelhak.bouayad@um6p.ma (A. Bouayad), zakaria.yartaoui@um6p.ma (Z. Yartaoui), ismail.berrada@um6p.ma (I. Berrada).

[1] https://www.enisa.europa.eu/publications/enisa-threat-landscape-2022.
[2] https://securelogix.com/news/fbi-sees-400-spike-in-cyber-crime-reports-during-coronavirus-pandemic/.

https://doi.org/10.1016/j.future.2024.107641
Received 7 December 2023; Received in revised form 15 November 2024; Accepted 26 November 2024
Available online 11 December 2024

resource-intensive [6]. In this challenging context, opting for flow-based methods [7,8] emerges as a practical and highly efficient alternative. Unlike payload-based approaches [9], which involve inspecting the complete content of packets, including both headers and payload, flow-based methods leverage aggregated network information in the form of flows [10]. This approach substantially reduces the volume of data requiring analysis, thereby minimizing the time and resources needed for detecting and responding to potential threats. According to [11], flow-based analysis can achieve data reduction in the order of 1/2000 of the original volume, as packets are aggregated into flows rather than being individually processed.

Integrating Machine Learning (ML) techniques into flow-based approaches for network security systems enhances the efficacy of these tools. Using ML algorithms enhances the precision of network flow analysis and allows for better adaptation to evolving threat landscapes compared to traditional rule-based methods [12,13]. This integration plays a pivotal role in significantly reducing the volume of data that requires analysis, thus minimizing the time and resources essential for detecting and responding to potential threats. In the context of NIDS [14], numerous ML-based solutions have been proposed as a response to the continuously evolving and increasingly sophisticated nature of cyber threats, necessitating more effective and adaptable detection methods. These solutions encompass a diverse range of ML techniques, including generative adversarial networks [15], autoencoders [16,17], self-learning [18], etc. Yet, it is crucial to emphasize that the effectiveness of these models is tied to the quality of the flow features used in the learning process [19]. Choosing and extracting pertinent features from network traffic data is crucial for the overall performance and accuracy of these ML-based NIDS solutions [20]. Sarhan et al. [21] introduced an extended NetFlow feature set, demonstrating that it outperforms the original and basic feature sets in terms of F1-score for both binary-class and multi-class classification scenarios. The improved feature set showed a significant advantage especially over the basic NetFlow features, with performance differences exceeding 30%. Consequently, the feature extraction process significantly impacts these systems' capabilities to identify and effectively mitigate emerging threats.

To extract network flow features, several tools are available such as NFStream [22], CICFlowMeter,[3] Zeek/Bro-IDS,[4] and Argus.[5] These tools are vital for collecting and analyzing network flows, generally relying on a set of hyperparameters that configure and govern various aspects of the feature extraction process. These hyperparameters influence how features are selected, extracted, and represented from the raw network flow data, thus impacting the quality of the extracted features. One of these properties is the time interval, comprising two timeouts, an active and an idle (inactive) timeout [6]. The active timeout denotes the duration of an established connection during which data is actively transmitted between the source and the destination. In contrast, the idle timeout represents the duration of inactivity within an established connection. Modifying the values of these two parameters leads to the extraction of different features, potentially influencing the performance of ML models that rely on these features for their learning and decision-making processes.

To the best of our knowledge, there exists a lack of comprehensive examination of the impact of idle and active timeout values on the overall performance of ML-based NIDS. Additionally, a clear justification for the choice of these values is often missing. Therefore, this paper seeks to bridge this notable knowledge gap. We recognize that while this study addresses this problem in the context of NIDS, it remains a relevant concern for other flow-based network security systems. Thus, in this paper, we propose an in-depth examination of the influence of active and idle timeout values on the performance of ML models in detecting security threats. By designing a series of experiments, our research aims to provide a holistic understanding of the interplay between model inputs, model nature, and the choice of idle and active timeouts. We have considered:

- Three well-known flow extractors, namely, NFstream, Zeek, and Argus,
- A total of 32 combinations of active and idle timeouts for NFStram and 11 idle timeouts for both Zeek and Argus,
- Various ML models, namely Extra Trees Classifier (ETC), Random Forest Classifier (RFC), and Multi-Layer Perceptron (MLP),
- Different evaluation metrics,
- Different sets of features, and,
- Two training strategies, namely a centralized and a distributed architecture (federated learning).

Our findings underscore the significant impact of varying idle and active timeouts on the performance of ML models. For instance, the performance dropped by 8.77% in terms of macro F1-score for the CICIDS2017 dataset and 7.02% for the CUPID dataset. Furthermore, our findings show that ETC and RFC models are more stable to timeout variance in comparison to MLP which exhibits greater performance variability. On the other hand, federated learning showed promise in scenarios where distinct timeouts were employed for data generated by multiple NIDS. Finally, the experimental findings emphasize that there are no universally perfect idle and active timeout values, underscoring the essential need for fine-tuning these parameters. In summary, the main contributions of this paper are as follows:

1. We conduct a thorough examination of the effects of different active and idle timeout values on the performance of ML models in the context of NIDS with various flow exporters.
2. We assess the effectiveness of a variety of ML models, including ETC, RFC, and MLP models. This comprehensive evaluation offers insights into the suitability of various models for NIDS with various combinations of idle and active timeouts.
3. We assess the models' performance using distinct sets of features for each idle and active timeout couple. This analysis seeks to identify whether a specific flow timeout excels in performance.
4. We model a realistic scenario involving distributed NIDS instances, each owning data extracted using distinct idle and active timeouts. For this purpose, we explore federated learning and assess its performance to determine whether it introduces noteworthy enhancements. This investigation demonstrates the promise of federated learning in handling distributed NIDS with varying idle and active timeouts.

To the best of our knowledge, this paper represents the first examination of the impact of idle and active timeout values on the overall performance of both centralized and decentralized ML-based network security systems in the context of NIDS.

The rest of the paper is structured as follows. Section 2 provides the research context defining the scope of our work and the research objectives. Section 3 introduces the different components of the experimental design of the paper. Section 4 describes the implementation details and the experimental results and findings. Section 5 presents existing literature and related work. Finally, Section 6 draws conclusions, the paper's findings, and future works.

2. Problem statement

In this section, we provide a detailed introduction to IP flows within the context of network monitoring, with a particular emphasis on active and idle timeouts and their substantial impact on data features. We then describe the problem raised in this paper and our research objectives.

[3] https://github.com/ahlashkari/CICFlowMeter.
[4] https://docs.zeek.org/en/master/about.html.
[5] https://openargus.org/documentation.

2.1. IP flows

Traditionally, NIDS employ Deep Packet Inspection (DPI) [23] which involves a thorough analysis of the data packets traveling through a network. Each packet contains information about its source, destination, content, and other relevant details. DPI goes beyond merely examining the headers of these packets; it delves deep into the packet's payload, inspecting the actual data within. By inspecting the contents of data packets, payload-based NIDS can identify known attack signatures, unusual data patterns, or deviations from normal network behavior. However, inspecting the entire payload incurs significant computational overhead and hinders performance, especially in high-speed IP networks [24]. Moreover, the proliferation of encrypted protocols in network traffic adds substantial complexity to the task of implementing packet-based NIDS [25].

With these challenges in mind, researchers have shifted their attention toward flow-based methods. Flow-based NIDS leverage network flow records as their input source to detect potentially malicious activities. In the context of computer networks, a flow refers to a sequence of packets that share certain common attributes and characteristics as they traverse a network. In contrast to payload-based NIDS, flow-based NIDS deal with considerably reduced amounts of data as only the flow records are analyzed. In other words, with such systems, the aggregated information of the network is inspected instead of the content of data packets. According to a study that was conducted on the University of Twente (UT) network [6], the analysis of flow packets within a network typically accounts for approximately 0.1% of the overall network traffic. For network load, the added overhead resulting from flow collection and export protocol (Netflow) averages around 0.2%. Another advantage of flow-based NIDS is the ease with which flow data can be collected from network devices that employ standard and widely recognized protocols such as Cisco, NetFlow, and IETF IPFIX, without the need for additional software installations. Flow-based NIDS are therefore a rational choice for high-speed networks.

The IP Flow Information Export (IPFIX) proposed a standard definition of network flow [26]:

> "A Flow is defined as a set of packets or frames passing an observation point in the network during a certain time interval. All packets belonging to a particular Flow have a set of common properties".

These properties are called *flow keys* and typically are a 5-tuple consisting of the source and destination IPs and ports and the protocol:

```
(ip_src, ip_dst, port_src, port_dst, proto)
```

The typical structure of a network flow monitoring tool consists of three main steps: *packet observation*, *flow metering and export*, and *flow collection*. First, the packet observation stage is designed to capture packets from an *observation point* and pre-process them. Next, the metering process generates flow records based on headers extracted from packets collected from the observation point. Each header is then marked with a timestamp. Every incoming packet header initiates an update to an existing flow entry in the flow cache used to temporarily store flow entries and maintain a record of active flows in real-time. If the packet header matches an existing entry then the flow features are updated. Otherwise, a new flow entry is initiated. A flow record is forwarded to the collecting process once it has expired. For both NetFlow [27] and IPFIX [28], the metering process terminates a flow record for the following reasons:

- *Active timeout*: Flows that remain active beyond this specified time duration are considered expired and the corresponding flow records are sent to the flow collector. Common timeout values typically fall within the range of 2 min to 30 min (NetFlow).
- *Idle timeout*: If no packets have been observed in the flow for a longer time than this value then the flow is expired. Common timeout durations vary from 15 s (NetFlow) to 5 min.

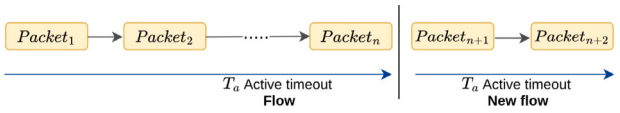


Fig. 1. Active timeout.

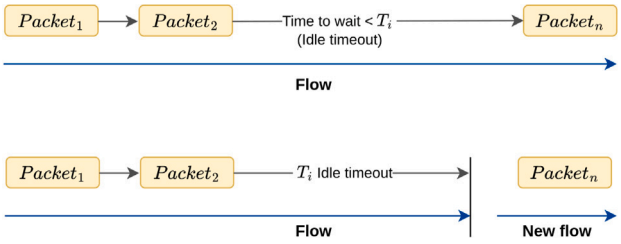


Fig. 2. Idle timeout.

- *Natural expiration*: The detection of a Transmission Control Protocol (TCP) packet with either the FIN or RST flag set indicates the termination of the TCP connection.
- *Emergency expiration*: If the flow cache memory is exhausted, a subset of flow entries are immediately terminated and exported to the collector.

Finally, the flow collector stores flow records in a format suitable for subsequent monitoring or analysis.

2.2. Active and idle timeouts

In the realm of networking, *idle* and *active* timeouts are two parameters that are commonly used in various network protocols and devices to manage flows (network connections and sessions). Active timeout, sometimes termed connection timeout, places a predetermined limit on the maximum duration a flow can remain open, irrespective of data activity. It measures the time between successive packets or data exchanges. If no data is transmitted within the specified period, the flow may be considered inactive or stalled, and the active timeout timer will start counting down. Once the active timeout expires, the flow is terminated. The principal purpose of active timeout is to prevent flows from persisting indefinitely, thereby conserving resources and ensuring network health. Active timeouts are fundamental to network protocols like TCP, contributing significantly to the maintenance of flow integrity and the prevention of resource exhaustion.

Idle timeout, also referred to as inactive timeout, is the period of inactivity between a source and a destination before the flow is considered idle and subsequently terminated. It measures the duration during which there is no data transmission occurring between the two endpoints. When the idle timeout expires, the flow is closed, and the resources associated with that connection are freed up for other tasks. This helps prevent flows from being kept open indefinitely when there is no ongoing communication. Therefore, idle timeout plays a central role in minimizing security risks by closing idle connections as well as optimizing the allocation of incoming network traffic. Figs. 1 and 2 provide illustrative representations of the active and idle timeouts respectively.

The values of idle and active timeouts in network flow analysis hold a substantial influence over the features extracted from the data, and consequently, may impact the performance of ML models optimized with these features. These timeout parameters influence the granularity and relevance of the extracted features. For instance, shorter idle and active timeouts can yield finer-grained features that capture transient or rapid changes in network activity, while longer timeouts tend to

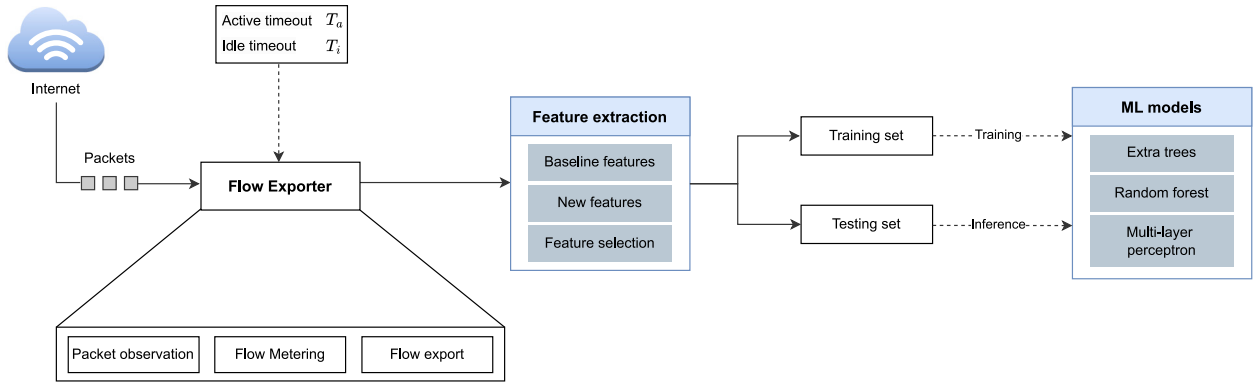


Fig. 3. The overflow of the proposed design of experiments.

result in coarser features representing broader time intervals. Moreover, these choices bear on the level of noise reduction within the data; shorter timeouts may help filter out idle or sporadic connections but risk discarding benign ones, whereas longer timeouts could introduce more noise. Furthermore, the sensitivity of the model to network dynamics and its computational resource utilization are both affected by the choice of timeout values. Striking the right balance between capturing fine details and managing resources is essential for optimal model performance, as is ensuring that the selected timeout values align with the specific network environment and analytical goals, ultimately influencing the model's ability to detect patterns and anomalies effectively.

2.3. Research objectives

Network analysis software and intrusion detection systems, both open-source (such as Snort [29] and Suricata [30]), and commercial (such as Palo Alto Networks [31] and Fortinet [32]) incorporate configurations for both active and idle timeouts. While many of these solutions come with predefined and optimal values for these parameters, it is important to note that there are instances where these values may not align with the specific requirements of the network. Opting for a value that is too low can heighten sensitivity to minor network delays. On the other hand, choosing an excessively high session timeout may result in delayed detection of failures. Moreover, when constructing NIDS datasets from the flow data, the quality and relevance of the extracted features highly depend on the idle and active timeouts. Therefore flow timeouts hold substantial importance in the development and assessment of network intrusion detection systems.

Throughout our literature review, we have noticed that the prevalent approach to configuring flow timeouts involves either adhering to the pre-configured values for both idle and active timeouts throughout the feature extraction process or opting to set new ones without providing substantial justification for their selection [33,34]. Surprisingly, limited attention has been devoted to investigating the potential ramifications of altering these parameters' values on the quality of extracted features and the subsequent impact on ML models' performance. This work aims therefore to address this significant knowledge gap. Its primary goal is to thoroughly investigate the effect of employing different values for both idle and active timeouts while utilizing various ML models. By doing so, we shed light on whether there is a compelling value for these two parameters before the feature extraction phase. Such insights are critical in optimizing NIDS systems and enhancing their ability to detect and respond to emerging security threats.

3. Design of experiments

In this section, we delve into the core elements of our experimental design illustrated in Fig. 3. First, we present an in-depth exploration of both active and idle timeouts, shedding light on their direct impact on feature extraction and, subsequently, on the overall performance of ML models. Following this, we present the feature extraction pipeline employed throughout our study. Then, we provide a brief description of the ML models leveraged in our research. Finally, we discuss the federated learning setting adopted to assess the performance in a decentralized environment for distributed NIDS.

The series of experiments conducted in this work, aim to answer the following research questions:

- **RQ1:** Considering different flow exporters, What is the impact of idle and active timeouts on the performance of three well-known ML models?
- **RQ2:** Within diverse feature sets of network flows, does a particular combination of idle and active timeouts exhibit superior performance compared to others?
- **RQ3:** In the context of a real-world scenario involving multiple NIDS with varying idle and active timeout settings, is it possible to construct a global model with federated learning that delivers strong performance across all these diverse environments?
- **RQ4:** Taking into account the responses to the preceding research questions, can we provide recommendations regarding the utilization of specific idle and active timeouts in the context of ML-based NIDS?

3.1. Feature extraction

Vormayr, Fabini and Zseby [35] demonstrated substantial differences in the results produced by different flow exporters, which can significantly affect subsequent analyses. Therefore, in this work, we consider three widely used flow exporters to extract features from raw traffic packets, namely, NFStream, Zeek, and Argus.

For NFStream, we build a set of flow features based on four distinct sources, including the default features proposed by NFStream, the features selected in the work [36], the features introduced in [37], and features developed by our team. Table 1 presents the set of features created by our team and their descriptions. In addition, we provide in Appendix A the list of features presented in both works [36,37]. The standard features of NFStream can be found in the documentation page.[6]

Zeek (formerly known as Bro) was employed to generate various logs from the packet capture files. Similarly to Giagkos et al. [38] and Rodríguez et al. [39], we focus on Zeek's conn.log file, which provides 21 distinct features offering a detailed view of the traffic flows. Table 16 in Appendix A provides the list of features extracted by Zeek and used in this paper.

[6] https://www.nfstream.org/docs/api#nflow.

Table 1
Set of features developed by our team.

Feature	Description
tcp_init_ms	Time in ms between first SYN and SYN-ACK packet
tcp_synack_ack_ms	Time in ms between SYN-ACK packet and its acknowledgment
tcp_half_closed_time_ms	Time in ms that connection is half closed
num_pkts_after_termination	The number of packet received in 15s after the four-way handshake
bidirectional_transport_bytes	Total bytes of exchanged transport segments
bidirectional_payload_bytes	Total bytes of exchanged packets payload
src2dst_transport_bytes	SRC2DST total bytes of exchanged transport segments
src2dst_payload_bytes	SRC2DST total bytes of exchanged packets payload
dst2src_transport_bytes	DST2SRC total bytes of exchanged transport segments
dst2src_payload_bytes	DST2SRC total bytes of exchanged packets payload

Lastly, for Argus, we utilized the main tool *argus*, which serves as the Flow Exporter and Capturer. This tool, which runs on the server side, converts raw PCAP files into a format compatible with Argus, generating detailed network flow records. These records are subsequently processed by *ra*, a powerful Argus client, to extract specific features from the flow data for further analysis. In this research, 27 features, presented in Table 17 in Appendix A, were extracted from both normal and malicious traffic. These features represent the common set used in both [40,41].

Note that for each flow exporter, we exclude identifiers like IP addresses, MAC addresses, and other unique identifiers from the extracted feature sets [42].

3.2. Machine learning models

Given that the task at hand involves multi-class classification, we use three well-known ML classifiers (representing different learning algorithm classes) to examine the impact of idle and active timeouts on NIDS performance.

- *Extra Trees Classifier (ETC) [43]* is an ensemble ML algorithm that belongs to the trees family. ECT is often used in classification tasks and is computationally efficient, making it suitable for large-scale datasets. Let $T_1, T_2, \ldots, T_N$ represent the individual decision trees, and given a new input X_{new}, the ETC's classification decision y_{final} is determined by:

$$y_{final} = mode(T_1(X_{new}), T_2(X_{new}), \ldots, T_N(X_{new})) \quad (1)$$

  where X_{new} is the input data to be classified, $T_i(X_{new})$ represents the prediction of the ith decision tree for the input X_{new}, and *mode* calculates the majority vote among the individual tree predictions.
- *Random Forest Classifier (RFC) [44]* is also an ensemble learning method for classification problems, similar to ETC. The key distinction lies in how they create data subsets and introduce randomness during the tree-building process. It is composed of a collection of decision trees, where each tree is trained on a different subset of the data. The final classification decision is made based on the majority vote of all the individual trees. The same representation of the ETC classifier holds for the RF classifier as both are ensemble methods based on decision trees.
- *Multilayer perceptron (MLP) [45]* is a type of feedforward neural network with one or more hidden layers and is widely employed in various domains due to its capability to model complex non-linear relationships between input features and target classes. Given input features $X = [x_1, x_2, \ldots, x_n]$, where n is the number of input features, and assuming a single hidden layer h with p neurons and an output layer with m neurons (for classification tasks with m classes) where Y represents a vector of output labels. The MLP classifier can be expressed as follows:
  Hidden layer output:

$$h = \sigma(W_1 X + b_1) \quad (2)$$

  Output layer output:

$$Y = \text{softmax}(W_2 h + b_2) \quad (3)$$

  where σ represents the chosen activation function for the hidden layer, such as the sigmoid function, hyperbolic tangent function, or ReLU [46,47], W_1 and W_2 are weight matrices of size $p \times n$ and $m \times p$ respectively, for the connections between layers, and b_1 and b_2 are bias vectors for the respective layers of size p and m respectively.

3.3. Distributed learning in a multiple timeouts environment

To assess the potential of distributed learning within the context of multiple NIDS, each characterized by a particular set of idle and active timeouts, we have opted to utilize the federated learning framework. Federated learning is a decentralized ML approach where a global model is collaboratively trained across multiple decentralized devices or data sources, all while keeping data localized. This approach offers advantages, including enhanced privacy, scalability, and efficiency. Our decision to employ federated learning aims to explore its potential benefits within the context of our specific learning task of heterogeneous network intrusion detection systems. By leveraging this decentralized approach, we aim to gain a deeper understanding of its applicability and effectiveness in managing diverse NIDS with varying configurations. Moreover, we seek to uncover whether federated learning can deliver superior results in terms of threat detection compared to traditional centralized approaches.

In our simulation of the distributed setting, we model it as a system consisting of K clients (NIDS) and a central server. Each client possesses data collected with a particular idle and active timeouts $(T_{i,k}, T_{a,k})$. During each communication round, a fraction C of clients is selected randomly to collectively optimize the global model. Initially, the server initializes and disseminates the global model to the selected clients in the first round. Subsequently, each client individually optimizes the model using its local data for E local epochs and then sends the model's weight updates back to the server. Once all the updates are received from the participating clients, the server aggregates the local models and updates the global model. The new global model is then shared with the participating clients in the subsequent round. These steps are repeated until convergence is reached, i.e., the loss reaches a steady state. Fig. 4 depicts the federated learning setting.

4. Experiments and results

This section presents the implementation details and discusses the performance evaluation examined from four perspectives: timeout-based, model-based, feature-based, and distributed learning performances.

4.1. Datasets

Four well-known datasets are used in this work, namely:

- ***USTC-TFC2016*** [48] is a well-known dataset consisting of two parts. The first part comprises 10 different types of malware traffic collected from public websites that were operational in a real network environment between 2011 and 2015. The second part consists of ten genres of normal network traffic, acquired using the IXIA BPS tool. The dataset is stored in the PCAP format, amounting to a total size of 3.71 GB.
- ***CICIDS2017*** [49][7] provides a realistic representation of network traffic in a controlled environment and aims to mimic real-world scenarios. The dataset was collected over five days, from Monday through Friday, capturing a diverse range of network activities. It includes two main categories of network traffic: "Benign" and "Attacks". The network traffic data is provided in the PCAP format.
- ***UNSW-NB15*** [50] is a modern KDD-19 alternative created and released in 2015. The IXIA PerfectStorm tool was used to create a hybrid testbed-based normal and abnormal network traffic. 87.35% of the data is benign flows while the 12.65% left is attack flows. The dataset consists of nine attack classes, namely, Fuzzers, Reconnaissance, DoS, Backdoors, Generic, Analysis, Worms, Shellcode, and Exploits. Note that the traffic was captured in the PCAP format.
- ***CUPID*** [51][8] is a recent dataset that was created specifically for evaluating NIDS. CUPID was annotated with penetration testing to reflect both automated and human-generated attacks. It provides diverse attack types, including, Webcrawling, ARP, nmap, Recorded live user interaction, DNSMap, Dig, Nslookup, DNSTracer, Password Brute Forcing, SQLi, Directory Traversal, DHCP attacks, STP, and Delivery of Reverse Meterpreter Shell. The dataset was captured in the PCAP format and processed using CICFlowMeter comprising approximately 50 GB.

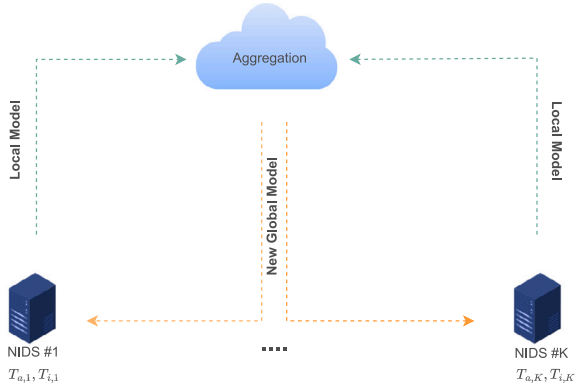


Fig. 4. Distributed scheme using federated learning.

4.2. Implementation details

To evaluate the proposed method in the centralized setting, we consider three classifiers defined in the Scikit-learn library [52], Extra Trees Classifier (ETC), Random Forest Classifier (RFC), and Multilayer perceptron (MLP). The ETC and RFC models used in this paper are both constructed of 100 decision trees, whereas the MLP model is trained for 8000 maximum iterations with the default parameters. To ensure the robustness and reliability of our evaluation, we assess the models using 5-fold cross-validation to estimate the average performance metrics over all 5 trials, along with their corresponding standard deviations. To thoroughly evaluate the proposed methodology, we examine a range of different flow timeout values.

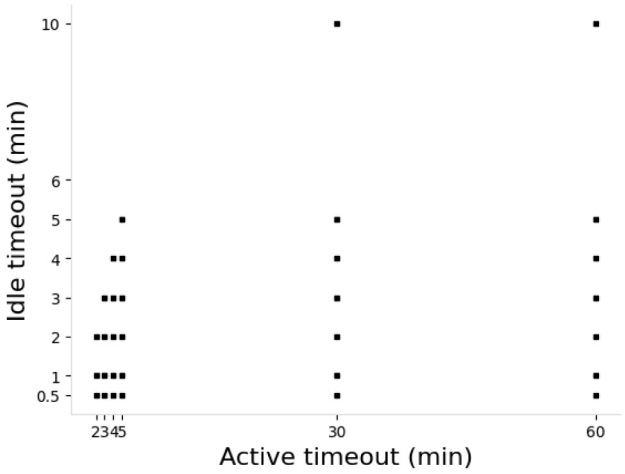


Fig. 5. The 32 combinations of idle and active timeouts.

For NFStream, we consider 32 combinations of idle and active timeouts, including default values ($T_i = 2$ mins and $T_a = 30$ mins). Fig. 5 presents all the combinations used for the evaluation:

- The idle timeout (min) takes values from [0.5, 1, 2, 3, 4, 5, 6, 10], and,
- The active timeout (min) varies within the list [2, 3, 4, 5, 30, 60].

It is important to notice that the idle timeout is always less than or equal to the active timeout (idle $\leq$ active).

As Zeek and Argus do not provide a direct configuration parameter for the active timeout, we have only configured the idle timeouts in our setup. For Zeek, these are the tcp_inactivity_timeout, udp_inactivity_timeout, and icmp_inactivity_timeout parameters, which are set by default to 5, 1, and 1 minutes, respectively. For Argus, the idle timeouts include ARGUS_IP_TIMEOUT, ARGUS_TCP_TIMEOUT, ARGUS_ICMP_TIMEOUT, ARGUS_IGMP_TIMEOUT, ARGUS_FRAG_TIMEOUT, ARGUS_ARP_TIMEOUT, and ARGUS_OTHER_TIMEOUT, which are fixed to 30, 60, 5, 30, 5, 5 and 30 seconds by default. In addition to the default values, we also consider the following idle timeout variations:

- The idle timeout (min) takes values from [0.5, 1, 2, 3, 4, 5, 6, 10, 30, 60].

For each timeout variation, we train ETC, RFC, and MLP models and we measure the performance in terms of four well-established metrics commonly used for intrusion detection: Accuracy, F1-score, Recall, and Precision. We specifically use the macro-average metric, as it assigns equal importance to all classes, making it preferable in our context compared to the micro-average, which aggregates the contributions of all classes. We record the best and worst timeouts for each dataset and model. The 'best timeout' denotes the specific combination of idle and active timeouts used for feature extraction, which yielded the highest performance for the ML model in terms of macro F1-score. Conversely, the 'worst timeout' represents the combination that resulted in the poorest model performance.

For the distributed scheme, we consider an environment of 32 clients, each owning local data extracted using NFStream with a particular and unique idle and active timeout couple (presented above). We adopt the standard federated learning algorithm, *FedAvg* [53]. At each communication round, a subset of fraction $C = 0.5$ is selected

[7] Canadian Institute for Cybersecurity Intrusion Detection Dataset 2017 (CICIDS2017).

[8] The Colorado University Pentesting Intrusion Dataset (CUPID) https://cupid.directory/.

Table 2
Multi-class classification results for USTC-TFC2016 dataset.

	Metrics	Best timeouts			Worst timeouts			Performance difference		
		ETC	RFC	MLP	ETC	RFC	MLP	ETC	RFC	MLP
NFstream	Macro F1-score	91.61 ± 0.03	91.82 ± 0.03	85.38 ± 0.04	89.54 ± 0.03	89.84 ± 0.03	81.39 ± 0.05	2.07	1.98	**3.98**
	Macro recall	93.01 ± 0.03	93.07 ± 0.03	87.45 ± 0.03	89.98 ± 0.02	90.06 ± 0.02	84.62 ± 0.03	3.03	3.01	2.83
	Macro precision	91.44 ± 0.04	91.72 ± 0.03	87.52 ± 0.05	90.51 ± 0.03	90.95 ± 0.04	83.83 ± 0.06	0.93	0.76	3.70
	Accuracy	93.48 ± 0.02	93.46 ± 0.02	89.29 ± 0.02	91.68 ± 0.02	92.10 ± 0.02	85.86 ± 0.04	1.79	1.36	3.43
	Timeouts	(10, 60)	(10, 60)	(3, 60)	(0.5, 4)	(0.5, 60)	(0.5, 30)	–	–	–
Zeek	Macro F1-score	88.70 ± 0.06	87.02 ± 0.03	80.28 ± 0.05	86.83 ± 0.06	84.59 ± 0.07	77.42 ± 0.06	1.87	2.43	**2.87**
	Macro recall	89.93 ± 0.04	88.93 ± 0.03	80.06 ± 0.05	88.91 ± 0.04	87.43 ± 0.04	79.65 ± 0.05	1.02	1.50	0.41
	Macro precision	89.43 ± 0.06	89.07 ± 0.05	82.59 ± 0.04	87.21 ± 0.07	86.00 ± 0.07	78.50 ± 0.06	2.22	3.07	4.09
	Accuracy	96.05 ± 0.02	95.44 ± 0.01	93.69 ± 0.02	95.15 ± 0.03	93.44 ± 0.04	92.93 ± 0.02	0.90	2.00	0.77
	Timeouts	Default	30	Default	0.5	0.5	60	–	–	–
Argus	Macro F1-score	82.69 ± 0.03	83.52 ± 0.02	75.18 ± 0.03	81.64 ± 0.02	82.52 ± 0.02	73.82 ± 0.02	1.05	1.01	**1.36**
	Macro recall	81.33 ± 0.03	82.25 ± 0.02	74.32 ± 0.03	80.08 ± 0.02	80.97 ± 0.01	72.70 ± 0.02	1.25	1.28	1.62
	Macro precision	86.11 ± 0.02	86.75 ± 0.02	82.73 ± 0.03	85.58 ± 0.02	86.33 ± 0.02	81.49 ± 0.02	0.53	0.42	1.24
	Accuracy	84.84 ± 0.03	85.12 ± 0.03	81.02 ± 0.03	83.07 ± 0.01	83.39 ± 0.01	78.91 ± 0.02	1.77	1.74	2.11
	Timeouts	2	2	2	5	5	4	–	–	–

Mean in percentage ± standard deviation.

Table 3
Multi-class classification results for CICIDS2017 dataset.

	Metrics	Best timeouts			Worst timeouts			Performance difference		
		ETC	RFC	MLP	ETC	RFC	MLP	ETC	RFC	MLP
NFstream	Macro F1-score	96.45 ± 0.03	94.98 ± 0.03	92.22 ± 0.04	89.29 ± 0.05	86.27 ± 0.04	84.38 ± 0.02	7.17	**8.71**	7.83
	Macro recall	94.80 ± 0.04	93.19 ± 0.04	92.63 ± 0.05	88.14 ± 0.05	84.93 ± 0.05	85.58 ± 0.05	6.66	8.26	7.05
	Macro precision	99.48 ± 0.01	97.85 ± 0.02	94.14 ± 0.03	92.21 ± 0.04	89.76 ± 0.03	85.23 ± 0.01	7.27	8.09	8.91
	Accuracy	99.70 ± 0.00	99.70 ± 0.00	99.66 ± 0.00	99.63 ± 0.00	99.63 ± 0.00	99.62 ± 0.00	0.07	0.07	0.05
	Timeouts	(0.5, 60)	(0.5, 60)	(1, 60)	(10, 60)	(10, 60)	(10, 60)	–	–	–
Zeek	Macro F1-score	71.89 ± 0.06	67.83 ± 0.12	64.31 ± 0.09	66.77 ± 0.05	64.32 ± 0.10	55.54 ± 0.11	5.12	3.51	**8.77**
	Macro recall	73.75 ± 0.07	70.54 ± 0.08	71.56 ± 0.06	69.81 ± 0.06	67.32 ± 0.08	66.11 ± 0.06	3.94	3.22	5.44
	Macro precision	72.99 ± 0.06	71.20 ± 0.09	63.94 ± 0.10	67.68 ± 0.04	67.22 ± 0.08	55.16 ± 0.12	5.31	3.98	8.77
	Accuracy	88.17 ± 0.03	78.30 ± 0.21	82.44 ± 0.12	88.51 ± 0.03	78.15 ± 0.22	76.29 ± 0.18	−0.34	0.15	6.15
	Timeouts	0.5	1	6	4	3	5	–	–	–
Argus	Macro F1-score	72.26 ± 0.03	73.74 ± 0.03	70.59 ± 0.02	69.05 ± 0.08	69.49 ± 0.08	64.92 ± 0.12	3.20	4.26	**5.67**
	Macro recall	72.98 ± 0.03	75.35 ± 0.03	70.69 ± 0.03	72.15 ± 0.04	72.54 ± 0.05	69.39 ± 0.06	0.84	2.80	1.30
	Macro precision	73.13 ± 0.04	75.76 ± 0.04	75.30 ± 0.02	72.67 ± 0.07	73.17 ± 0.08	69.77 ± 0.09	0.46	2.59	5.53
	Accuracy	93.09 ± 0.01	93.03 ± 0.01	93.97 ± 0.00	80.76 ± 0.25	80.11 ± 0.26	86.33 ± 0.16	12.33	12.92	7.64
	Timeouts	6	60	6	5	2	5	–	–	–

to optimize the global model on their local data for $E = 5$ epochs. The process is repeated for $R = 15$ communication rounds for the USTC-TFC2016 dataset and $R = 10$ for the CUPID, CICIDS2017, and UNSW-NB15 datasets. We define the global model as an MLP neural network consisting of three fully connected layers specified by the following number of neurons per layer: (57, 32, 16).

4.3. Performance evaluation

In the following, we present all the experiments and evaluations performed to study the impact of idle and active timeouts on the performance of ML-based NIDS.

4.3.1. Evaluating ML models' performance with different timeouts

In the first set of experiments, we measure the network intrusion detection performance of ETC, RFC, and MLP models for each dataset using the NFStream standard feature set, and Zeek and Argus feature sets presented in Tables 16 and 17 respectively (Appendix A). Each feature set is computed based on different timeouts' values (32 pairs of idle and active timeouts for NFStream and 11 idle timeouts for Zeek and Argus). Tables 2, 3, 4, and 5 present the recorded best and worst timeouts and their performance difference for USTC-TFC16, CICIDS2017, UNSW-NB15, and CUPID datasets, respectively. For each dataset, we observe a unique (T_i, T_a) combination that achieves the optimal performance, showing that the best timeout for a dataset can be the worst for another dataset. For instance, the pair *(10, 60)* is the best timeout pair when the USTC-TFC2016 dataset is used, however, it is the worst in the case of the CICIDS2017 dataset for all models. Furthermore, even if we focus on a single dataset it becomes evident that no timeout tuple excels with all three ML models. A timeout combination could, without varying the environment (dataset), score the best results with one model and the worst results with another. For instance, in Table 4, for the same dataset UNSW-NB15, the timeout tuple *(0.5, 3)* achieves the best score when using the MLP model, but the worst with the RFC model. It is worth mentioning that when using the CICIDS2017 dataset the best performances are scored when the active timeout is large (60 min) and the idle timeout is small (0.5 min and 1 min). When analyzing the F1-score performance difference between the best and worst timeouts, we note two major points:

1. ETC and RFC models are more stable than MLP. In Tables 2, 3, 4, and 5, in most cases, the performance difference of the MLP model is greater than that of the ETC and RFC models, with a maximum difference of approximately ∼9%.However, the computed performance difference of the ETC and RTC models is less than 2.5% for most cases.
2. For some datasets (CICIDS2017), the performance difference shows that it is mandatory to fine-tune the idle and active timeouts, especially when considering the NFStream tool. Yet, for

Table 4
Multi-class classification results for UNSW-NB15 dataset.

	Metrics	Best timeouts			Worst timeouts			Performance difference		
		ETC	RFC	MLP	ETC	RFC	MLP	ETC	RFC	MLP
NFstream	Macro F1-score	72.49 ± 0.01	71.97 ± 0.02	53.85 ± 0.01	71.75 ± 0.02	71.49 ± 0.02	51.15 ± 0.01	0.74	0.48	**2.70**
	Macro recall	71.33 ± 0.01	70.84 ± 0.01	53.06 ± 0.01	70.80 ± 0.00	70.15 ± 0.01	49.61 ± 0.02	0.53	0.69	3.45
	Macro precision	76.85 ± 0.06	77.31 ± 0.08	60.60 ± 0.04	76.24 ± 0.08	76.75 ± 0.08	62.87 ± 0.07	0.60	0.56	−2.27
	Accuracy	98.82 ± 0.00	98.83 ± 0.00	98.54 ± 0.00	98.79 ± 0.00	98.81 ± 0.00	98.57 ± 0.00	0.02	0.02	−0.03
	Timeouts	(4, 5)	(2, 2)	(0.5, 3)	(5, 60)	(0.5, 3)	(4, 60)	–	–	–
Zeek	Macro F1-score	43.02 ± 0.10	45.71 ± 0.09	27.34 ± 0.05	39.62 ± 0.09	44.09 ± 0.10	26.03 ± 0.06	**3.40**	1.62	1.31
	Macro recall	40.95 ± 0.12	46.09 ± 0.13	27.68 ± 0.08	38.83 ± 0.11	45.14 ± 0.13	26.17 ± 0.07	2.12	0.95	1.51
	Macro precision	55.98 ± 0.09	56.31 ± 0.09	39.44 ± 0.04	51.84 ± 0.09	53.64 ± 0.10	38.82 ± 0.06	4.14	2.67	0.63
	Accuracy	97.63 ± 0.01	97.21 ± 0.01	97.78 ± 0.00	97.63 ± 0.01	97.23 ± 0.01	97.81 ± 0.00	−0.01	−0.02	−0.02
	Timeouts	default	default	60	0.5	0.5	10	–	–	–
Argus	Macro F1-score	29.58 ± 0.07	36.95 ± 0.04	19.35 ± 0.04	26.92 ± 0.05	33.99 ± 0.03	17.69 ± 0.03	2.66	**2.96**	1.66
	Macro recall	28.07 ± 0.08	34.65 ± 0.05	18.28 ± 0.04	25.58 ± 0.07	31.87 ± 0.04	17.01 ± 0.04	2.49	2.78	1.26
	Macro precision	43.32 ± 0.10	51.27 ± 0.11	33.75 ± 0.08	40.29 ± 0.10	47.51 ± 0.12	28.31 ± 0.06	3.03	3.77	5.45
	Accuracy	97.49 ± 0.01	97.89 ± 0.01	97.47 ± 0.01	97.48 ± 0.01	97.87 ± 0.01	97.56 ± 0.01	0.01	0.02	−0.09
	Timeouts	60	60	10	1	1	1	–	–	–

Table 5
Multi-class classification results for CUPID for NFStream dataset.

	Metrics	Best timeouts			Worst timeouts			Performance difference		
		ETC	RFC	MLP	ETC	RFC	MLP	ETC	RFC	MLP
NFstream	Macro F1-score	94.93 ± 0.04	95.91 ± 0.05	95.10 ± 0.05	91.97 ± 0.07	95.18 ± 0.05	91.99 ± 0.07	2.97	0.73	**3.11**
	Macro recall	96.52 ± 0.05	96.68 ± 0.05	96.54 ± 0.05	95.93 ± 0.05	96.58 ± 0.05	92.31 ± 0.09	0.59	0.11	4.23
	Macro precision	95.98 ± 0.05	97.37 ± 0.04	95.95 ± 0.04	93.05 ± 0.09	96.16 ± 0.05	95.57 ± 0.05	2.93	1.22	0.38
	Accuracy	96.10 ± 0.04	97.08 ± 0.04	97.15 ± 0.03	96.21 ± 0.03	96.27 ± 0.04	96.96 ± 0.04	−0.11	0.81	0.19
	Timeouts	(3, 3)	(1, 60)	(4, 60)	(1, 2)	(4, 30)	(10, 60)	–	–	–
Zeek	Macro F1-score	77.82 ± 0.11	73.74 ± 0.05	71.73 ± 0.08	70.80 ± 0.10	71.12 ± 0.05	69.92 ± 0.06	**7.02**	2.61	1.81
	Macro recall	81.84 ± 0.06	73.57 ± 0.04	73.54 ± 0.04	75.92 ± 0.05	71.81 ± 0.03	72.31 ± 0.02	5.93	1.76	1.23
	Macro precision	76.87 ± 0.12	74.32 ± 0.05	71.81 ± 0.09	70.40 ± 0.11	71.13 ± 0.06	70.25 ± 0.09	6.46	3.20	1.56
	Accuracy	96.12 ± 0.04	97.29 ± 0.03	97.63 ± 0.03	96.44 ± 0.05	96.79 ± 0.03	97.01 ± 0.04	−0.32	0.50	0.62
	Timeouts	5	4	2	0.5	6	3	–	–	–
Argus	Macro F1-score	71.31 ± 0.09	69.61 ± 0.05	65.59 ± 0.14	65.95 ± 0.20	63.70 ± 0.16	59.64 ± 0.09	5.36	5.91	**5.94**
	Macro recall	70.73 ± 0.08	69.00 ± 0.05	71.73 ± 0.05	71.45 ± 0.07	69.73 ± 0.05	64.82 ± 0.07	−0.73	−0.73	6.91
	Macro precision	74.97 ± 0.11	72.82 ± 0.05	66.76 ± 0.16	68.70 ± 0.23	65.79 ± 0.18	62.03 ± 0.12	6.26	7.04	4.74
	Accuracy	96.70 ± 0.04	96.94 ± 0.04	96.28 ± 0.06	91.66 ± 0.15	92.17 ± 0.10	92.06 ± 0.11	5.04	4.77	4.22
	Timeouts	10	5	Default	0.5	30	30	–	–	–

other datasets (CUPID/UNSW-NB15 with NFStream and USTC-TFC2016 with Argus) and with the right model, the idle and active timeouts have little impact on the performance.

3. When considering Argus as the flow exporter, the results seems to be more consistent regarding timeouts, particularly with USTC-TFC2016 and UNSW-NB15 datasets. The best and worst timeouts are generally the same across the three models.

In the remainder of this paper, we will focus exclusively on NFStream, as it yielded the best performance in the previous analysis compared to Zeek and Argus.

4.3.2. Measuring the relationship between flow features and timeouts

In our forthcoming series of experiments, we redirect our focus toward examining the relationship between flow features and both idle and active timeouts. We used all NFStream flow features described in Section 3.1. Similar to the previous set of experiments, Tables 6, 7, 8, and 9 present the best and worst obtained results with their difference when all flow features are used with all 32 idle and active timeouts combinations. The performance of the MLP model has significantly increased, for instance, the F1-score increased from 85.38% to 94.87% with the USTC-TFC2016 dataset. Similarly, the performance is on par or better when ETC or RFC is used with the expanded flow feature set. However, considering all the scores, no timeout pair achieves top performance with all datasets and models.

Since the scores increased when all flow features were used, we performed various experiments to ascertain the relative importance of features and to identify any relationship between features and timeout pairs. Thus, we build two new feature sets using a feature selection strategy that consists of: (1) computing features' importance using the ETC algorithm; (2) sorting features based on their importance; and (3) selecting 30% and 50% from the sorted list of features. Next,

Table 6
Multi-class classification results for USTC-TFC2016 dataset using NFStream flow features (from 4 sources).

Metrics	Best timeouts			Worst timeouts			Performance difference		
	ETC	RFC	MLP	ETC	RFC	MLP	ETC	RFC	MLP
Macro F1-score	95.39 ± 0.03	96.81 ± 0.03	94.87 ± 0.03	92.88 ± 0.03	92.88 ± 0.03	91.75 ± 0.04	2.51	**3.93**	3.12
Macro recall	96.55 ± 0.03	96.83 ± 0.03	96.18 ± 0.03	93.36 ± 0.02	93.39 ± 0.02	92.76 ± 0.02	3.19	3.44	3.43
Macro precision	95.33 ± 0.03	96.91 ± 0.02	94.91 ± 0.03	93.77 ± 0.03	93.64 ± 0.04	92.72 ± 0.05	1.56	3.27	2.19
Accuracy	97.14 ± 0.02	98.31 ± 0.01	96.90 ± 0.02	95.69 ± 0.02	95.73 ± 0.02	94.29 ± 0.04	1.45	2.59	2.61
Timeouts	(10, 30)	(10, 60)	(10, 60)	(0.5, 60)	(0.5, 60)	(0.5, 3)	–	–	–

Table 7
Multi-class classification results for CICIDS2017 dataset using NFStream flow features (from 4 sources).

Metrics	Best timeouts			Worst timeouts			Performance difference		
	ETC	RFC	MLP	ETC	RFC	MLP	ETC	RFC	MLP
Macro F1-score	95.83 ± 0.03	95.81 ± 0.03	94.70 ± 0.03	87.58 ± 0.04	87.77 ± 0.05	86.64 ± 0.05	**8.26**	8.04	8.07
Macro recall	94.04 ± 0.04	93.60 ± 0.05	94.77 ± 0.03	86.48 ± 0.04	86.40 ± 0.06	87.64 ± 0.06	7.56	7.19	7.13
Macro precision	99.40 ± 0.01	99.52 ± 0.01	95.65 ± 0.04	90.66 ± 0.03	90.81 ± 0.03	87.18 ± 0.04	8.75	8.71	8.47
Accuracy	99.47 ± 0.00	99.39 ± 0.00	99.60 ± 0.00	99.34 ± 0.00	99.25 ± 0.00	99.47 ± 0.01	0.14	0.14	0.12
Timeouts	(0.5, 60)	(0.5, 2)	(0.5, 60)	(10, 60)	(10, 60)	(3, 60)	–	–	–

Table 8
Multi-class classification results for UNSW-NB15 dataset using NFStream flow features (from 4 sources).

Metrics	Best timeouts			Worst timeouts			Performance difference		
	ETC	RFC	MLP	ETC	RFC	MLP	ETC	RFC	MLP
Macro F1-score	73.42 ± 0.02	73.15 ± 0.02	62.05 ± 0.02	72.79 ± 0.02	72.73 ± 0.02	60.04 ± 0.01	0.63	0.42	**2.01**
Macro recall	72.56 ± 0.01	72.21 ± 0.01	60.06 ± 0.01	71.64 ± 0.01	71.62 ± 0.01	58.54 ± 0.02	0.92	0.59	1.52
Macro precision	77.18 ± 0.07	76.90 ± 0.06	68.73 ± 0.06	76.47 ± 0.07	76.73 ± 0.07	69.85 ± 0.06	0.71	0.17	−1.11
Accuracy	98.77 ± 0.00	98.78 ± 0.00	98.72 ± 0.00	98.74 ± 0.00	98.76 ± 0.00	98.63 ± 0.00	0.03	0.01	0.10
Timeouts	(5, 60)	(2, 60)	(5, 5)	(0.5, 2)	(1, 30)	(4, 60)	–	–	–

we evaluate the ETC model with the two new feature sets and all combinations of the active and idle timeouts. We justify the selection of the ETC model to perform these evaluations by the fact that, overall, it consistently exhibits higher levels of effectiveness and stability in the previous set of experiments. Tables 10, 11, 12, and 13 presents the obtained best and worst measures when using the ETC model with feature selection strategy. We observe that the selection of features influences both the best and worst timeout. While the worst and best timeouts are consistent between 30% and 50% feature sets for USTC-TFC2016, they vary across the other datasets. Additionally, the application of the feature selection strategy has not yielded substantial improvements in model performance when compared with evaluations with all flow features. In fact, there are instances where implementing feature selection has resulted in a decline in the model's performance in terms of different metrics.

Given all the experiments done so far, we can answer the two research questions **RQ1** and **RQ2** related to the relationship between idle and active timeouts with flow features and model performances. Overall, we notice that the selection of flow timeouts influences the performance of machine learning models, with the impact being more significant for certain datasets (CICIDS2017) compared to others. Furthermore, we note that while performance improves with additional features, the optimal and suboptimal timeout values vary based on the features employed. However, while feature selection does not appear to enhance overall performance, it still influences which timeout values yield the best and worst results.

4.3.3. Measuring the relationship between the learning strategies and timeouts

In the next set of experiments, we explore a real-case scenario of distributed NIDS to answer the third research question **RQ3**. *FedAvg* algorithm was used to train a global model across 32 clients, each owning data extracted using a unique (T_i, T_a) combination. Fig. 6 provides a visual representation of accuracy and F1-score when evaluating the MLP classifier on the test set of each client as well as all test sets combined for USTC-TFC2016. The results for CICIDS2017, UNSW-NB15, and CUPID datasets are provided in Appendix B. Notably, the model exhibits varying performance levels across different clients. In other words, the performance of the global model significantly fluctuates from client to client as idle and active timeouts change, showing a difference of up to 25% in the F1-score. However, while the global model may not demonstrate any noticeable performance improvement on the entire test set of USTC-TFC2016, the results obtained from CICIDS2017, UNSW-NB15, and CUPID datasets (Figs. 10, 11, and 12 in Appendix B) suggest a significant potential for enhancement. The global model exhibits the capacity to achieve a performance level that, although falling short of being the absolute best, remains commendably satisfactory to a certain degree. These findings emphasize that the application of the *FedAvg* algorithm led to fairly good performance enhancement within distributed NIDS with different timeouts. Consequently, this outcome underscores the potential of federated learning and the need for further in-depth investigations and studies of federated learning algorithms that consider different NIDS with distinct characteristics resulting from different timeout configurations.

4.3.4. Explainability using SHAP

In order to better understand the impact of features on the model output when varying the values of flow timeouts, the SHAP (SHapley Additive exPlanations) approach [54] was utilized. SHAP is a game-theoretic framework that provides explanations for the outputs generated by any machine learning model. We utilized Shapley values to interpret the predictions made by the ETC model on the NFStream feature set (from the 4 sources) for both datasets USTC-TFC2016 and CICIDS2017. We identify the top 10 features that most significantly influence the model's predictions for the top (best performing) four timeouts, thereby offering insights into the underlying decision-making process. Fig. 7 provides a ranking of the ten most contributing features according to their Shapley values for 11-class (USTC-TFC2016) and 15-class (CICIDS2017) classification tasks (see Table 18 in Appendix A

Table 9
Multi-class classification results for CUPID dataset using NFStream flow features (from 4 sources).

Metrics	Best timeouts			Worst timeouts			Performance difference		
	ETC	RFC	MLP	ETC	RFC	MLP	ETC	RFC	MLP
Macro F1-score	96.35 ± 0.04	96.30 ± 0.04	95.55 ± 0.04	95.84 ± 0.04	95.88 ± 0.04	92.32 ± 0.07	0.50	0.41	**3.23**
Macro recall	97.01 ± 0.04	97.03 ± 0.04	96.76 ± 0.04	96.62 ± 0.05	96.85 ± 0.04	96.50 ± 0.05	0.39	0.18	0.26
Macro precision	97.45 ± 0.04	97.36 ± 0.04	96.35 ± 0.05	97.16 ± 0.04	96.74 ± 0.04	92.40 ± 0.09	0.29	0.61	3.95
Accuracy	96.82 ± 0.04	97.23 ± 0.04	97.65 ± 0.03	97.09 ± 0.04	96.63 ± 0.04	97.10 ± 0.03	−0.27	0.60	0.55
Timeouts	(10, 30)	(1, 60)	(0.5, 3)	(2, 3)	(5, 60)	(0.5, 30)	–	–	–

for the list of features). Different colors illustrate the contributions of features to each class. For USTC-TFC2016 (Fig. 7.1) we observe that:

- Features F3 (scr_port), F65 (udps.max_ttl), and F68 (udps.src2dst_flags) are the top three most important features in the ETC model, as indicated by their higher average SHAP values. These features greatly impact the model's output across the samples.
- Feature F3 is the most impactful feature, playing a significant role in predicting multiple classes especially *Neris* class. This suggests that F3 is highly informative across a wide variety of classes.
- Feature F65 also plays a crucial role, particularly influencing predictions for *Benign* class.
- Lower-ranked features (e.g., F70, F4) have less influence overall, but they still contribute to the predictions of certain classes. For instance, F70 (udps.tcp_flags) contributes to predictions for the *Benign* class, while F4 (dst_port) contributes to the *Geodo* and *Miuref* classes.

Similarly for CICIDS2017 (Fig. 7.2) dataset:

- Features F57 (application_confidence), F93 (udps.tcp_half_closed_time_ms), and F132 (udps.dst2src_small_packet_payload_ratio) are the top three most important features.
- F57 is most influential in predicting *Benign* and, *Portscan* and *Infiltration-Portscan*.
- Lower-ranked features like F70 (udps.tcp_flags) have less influence overall, but they still contribute to the detection of certain attack types such as *Portscan* and *DoS*.

The importance of features can shift depending on the timeout variations. For instance, in the USTC-TFC2016 dataset, F3 leads in early timeouts (10, 30) and (10, 60), but F65 becomes more significant in later timeouts (3, 3) and (3, 4). However, F57 maintains consistent importance across all timeouts in the second analysis on CICIDS2017. Moreover, feature importance can vary significantly from one dataset to another. While features like F3 and F65 dominate in one dataset, F57 and F93 take precedence in another, emphasizing the dataset-specific nature of feature contributions to the model's predictions. However, it is important to highlight that certain features, like F68, play a significant role across both datasets.

4.3.5. Training data size vs. model performance

Finding representative training data for ML-based NIDS is a well-known challenge in the field, as pointed out in prior research by Apruzzese et al. [55]. This section aims to examine the impact of varying amounts of training data on the performance of the ETC model and identify representative data for the best timeout. For each dataset, we consider the best timeout on the NFStream baseline feature set (identified in Section 4.3.2), and we incrementally increase the size of the training data while maintaining the test set constant. For each variation, we measured the accuracy and macro F1-score. We considered the following percentages: {10%, 20%, 30%, 40%, 50%, 60%, 70%, 80%, 90%, 95%, 97%, 98%, 100%}. The curves for USTC-TFC2016 are presented in Fig. 8, while the curves for CICIDS2017, UNSW-NB15, and CUPID are available in Appendix D.

For training set sizes below approximately 60%, the accuracy remains low, around 60% to 65%. This indicates that the model is unable to generalize effectively even when trained on more than half of the dataset size. After approximately 60% training set size, the accuracy increases rapidly, until reaching approximately 95%. This demonstrates that increasing the amount of training data significantly enhances the model's classification capability. Even with the best-performing timeout combination, the findings suggest that the model continues to have difficulty generalizing when only a small amount of data is available (see Fig. 8).

4.3.6. Results' analysis

On a more holistic level, we conducted one last analysis leveraging results from previous experiments (NFStream). The main idea is to evaluate whether a specific timeout tuple achieves better performance in comparison to the others. For each timeout tuple, we performed a total number of 36 experiments: (1) the first set of experiments contains 12 measures for each timeout tuple (4 datasets and 3 models, see Tables 2, 3, 4, and 5); (2) the second set of experiments consists of two main sets of experiments, first 12 experiments for each timeout (4 datasets and 3 models, see Tables 2, 3, 4, and 5) then 8 experiments which use one model with two different flow features (4 datasets and 2 flow feature sets, see Tables 10, 11, 12, and 13); (3) the last evaluations performed 4 experiments for each timeout tuple (4 datasets, see Fig. 6). This analysis is visually represented in the form of a boxplot illustrating the F1-scores for various (T_i, T_a) combinations, as depicted in Fig. 9. The latter shows a variation in model performance across the diverse timeout combinations, however, no specific timeout setting appears to stand out as the clear choice for an initial starting point. This finding underscores the complexity of selecting an ideal timeout combination right from the outset.

Table 10
Multi-class classification results for USTC-TFC16 dataset using feature selection.

Metrics	Best timeouts		Worst timeouts	
	ETC (30%)	ETC (50%)	ETC (30%)	ETC (50%)
Macro F1-score	95.40	**95.51**	92.95	92.90
Macro recall	96.57	96.71	93.44	93.42
Macro precision	95.35	95.41	93.82	93.74
Accuracy	97.16	97.19	95.74	95.72
Timeouts	(10, 60)	**(10, 60)**	(0.5, 60)	(0.5, 60)

Table 11
Multi-class classification results for CICIDS2017 dataset using feature selection.

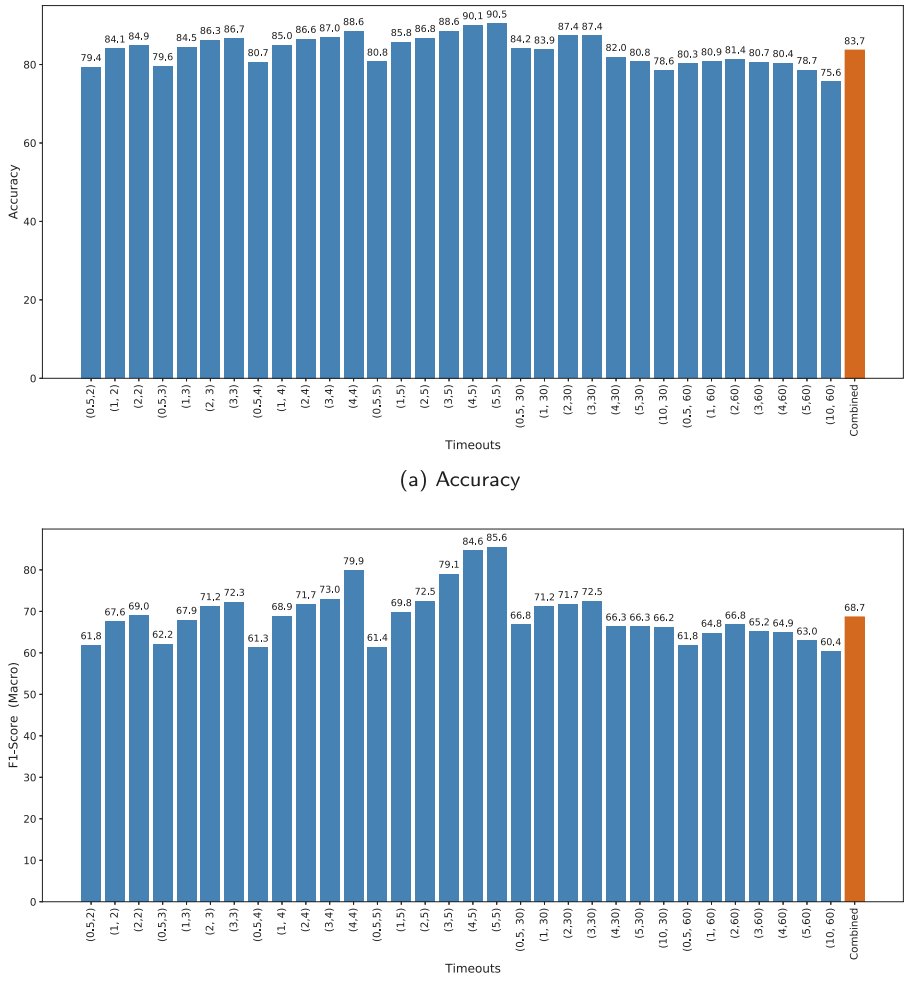
Metrics	Best timeouts		Worst timeouts	
	ETC (30%)	ETC (50%)	ETC (30%)	ETC (50%)
Macro F1-score	**95.82**	95.30	89.79	87.25
Macro recall	94.02	93.42	88.25	85.77
Macro precision	99.23	99.19	93.34	90.75
Accuracy	99.40	99.43	99.32	99.34
Timeouts	**(4, 4)**	(1, 30)	(10, 60)	(10, 60)

Table 12
Multi-class classification results for UNSW-NB15 dataset using feature selection.

Metrics	Best timeouts		Worst timeouts	
	ETC (30%)	ETC (50%)	ETC (30%)	ETC (50%)
Macro F1-score	**74.61**	74.21	73.54	73.59
Macro recall	74.64	73.69	73.98	72.85
Macro precision	78.27	78.22	77.79	77.79
Accuracy	98.77	98.77	98.75	98.75
Timeouts	**(3, 4)**	(2, 4)	Default	(0.5, 5)

Table 13
Multi-class classification results for CUPID dataset using feature selection.

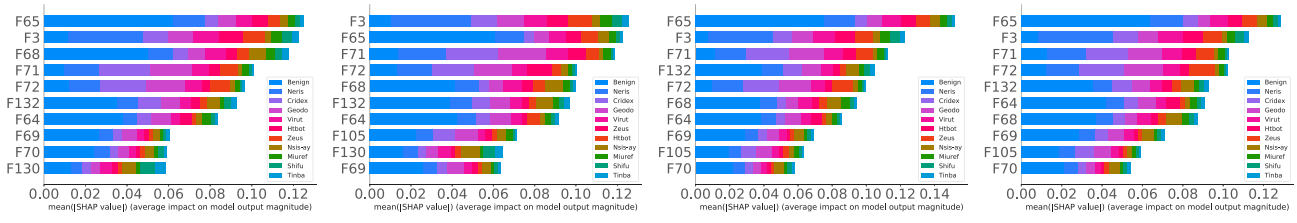
Metrics	Best timeouts		Worst timeouts	
	ETC (30%)	ETC (50%)	ETC (30%)	ETC (50%)
Macro F1-score	96.30	**96.38**	95.90	95.94
Macro recall	97.01	97.03	96.80	96.88
Macro precision	97.36	97.54	96.84	96.84
Accuracy	97.33	97.21	97.04	96.78
Timeouts	(2, 4)	**(1, 60)**	(1, 5)	(5, 60)



(a) Accuracy

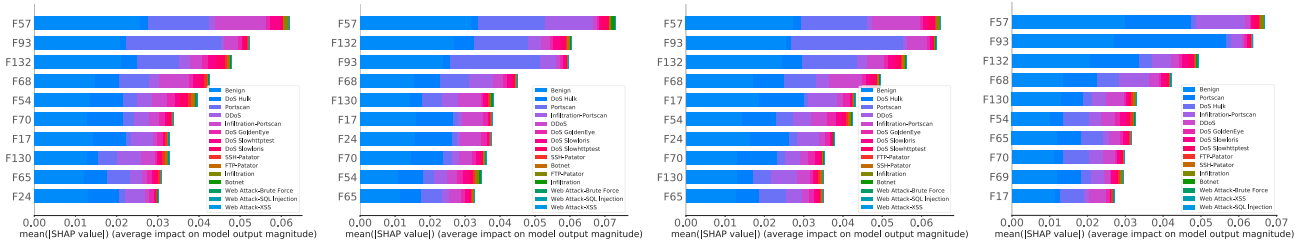
(b) Macro F1-score

Fig. 6. Performance evaluation of federated learning for distributed NIDS with different timeouts on USTC-TFC2016 dataset.



(a) (10, 30) (b) (10, 60) (c) (3, 3) (d) (3, 4)

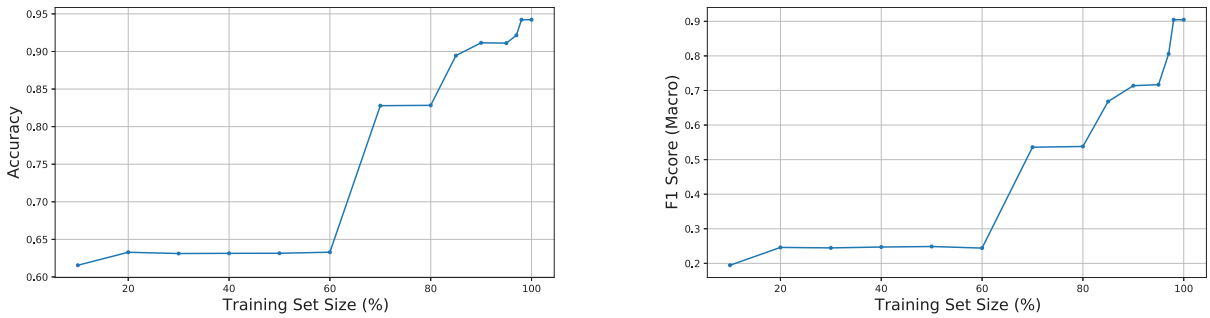
(1) USTC-TFC2016 Dataset



(a) (0.5, 60) (b) (0.5, 4) (c) (0.5, 30) (d) (0.5, 2)

(2) CICIDS2017 Dataset

Fig. 7. SHAP Summary Plots for Extra Trees Classifier: Feature importance of each feature in the model Across top four best timeout combinations on both USTC-TFC2016 and CICIDS2017 datasets.



(a) Accuracy vs Training Set Size (b) F1-Score vs Training Set Size

Fig. 8. Accuracy and F1-score of ETC model vs. training set size for USTC-TFC2016 Dataset with best timeout (10, 60).

4.3.7. Recommendations

In light of these results, it is evident that the choice of timeout can influence NIDS ML models' performance by as much as 8.77% (Table 7). On the one hand, the ETC model is stable and scores better results, On the other hand, the MLP model is the most impacted by timeout selection, this is important to notice given the current deep learning landscape. Fig. 9 provides an answer to **RQ4** and shows that no one-size-fits-all timeout tuple exists, and the choice depends on the environment of the application (dataset) and the ML model. Consequently, incorporating a fine-tuning stage into the model development process mitigates the risk of settling for an inefficient timeout thereby enhancing the overall effectiveness of the ML-based network intrusion detection.

5. Related work

Since our experiments have exclusively focused on the case of NIDS, this section will only encompass research works related to NIDS.

Network-based datasets play a crucial role in both the development and validation of NIDS. They enable the comparison of various NIDS solutions, assisting security professionals in selecting the most suitable options for their specific environments. Nevertheless, because of the challenges involved in acquiring realistic labeled network data flows, researchers have developed benchmark NIDS datasets by simulating network behaviors in controlled testbed environments [56]. Using appropriate tools and frameworks, network flows are generated through several phases including data collection, labeling, and feature extraction. Nevertheless, multiple investigations revealed that NIDS datasets exhibit various deficiencies within one or more of these phases.

Vormayr et al. [35] highlighted significant variations in the results generated by different flow exporters (go-flows, argus, pmacct, Vermont, yaf, and Joy), which can have a profound impact on the accuracy and reliability of subsequent analyses. These discrepancies stem from the unique ways each exporter handles flow data, which can affect both the completeness and the structure of the collected information. In their paper, Sarhan et al. [57] assessed the performance of ML-based NIDS models by comparing two feature sets (NetFlow and CICFlowMeter) across three key datasets, demonstrating the superiority of the NetFlow-based feature set in terms of detection accuracy. Additionally, the authors argued the importance of standardization in feature sets across NIDS datasets, as it allows for the evaluation of ML model generalizability in various network environments. Moreover, the SHAP method (SHapley Additive exPlanations) was also used to explain and interpret classification decisions in the ML-based NIDS models utilizing both feature sets.

Engelen et al. [58] identified some major flaws in the widely-used dataset CICIDS2017, related to flow construction and feature extraction process. In their attempt to fix these issues, the authors released an update of CICFlowMeter. CSE-CIC-IDS-2018 dataset[9] which is considered a reliable and updated version of CICIDS2017 was also found to be flawed [59] through a series of experiments. These issues are mainly related to feature generation and labeling.

Sarhan et al. [36] highlighted some limitations of existing NIDS datasets, particularly, the lack of a common ground feature set shared

[9] https://www.unb.ca/cic/datasets/ids-2018.html.

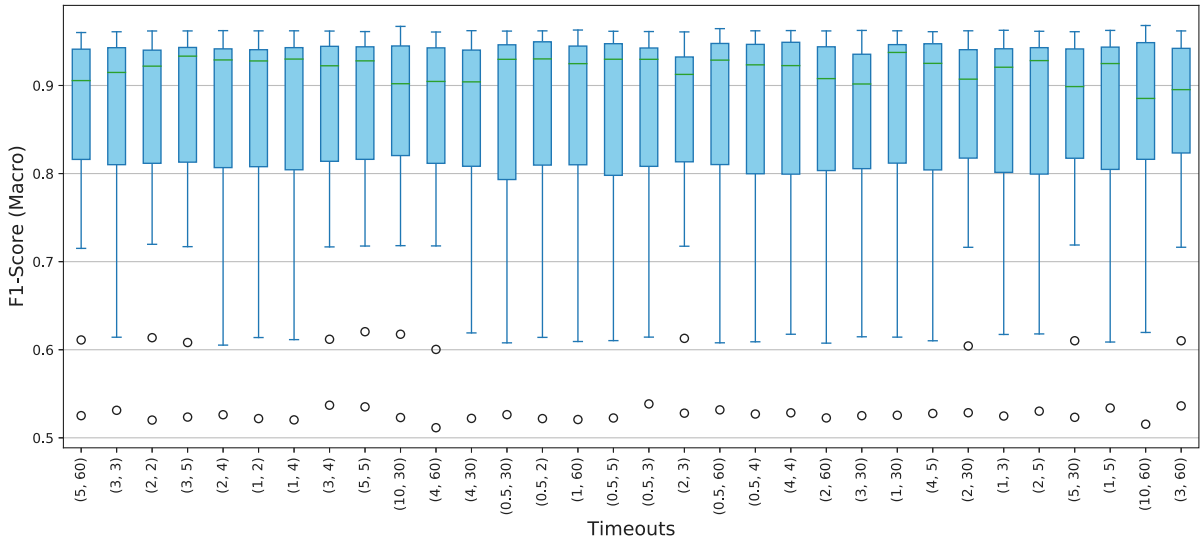


Fig. 9. Boxplot depicting F1-score variation of different timeouts considering all preceding experiments (NFStream).

among datasets. This makes it difficult to fairly evaluate ML models' performance and their generalization ability across different datasets. To overcome this limitation, the authors proposed to use a standardized feature set that can be used by researchers to train and cross-evaluate various NIDS models. Four datasets were created by converting synthetic NIDS datasets from their PCAP files to the standard NetFlow format with a total of 12 shared features. To evaluate the new datasets, an extra tree ensemble classifier was applied across the four datasets. The results showed that higher performance is achieved for the binary-class classification scenario. However, inferior results were obtained in the second scenario of multi-class classification. This is due to the inadequate security information provided by the standard NetFlow features. To improve the performance, the authors proposed a second version of the same beforementioned datasets with an extended NetFlow feature set with 43 features [21]. Similarly, an extra trees classifier was used to evaluate the new version of each dataset. The proposed NetFlow feature set led to a significant improvement in multi-class classification performance. The studies and analyses discussed reveal that the publically available NIDS datasets have various issues related to feature extraction and selection, that can significantly impact the quality and performance of NIDS. As NIDS rely on these datasets for training and evaluation, the presence of flaws can hinder their ability to accurately detect and respond to security threats. Therefore, identifying and addressing these dataset limitations and enhancing their quality is crucial for the development and improvement of NIDS systems. In this work, we shed light on deficiencies of NIDS datasets from a different angle. We delve into an exploration of how idle and active timeout parameters affect the quality of extracted features and, consequently, the performance of machine learning models.

6. Conclusion

This work aims to shed light on the relationship between flow timeout and the performance of machine learning models. Through extensive experimentation, we investigate the impact of varying the values of idle and active timeouts on the quality of the extracted features and therefore on ML models used in the context of NIDS. Our findings demonstrate that adopting a one-size-fits-all approach to timeouts can lead to suboptimal performance of machine learning models. Specifically, we found that different combinations of idle and active timeouts had a substantial impact on the accuracy and performance of the NIDS models, with a significant difference in performance between the most effective timeout combination and the least effective. Furthermore, the results indicated that the ETC and RFC models exhibit greater stability and resilience to fluctuations in idle and active timeouts compared to the MLP model. Notably, ETC also exhibited superior performance when used with longer timeouts. The study also revealed that extending the feature set reduces the variance in models' performance considering different timeouts. Through the application of federated learning, we also demonstrate its promising potential in handling distributed NIDS with diverse timeout configurations.

Our analysis provides valuable insights into the impact of flow timeout on the performance of NIDS, underscoring the need for network security professionals and data scientists to pay meticulous attention to the selection and configuration of idle and active timeouts when training machine learning models for NIDS. Our future research will further examine the impact of timeouts on different network security systems (Network Security Monitoring, Security Information and Event Management, etc.). In addition, we plan to explore heterogeneous federated learning algorithms to enhance the performance of machine learning-based NIDS when different configurations of timeouts are used.

CRediT authorship contribution statement

Meryem Janati Idrissi: Writing – original draft, Validation, Methodology, Investigation, Data curation, Conceptualization. **Hamza Alami:** Writing – review & editing, Validation, Supervision, Methodology. **Abdelkader El Mahdaouy:** Writing – review & editing, Resources, Methodology. **Abdelhak Bouayad:** Writing – review & editing. **Zakaria Yartaoui:** Writing – review & editing. **Ismail Berrada:** Writing – review & editing, Supervision, Project administration.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Meryem Janati Idrissi reports equipment, drugs, or supplies was provided by Mohammed VI Polytechnic University.

Acknowledgments

Experiments presented in this study were provided by the computing facilities of the High Performance Computing simlab-cluster, of Mohammed VI Polytechnic University at Benguerir.

Appendix A. Features

We provide the set of features presented in both works [21,37]. Table 14 presents the features selected in the paper [21]. Table 15 describes the flow features we used and presented in [37]. We also present the list of features extracted by Zeek and Argus, as shown in Tables 16 and 17, respectively, and utilized in this study.

Appendix B. Federated learning for distributed NIDS with different timeouts

Here, we provide more results of evaluating federated learning for distributed NIDS, each configured with different timeouts. Figs. 10, 11, and 12 depicts the results for CICIDS2017, UNSW-NB15, and CUPID datasets respectively.

Appendix C. Explainability using SHAP

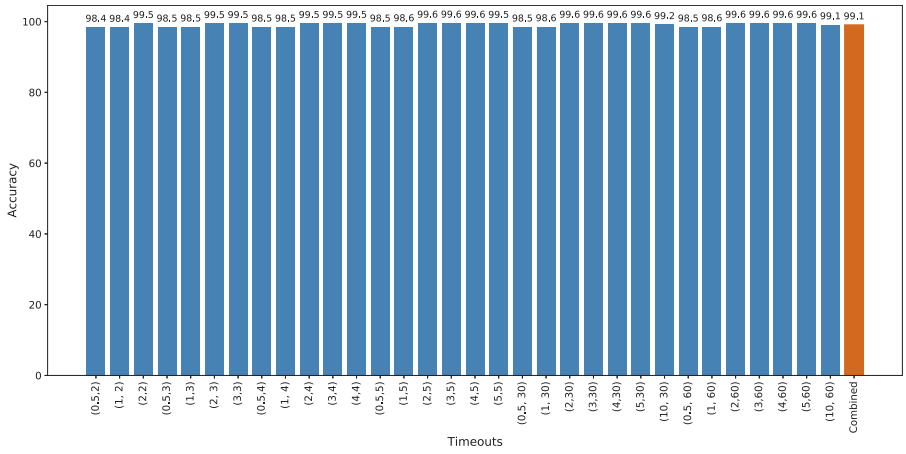
In this section, we provide in Table 18, the corresponding names of the features' IDs used in Section 4.3.4.

Appendix D. Training data size vs. model performance

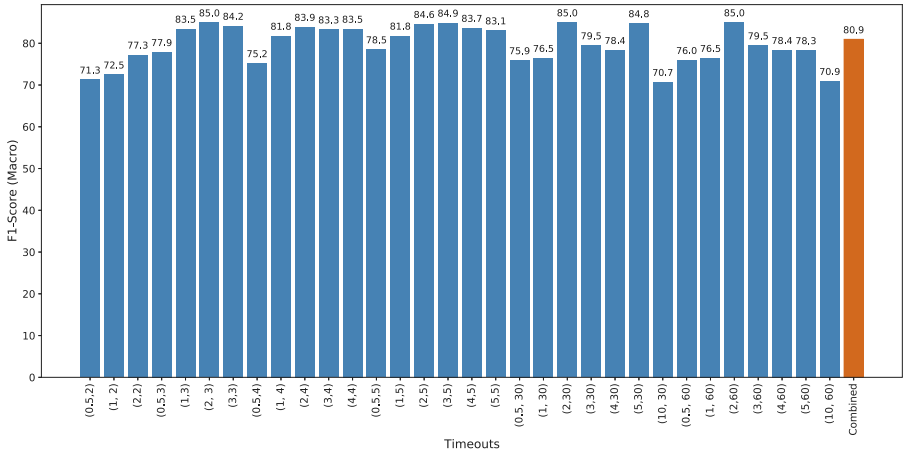
In this section, we present the accuracy and macro F1-score curves for the CICIDS2017, UNSW-NB15, and CUPID datasets (see Figs. 13–15). Unlike USTC-TFC2016, the performance of the model on CICIDS2017 reaches approximately 85% accuracy with just 20% of the data, which can be attributed to the significantly larger size of the CICIDS2017 dataset compared to USTC-TFC2016. Similar observations can be made for UNSW-NB15. In the case of CUPID, the model performs well with less than 20% of the data and continues to improve, ultimately achieving 97% accuracy.

Data availability

No data was used for the research described in the article.



(a) Accuracy

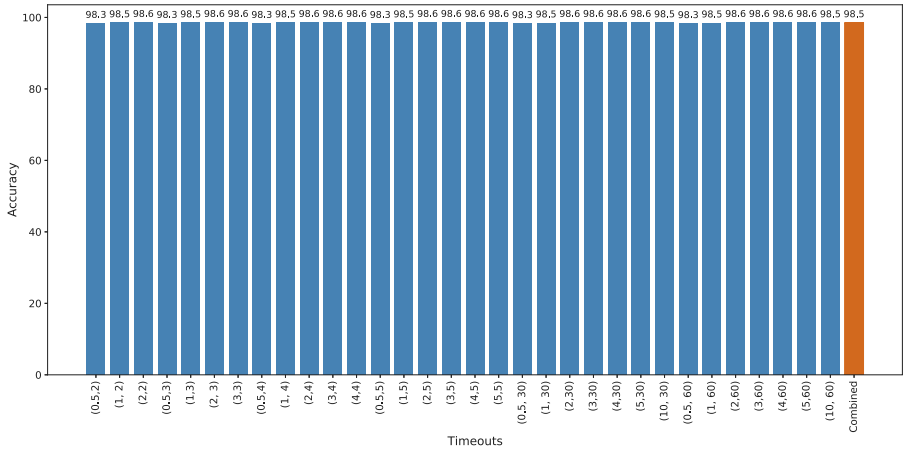


(b) Macro F1-score

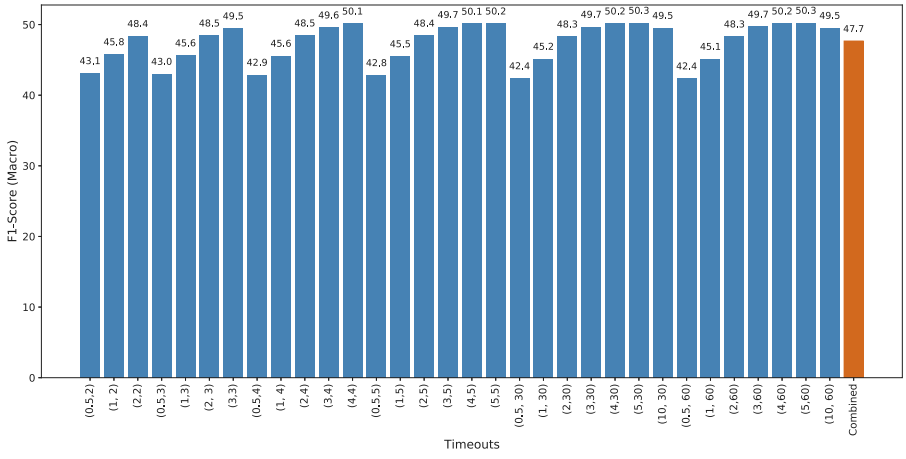
Fig. 10. Performance evaluation of federated learning for distributed NIDS with different timeouts on CICIDS2017 dataset.

Table 14
Set of features proposed in [21].

Feature	Description
tcp_flags	Cumulative of all flow TCP flags
src2dst_flags	Cumulative of all client TCP flags
dst2src_flags	Cumulative of all server TCP flags
min_ttl	Min flow TTL
max_ttl	Max flow TTL
min_ip_pkt_len	Len of the smallest flow IP packet observed
max_ip_pkt_len	Len of the largest flow IP packet observed
src_to_dst_second_bytes	Bytes/sec (src->dst)
dst_to_src_second_bytes	Bytes/sec (dst->src)
retransmitted_in_bytes	Number of retransmitted TCP flow bytes (src->dst)
retransmitted_in_packets	Number of retransmitted TCP flow packets (src->dst)
retransmitted_out_bytes	Number of retransmitted TCP flow bytes (dst->src)
retransmitted_out_packets	Number of retransmitted TCP flow packets (dst->src)
src_to_dst_avg_throughput	Src to dst average that (bps)
dst_to_src_avg_throughput	Dst to src average that (bps)
num_pkts_up_to_128_bytes	packets whose IP size $\leq$ 128
num_pkts_128_to_256_bytes	packets whose IP size $>$ 128 and $\leq$ 256
num_pkts_256_to_512_bytes	packets whose IP size $>$ 256 and $<$ 512
num_pkts_512_to_1024_bytes	packets whose IP size $>$ 512 and $<$ 1024
num_pkts_1024_to_1514_bytes	packets whose IP size $>$ 1024 and $\leq$ 1514
tcp_win_max_in	Max TCP Window (src->dst)
tcp_win_max_out	Max TCP Window (dst->src)
icmp_type	ICMP Type * 256 + ICMP code
icmp_v4_type	ICMP Type
dns_query_id	DNS query transaction Id
dns_query_type	"DNS query type (e.g., 1 = A, 2 = NS..)"
dns_ttl_answer	TTL of the first A record (if any)
ftp_command_ret_code	FTP client command return code

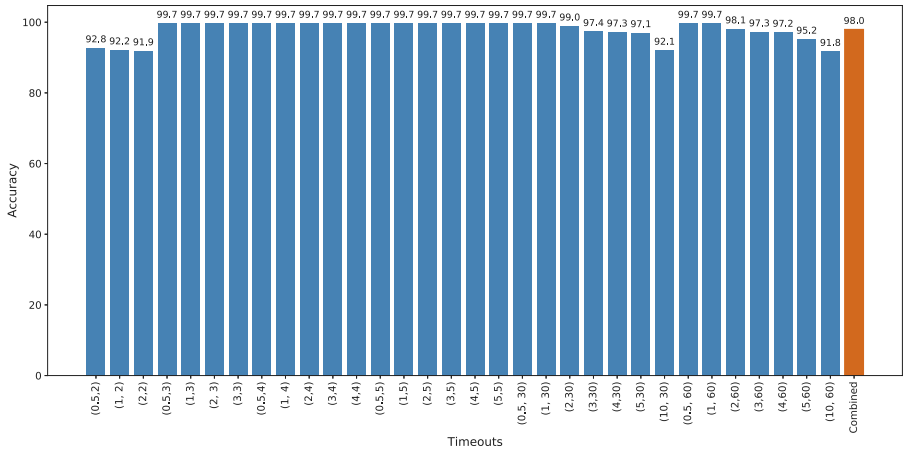


(a) Accuracy

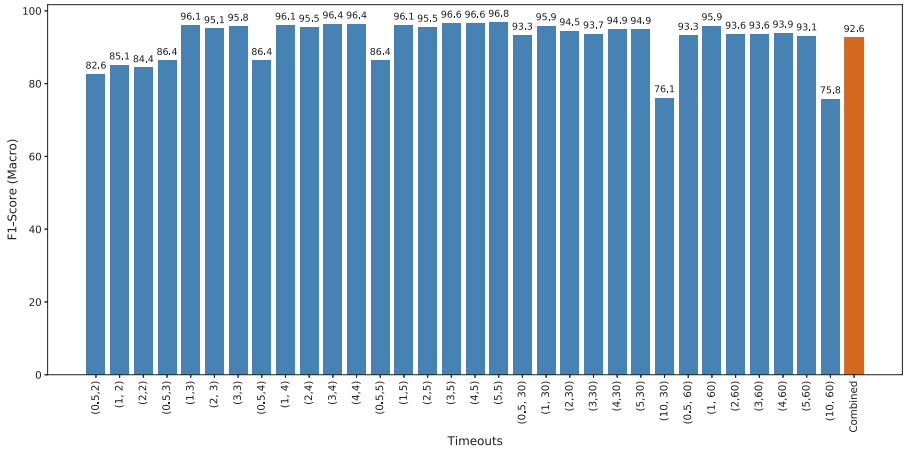


(b) Macro F1-score

Fig. 11. Performance evaluation of federated learning for distributed NIDS with different timeouts on UNSW-NB15 dataset.

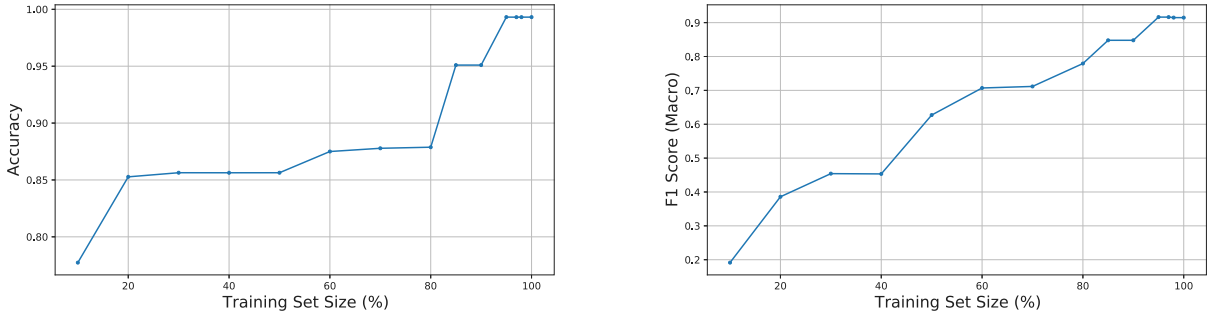


(a) Accuracy



(b) Macro F1-score

Fig. 12. Performance evaluation of federated learning for distributed NIDS with different timeouts on CUPID dataset.

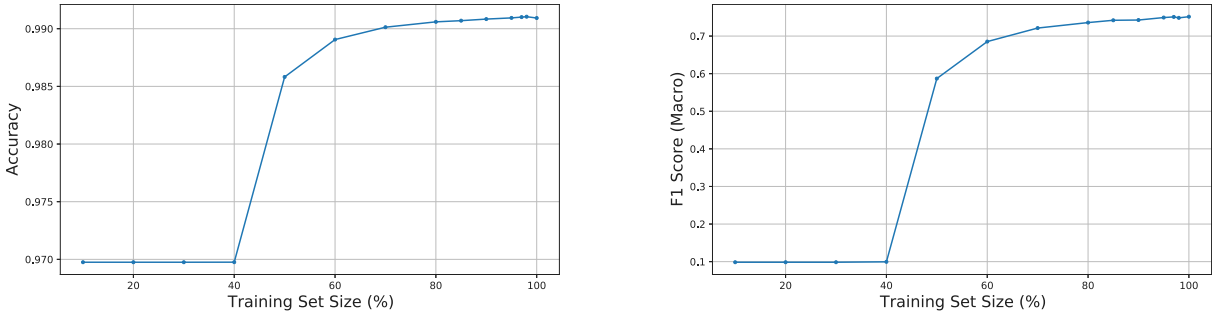


(a) Accuracy vs Training Set Size (b) F1-Score vs Training Set Size

Fig. 13. Accuracy and F1-score of ETC model vs. training set size for CICIDS2017 Dataset with best timeout (0.5, 60).

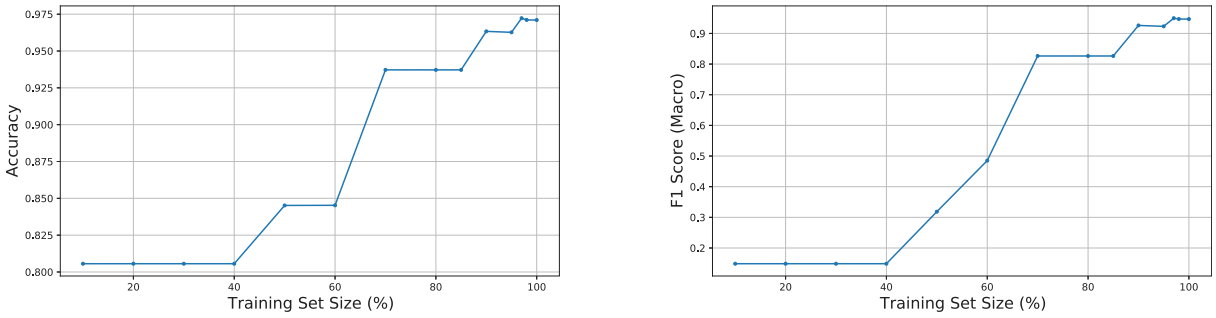
Table 15
Set of features proposed in [37].

Features	Description
src2dst_first_packet_payload_len	First packet's payload length src2dst direction
dst2src_first_packet_payload_len	First packet's payload length dst2src direction
src2dst_most_freq_payload_ratio	The ratio of number of packets with most freq payload len for direction src2dst to the total number of packets in direction src2dst
src2dst_most_freq_payload_len	The number of packets with most freq payload len for direction src2dst
dst2src_most_freq_payload_ratio	The ratio of number of packets with most freq payload len for direction dst2src to the total number of packets in direction src2dst
dst2src_most_freq_payload_len	The number of packets with most freq payload len for direction dst2src
bidirectional_mean_packet_relative_times	The average timestamp in milliseconds between first flow bidirectional packet and all other flow bidirectional packets.
bidirectional_stddev_packet_relative_times	The standard deviation timestamp in milliseconds between first flow bidirectional packet and all other flow bidirectional packets.
bidirectional_variance_packet_relative_times	The variance timestamp in milliseconds between first flow bidirectional packet and all other flow bidirectional packets.
bidirectional_coeff_of_var_packet_relative_times	The coefficient of variance (std/avg) of the difference between first flow bidirectional packet and all other flow bidirectional packets.
bidirectional_skew_from_median_packet_relative_times	The skewness from median (3×(avg-median)/std) of the difference between first flow bidirectional packet and all other flow bidirectional packets.
src2dst_mean_packet_relative_times	The average timestamp in milliseconds between first flow bidirectional packet and all other flow packets with direction src->dst.
src2dst_stddev_packet_relative_times	The standard deviation timestamp in milliseconds between first flow bidirectional packet and all other flow packets with direction src->dst.
src2dst_variance_packet_relative_times	The variance timestamp in milliseconds between first flow bidirectional packet and all other flow packets with direction src->dst.
src2dst_coeff_of_var_packet_relative_times	The coefficient of variance (std/avg) of the difference between first flow bidirectional packet and all other flow packets with direction src->dst.
src2dst_skew_from_median_packet_relative_times	The skewness from median (3×(avg-median)/std) of the difference between first flow bidirectional packet and all other flow packets with direction src->dst.
dst2src_mean_packet_relative_times	The average timestamp in milliseconds between first flow bidirectional packet and all other flow packets with direction dst->src.
dst2src_stddev_packet_relative_times	The standard deviation timestamp in milliseconds between first flow bidirectional packet and all other flow packets with direction dst->src.
dst2src_variance_packet_relative_times	The variance timestamp in milliseconds between first flow bidirectional packet and all other flow packets with direction dst->src.
dst2src_coeff_of_var_packet_relative_times	The coefficient of variance (std/avg) of the difference between first flow bidirectional packet and all other flow packets with direction dst->src.
dst2src_skew_from_median_packet_relative_times	The skewness from median (3×(avg-median)/std) of the difference between first flow bidirectional packet and all other flow packets with direction dst->src.
min_req_res_time_diff	The minimum timestamp difference between the request and its response (delta time between packet from source and the next packet from dst or vice versa).
max_req_res_time_diff	The maximum timestamp difference between the request and its response (delta time between packet from source and the next packet from dst or vice versa)
mean_req_res_time_diff	The mean of all timestamp difference between the request and its response (delta time between packet from source and the next packet from dst or vice versa)
median_req_res_time_diff	The median of all timestamp difference between the request and its response (delta time between packet from source and the next packet from dst or vice versa)
stddev_req_res_time_diff	The standard deviation of all timestamp difference between the request and its response (timestamp between packet from source and the next packet from dst or vice versa)
variance_req_res_time_diff	The variance of all timestamp difference between the request and its response (timestamp between packet from source and the next packet from dst or vice versa)
coeff_of_var_req_res_time_diff	The coefficient of variance (std/avg) of all timestamp difference between the request and its response (timestamp between packet from source and the next packet from dst or vice versa).
skew_from_median_req_res_time_diff	The skewness from median (3×(avg-median)/std) of all timestamp difference between the request and its response (timestamp between packet from source and the next packet from dst or vice versa).
src2dst_small_packet_payload_packets	The number of src2dst packets with a small size (< 32 bytes)
src2dst_small_packet_payload_ratio	The ratio of small src2dst packets (<32 bytes) to the number of all src2dst packets.
dst2src_small_packet_payload_packets	The number of dst2src packets with a small size (< 32 bytes)
dst2src_small_packet_payload_ratio	The ratio of small dst2src packets (<32 bytes) to the number of all src2dst packets.
sent_recv_packet_ratio	The ratio of the number of received packets to the number of sent packets = recv/sent.
bidirectional_ps_first_quartile	The first quartile of flow packets size (link layer packet size).
bidirectional_ps_second_quartile	The second quartile of flow packets size (link layer packet size).
bidirectional_ps_third_quartile	The third quartile of flow packets size (link layer packet size).
bidirectional_ps_median_absoulte_deviation	The median of the absolute difference between packets size and the median (median × | packet_size - median |)
bidirectional_ps_skewness	The skewness of flow packets size (link layer packet size).
bidirectional_ps_kurtosis	The kurosis of flow packets size (link layer packet size).
bidirectional_piat_first_quartile	The first quartile of flow packets delta time (Delta time in milliseconds with previous flow packet).
bidirectional_piat_second_quartile	The second quartile of flow packets delta time (Delta time in milliseconds with previous flow packet).
bidirectional_piat_third_quartile	The third quartile of flow packets delta time (Delta time in milliseconds with previous flow packet).
bidirectional_piat_median_absoulte_deviation	The median of the absolute difference between flow packets delta time and the median (median × | packet_size - median |)
bidirectional_piat_skewness	The skewness of flow packets delta time.
bidirectional_piat_kurtosis	The kurtosis of flow packets delta time.



(a) Accuracy vs Training Set Size (b) F1-Score vs Training Set Size

Fig. 14. Accuracy and F1-score of ETC model vs. training set size for UNSW-NB15 Dataset with best timeout (4, 5).



(a) Accuracy vs Training Set Size (b) F1-Score vs Training Set Size

Fig. 15. Accuracy and F1-score of ETC model vs. training set size for CUPID Dataset with best timeout (3, 3).

Table 16
The list of Zeek/Bro-IDS flow-based statistical features used in this paper.

Feature	Description
ts	Timestamp of the start of the connection
id.orig_p	Source port of the originator
id.resp_p	Destination port of the responder
proto	Transport layer protocol used (e.g., TCP, UDP)
service	Application layer protocol detected (e.g., HTTP, DNS)
duration	Duration of the connection in seconds
orig_bytes	Number of bytes sent by the originator
resp_bytes	Number of bytes sent by the responder
conn_state	Connection state (e.g., established, reset, closed)
local_orig	Indicator if the originator is part of the local network
local_resp	Indicator if the responder is part of the local network
missed_bytes	Number of bytes missed in the capture
history	History of TCP connection flags (e.g., SYN, FIN, ACK)
orig_pkts	Number of packets sent by the originator
orig_ip_bytes	Total number of IP layer bytes sent by the originator
resp_pkts	Number of packets sent by the responder
resp_ip_bytes	Total number of IP layer bytes sent by the responder

Table 17
The list of Argus flow-based statistical features used in this paper.

Feature	Description
Dur	Duration of the flow in milliseconds
RunTime	Time in milliseconds during which the flow was active
IdleTime	Time in milliseconds the flow was idle (no packets transmitted)
Mean	Mean packet size for the flow
StdDev	Standard deviation of packet sizes in the flow
Sum	Total sum of all packet sizes in the flow
Min	Minimum packet size in the flow
Max	Maximum packet size in the flow
Proto	Protocol used in the flow (e.g., TCP, UDP)
Cause	Reason for the flow termination
TotPkts	Total number of packets transmitted in the flow
SrcPkts	Number of packets transmitted by the source
DstPkts	Number of packets transmitted by the destination
TotBytes	Total number of bytes transmitted in the flow
SrcBytes	Number of bytes transmitted by the source
DstBytes	Number of bytes transmitted by the destination
Load	Average bytes per second over the duration of the flow
SrcLoad	Average bytes per second transmitted by the source
DstLoad	Average bytes per second transmitted by the destination
Rate	Packet rate per second over the duration of the flow
SrcRate	Packet rate per second from the source
DstRate	Packet rate per second from the destination
SAppBytes	Number of application layer bytes sent by the source
DAppBytes	Number of application layer bytes sent by the destination
SynAck	Round-trip time for the SYN-ACK handshake
AckDat	Round-trip time between ACK and data packet
TcpRtt	Estimated round-trip time for TCP flow

Table 18
Feature IDs and their corresponding names.

Feature ID	Feature name	Feature ID	Feature name
F3	src_port	F130	udps.src2dst_small_packet_payload_ratio
F65	udps.max_ttl	F69	udps.dst2src_flags
F68	udps.src2dst_flags	F70	udps.tcp_flags
F71	udps.tcp_win_max_in	F4	dst_port
F72	udps.tcp_win_max_out	F105	udps.dst2src_most_freq_payload_ratio
F132	udps.dst2src_small_packet_payload_ratio	F57	application_confidence
F64	udps.min_ttl	F93	udps.tcp_half_closed_time_ms
F69	udps.dst2src_flags	F54	application_name
F70	udps.tcp_flags	F17	bidirectional_stddev_ps
F24	dst2src_mean_ps	F25	dst2src_stddev_ps
F65	udps.max_ttl	F130	udps.src2dst_small_packet_payload_ratio

References

[1] H. Saleous, M. Ismail, S.H. AlDaajeh, N. Madathil, S. Alrabaee, K.-K.R. Choo, N. Al-Qirim, COVID-19 pandemic and the cyberthreat landscape: Research challenges and opportunities, Digit. Commun. Netw. 9 (1) (2023) 211–222, http://dx.doi.org/10.1016/j.dcan.2022.06.005, URL: https://www.sciencedirect.com/science/article/pii/S2352864822001274.

[2] M. Fuentes-García, J. Camacho, G. Maciá-Fernández, Present and future of network security monitoring, IEEE Access 9 (2021) 112744–112760, http://dx.doi.org/10.1109/ACCESS.2021.3067106.

[3] G. González-Granadillo, S. González-Zarzosa, R. Diaz, Security Information and Event Management (SIEM): Analysis, trends, and usage in critical infrastructures, Sensors 21 (14) (2021) http://dx.doi.org/10.3390/s21144759, URL: https://www.mdpi.com/1424-8220/21/14/4759.

[4] Z. Ahmad, A. Shahid Khan, C. Wai Shiang, J. Abdullah, F. Ahmad, Network intrusion detection system: A systematic study of machine learning and deep learning approaches, Trans. Emerg. Telecommun. Technol. 32 (1) (2021) e4150, http://dx.doi.org/10.1002/ett.4150.

[5] Z. Yang, X. Liu, T. Li, D. Wu, J. Wang, Y. Zhao, H. Han, A systematic literature review of methods and datasets for anomaly-based network intrusion detection, Comput. Secur. 116 (2022) 102675, http://dx.doi.org/10.1016/j.cose.2022.102675, URL: https://www.sciencedirect.com/science/article/pii/S0167404822000736.

[6] A. Sperotto, G. Schaffrath, R. Sadre, C. Morariu, A. Pras, B. Stiller, An overview of IP flow-based intrusion detection, IEEE Commun. Surv. Tutor. 12 (3) (2010) 343–356, http://dx.doi.org/10.1109/SURV.2010.032210.00054.

[7] C.F.T. Pontes, M.M.C. de Souza, J.J.C. Gondim, M. Bishop, M.A. Marotta, A new method for flow-based network intrusion detection using the inverse potts model, IEEE Trans. Netw. Serv. Manag. 18 (2) (2021) 1125–1136, http://dx.doi.org/10.1109/TNSM.2021.3075503.

[8] L.G. Nguyen, K. Watabe, Flow-based network intrusion detection based on BERT masked language model, in: Proceedings of the 3rd International CoNEXT Student Workshop, in: CoNEXT-SW '22, Association for Computing Machinery, New York, NY, USA, 2022, pp. 7–8, http://dx.doi.org/10.1145/3565477.3569152.

[9] V. Jyothsna, D. Mukesh, A.N. Sreedhar, A flow-based network intrusion detection system for high-speed networks using meta-heuristic scale, in: S.-L. Peng, N. Dey, M. Bundele (Eds.), Computing and Network Sustainability, Springer Singapore, Singapore, 2019, pp. 337–347.

[10] G. Sadasivan, N. Brownlee, B. Claise, J. Quittek, Architecture for IP Flow Information Export, Technical Report, 2009.

[11] R. Hofstede, P. Čeleda, B. Trammell, I. Drago, R. Sadre, A. Sperotto, A. Pras, Flow monitoring explained: From packet capture to data analysis with netflow and IPFIX, IEEE Commun. Surv. Tutor. 16 (4) (2014) 2037–2064, http://dx.doi.org/10.1109/COMST.2014.2321898.

[12] S. Layeghy, M. Baktashmotlagh, M. Portmann, DI-NIDS: Domain invariant network intrusion detection system, Knowl.-Based Syst. 273 (2023) 110626, http://dx.doi.org/10.1016/j.knosys.2023.110626, URL: https://www.sciencedirect.com/science/article/pii/S0950705123003763.

[13] P.-J. Chuang, P.-Y. Huang, Enhancing network intrusion detection by lifelong active online learning, J. Supercomput. 80 (11) (2024) 16428–16451, http://dx.doi.org/10.1007/s11227-024-06070-4.

[14] T. Yi, X. Chen, Y. Zhu, W. Ge, Z. Han, Review on the application of deep learning in network attack detection, J. Netw. Comput. Appl. 212 (2023) 103580, http://dx.doi.org/10.1016/j.jnca.2022.103580, URL: https://www.sciencedirect.com/science/article/pii/S1084804522002211.

[15] A. Tabassum, A. Erbad, W. Lebda, A. Mohamed, M. Guizani, Fedgan-IDS: Privacy-preserving IDS using GAN and federated learning, Comput. Commun. 192 (2022) 299–310, http://dx.doi.org/10.1016/j.comcom.2022.06.015, URL: https://www.sciencedirect.com/science/article/pii/S0140366422002171.

[16] Y. Mirsky, T. Doitshman, Y. Elovici, A. Shabtai, Kitsune: An ensemble of autoencoders for online network intrusion detection, 2018, CoRR abs/1802.09089, arXiv:1802.09089, URL: http://arxiv.org/abs/1802.09089.

[17] M.J. Idrissi, H. Alami, A. El Mahdaouy, A. El Mekki, S. Oualil, Z. Yartaoui, I. Berrada, Fed-anids: Federated learning for anomaly-based network intrusion detection systems, Expert Syst. Appl. 234 (2023) 121000, http://dx.doi.org/10.1016/j.eswa.2023.121000, URL: https://www.sciencedirect.com/science/article/pii/S0957417423015026.

[18] W. Wang, S. Jian, Y. Tan, Q. Wu, C. Huang, Robust unsupervised network intrusion detection with self-supervised masked context reconstruction, Comput. Secur. 128 (2023) 103131, http://dx.doi.org/10.1016/j.cose.2023.103131, URL: https://www.sciencedirect.com/science/article/pii/S016740482300041X.

[19] L. Budach, M. Feuerpfeil, N. Ihde, A. Nathansen, N. Noack, H. Patzlaff, F. Naumann, H. Harmouch, The effects of data quality on machine learning performance, 2022, arXiv:2207.14529.

[20] Y. Zhou, G. Cheng, S. Jiang, M. Dai, Building an efficient intrusion detection system based on feature selection and ensemble classifier, Comput. Netw. 174 (2020) 107247, http://dx.doi.org/10.1016/j.comnet.2020.107247, URL: https://www.sciencedirect.com/science/article/pii/S1389128619314203.

[21] M. Sarhan, S. Layeghy, N. Moustafa, M. Portmann, Towards a standard feature set of NIDS datasets, 2021, CoRR abs/2101.11315, arXiv:2101.11315, URL: https://arxiv.org/abs/2101.11315.

[22] Z. Aouini, A. Pekar, NFStream: A flexible network data analysis framework, Comput. Netw. 204 (2022) 108719, http://dx.doi.org/10.1016/j.comnet.2021.108719, URL: https://www.sciencedirect.com/science/article/pii/S1389128621005739.

[23] T. AbuHmed, A. Mohaisen, D. Nyang, A survey on deep packet inspection for intrusion detection systems, 2008, arXiv preprint arXiv:0803.0037.

[24] H. Dreger, A. Feldmann, V. Paxson, R. Sommer, Operational experiences with high-volume network intrusion detection, in: Proceedings of the 11th ACM Conference on Computer and Communications Security, 2004, pp. 2–11.

[25] R. Koch, Towards next-generation intrusion detection, in: 2011 3rd International Conference on Cyber Conflict, 2011, pp. 1–18.

[26] B. Claise, B. Trammell, P. Aitken, Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information, Technical Report, 2013.

[27] B. Claise, Cisco Systems Netflow Services Export Version 9, Technical Report, 2004.

[28] S. Leinen, Evaluation of Candidate Protocols for IP Flow Information Export (IPFIX), Technical Report, 2004.

[29] B. Caswell, J. Beale, Snort 2.1 Intrusion Detection, Elsevier, 2004.

[30] W. Park, S. Ahn, Performance comparison and detection analysis in snort and suricata environment, Wirel. Pers. Commun. 94 (2) (2017) 241–252.

[31] Palo Alto Networks, 2005, https://www.paloaltonetworks.com/. (Accessed 17 November 2023).

[32] Fortinet, 2000, https://www.fortinet.com/. (Accessed 17 November 2023).

[33] A. Pekar, A. Duque-Torres, W. Seah, M. Caicedo, Knowledge discovery: Can it shed new light on threshold definition for heavy-hitter detection? J. Netw. Syst. Manage. 29 (2021) 30, http://dx.doi.org/10.1007/s10922-021-09593-w.

[34] L. Liu, P. Wang, J. Ruan, J. Lin, Conflow: Contrast network flow improving class-imbalanced learning in network intrusion detection, 2022, http://dx.doi.org/10.21203/rs.3.rs-1572776/v1.

[35] G. Vormayr, J. Fabini, T. Zseby, Why are my flows different? A tutorial on flow exporters, IEEE Commun. Surv. Tutor. 22 (3) (2020) 2064–2103, http://dx.doi.org/10.1109/COMST.2020.2989695.

[36] M. Sarhan, S. Layeghy, N. Moustafa, M. Portmann, NetFlow datasets for machine learning-based network intrusion detection systems, in: Z. Deze, H. Huang, R. Hou, S. Rho, N. Chilamkurti (Eds.), Big Data Technologies and Applications, Springer International Publishing, Cham, 2021, pp. 117–135.

[37] O. Bader, A. Lichy, A. Dvir, R. Dubin, C. Hajaj, Open-source framework for encrypted internet and malicious traffic classification, 2022, arXiv preprint arXiv:2206.10144.

[38] D. Giagkos, O. Kompougias, A. Litke, N. Papadakis, Zeekflow: Deep learning-based network intrusion detection a multimodal approach, in: S. Katsikas, H. Abie, S. Ranise, L. Verderame, E. Cambiaso, R. Ugarelli, I. Pra¸ca, W. Li, W. Meng, S. Furnell, B. Katt, S. Pirbhulal, A. Shukla, M. Ianni, M. Dalla Preda, K.-K.R. Choo, M. Pupo Correia, A. Abhishta, G. Sileno, M. Alishahi, H. Kalutarage, N. Yanai (Eds.), Computer Security. ESORICS 2023 International Workshops, Springer Nature Switzerland, Cham, 2024, pp. 409–425.

[39] M. Rodríguez, A. Alesanco, L. Mehavilla, J. García, Evaluation of machine learning techniques for traffic flow-based intrusion detection, Sensors 22 (23) (2022) http://dx.doi.org/10.3390/s22239326, URL: https://www.mdpi.com/1424-8220/22/23/9326.

[40] N. Thapa, Z. Liu, A. Shaver, A. Esterline, B. Gokaraju, K. Roy, Secure cyber defense: An analysis of network intrusion-based dataset CCD-IDSv1 with machine learning and deep learning models, Electronics 10 (15) (2021) http://dx.doi.org/10.3390/electronics10151747, URL: https://www.mdpi.com/2079-9292/10/15/1747.

[41] A. Pektaş, T. Acarman, A deep learning method to detect network intrusion through flow-based features, Int. J. Netw. Manage. 29 (3) (2019) e2050, http://dx.doi.org/10.1002/nem.2050, arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/nem.2050, URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/nem.2050, e2050 nem.2050.

[42] D. Arp, E. Quiring, F. Pendlebury, A. Warnecke, F. Pierazzi, C. Wressnegger, L. Cavallaro, K. Rieck, Dos and don'ts of machine learning in computer security, in: 31st USENIX Security Symposium, USENIX Security 22, USENIX Association, Boston, MA, 2022, pp. 3971–3988, URL: https://www.usenix.org/conference/usenixsecurity22/presentation/arp.

[43] P. Geurts, D. Ernst, L. Wehenkel, Extremely randomized trees, Mach. Learn. 63 (2006) 3–42.

[44] L. Breiman, Random forests, Mach. Learn. 45 (2001) 5–32.

[45] F. Murtagh, Multilayer perceptrons for classification and regression, Neurocomputing 2 (5) (1991) 183–197, http://dx.doi.org/10.1016/0925-2312(91)90023-5, URL: https://www.sciencedirect.com/science/article/pii/0925231291900235.

[46] V. Nair, G.E. Hinton, Rectified linear units improve restricted Boltzmann machines, in: Proceedings of the 27th International Conference on International Conference on Machine Learning, ICML '10, Omni Press, Madison, WI, USA, 2010, pp. 807–814.

[47] X. Glorot, A. Bordes, Y. Bengio, Deep sparse rectifier neural networks, in: G. Gordon, D. Dunson, M. Dudík (Eds.), Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, in: Proceedings of Machine Learning Research, vol. 15, PMLR, Fort Lauderdale, FL, USA, 2011, pp. 315–323, URL: https://proceedings.mlr.press/v15/glorot11a.html.

[48] W. Wang, M. Zhu, X. Zeng, X. Ye, Y. Sheng, Malware traffic classification using convolutional neural network for representation learning, in: 2017 International Conference on Information Networking, ICOIN, 2017, pp. 712–717, http://dx.doi.org/10.1109/ICOIN.2017.7899588.

[49] I. Sharafaldin, A.H. Lashkari, A.A. Ghorbani, Toward generating a new intrusion detection dataset and intrusion traffic characterization, in: International Conference on Information Systems Security and Privacy, 2018, URL: https://api.semanticscholar.org/CorpusID:4707749.

[50] N. Moustafa, J. Slay, UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set), in: 2015 Military Communications and Information Systems Conference, MilCIS, 2015, pp. 1–6, http://dx.doi.org/10.1109/MilCIS.2015.7348942.

[51] H. Lawrence, U. Ezeobi, O. Tauil, J. Nosal, O. Redwood, Y. Zhuang, G. Bloom, CUPID: A labeled dataset with pentesting for evaluation of network intrusion detection, J. Syst. Archit. 129 (2022) 102621.

[52] 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, E. Duchesnay, Scikit-learn: Machine learning in python, J. Mach. Learn. Res. 12 (2011) 2825–2830, http://dx.doi.org/10.5555/1953048.2078195, URL: https://dl.acm.org/doi/10.5555/1953048.2078195.

[53] B. McMahan, E. Moore, D. Ramage, S. Hampson, B.A.y. Arcas, Communication-efficient learning of deep networks from decentralized data, in: A. Singh, J. Zhu (Eds.), Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, in: Proceedings of Machine Learning Research, vol. 54, PMLR, 2017, pp. 1273–1282, URL: https://proceedings.mlr.press/v54/mcmahan17a.html.

[54] S.M. Lundberg, S.-I. Lee, A unified approach to interpreting model predictions, in: Proceedings of the 31st International Conference on Neural Information Processing Systems, NIPS '17, Curran Associates Inc., Red Hook, NY, USA, 2017, pp. 4768–4777.

[55] G. Apruzzese, P. Laskov, E. Montes de Oca, W. Mallouli, L. Brdalo Rapa, A.V. Grammatopoulos, F. Di Franco, The role of machine learning in cybersecurity, Digit. Threats 4 (1) (2023) http://dx.doi.org/10.1145/3545574.

[56] M. Ring, S. Wunderlich, D. Scheuring, D. Landes, A. Hotho, A survey of network-based intrusion detection data sets, Comput. Secur. 86 (2019) 147–167, http://dx.doi.org/10.1016/j.cose.2019.06.005, URL: https://www.sciencedirect.com/science/article/pii/S016740481930118X.

[57] M. Sarhan, S. Layeghy, M. Portmann, Evaluating standard feature sets towards increased generalisability and explainability of ML-based network intrusion detection, Big Data Res. 30 (2022) 100359, http://dx.doi.org/10.1016/j.bdr.2022.100359, URL: https://www.sciencedirect.com/science/article/pii/S2214579622000533.

[58] G. Engelen, V. Rimmer, W. Joosen, Troubleshooting an intrusion detection dataset: the CICIDS2017 case study, in: 2021 IEEE Security and Privacy Workshops, SPW, 2021, pp. 7–12, http://dx.doi.org/10.1109/SPW53761.2021.00009.

[59] L. Liu, G. Engelen, T. Lynar, D. Essam, W. Joosen, Error prevalence in NIDS datasets: A case study on CIC-IDS-2017 and CSE-CIC-IDS-2018, in: 2022 IEEE Conference on Communications and Network Security, CNS, 2022, pp. 254–262, http://dx.doi.org/10.1109/CNS56114.2022.9947235.

Meryem Janati Idrissi received the M.S. degree in Big Data Analytics & Smart Systems from Sidi Mohamed Ben Abdellah University, Fes, Morocco, in 2018. She is currently pursuing the Ph.D. degree in Data science, Networking and Algorithmic thinking from Mohammed VI Polytechnic University, Benguerir, Morocco. His current research interests include intrusion detection, federated learning, privacy-preserving.

Hamza Alami is currently a postdoctoral research associate at Mohammed VI Polytechnic University (UM6P), SCCS, Benguerir, Morocco. In 2021, he received his PHD degree in computer science from Ibn Tofail University, Morocco. His research interests include natural language processing, distributed learning, network intrusion detection, and information extraction.

Abdelkader El Mahdaouy graduated from the Faculty of Science Dhar El Mahraz of Sidi Mohamed Ben Abdellah University in Computer Science, in 2012. Then, he received his Ph.D. in Computer Science in a joint program between Grenoble Alps University and Sidi Mohamed Ben Abdellah University in December 2017. In September 2020, He joined Mohammed VI Polytechnic University (UM6P) as a PostDoc. Currently, he is a Scientist at UM6P. My research interests include Information Retrieval and Arabic Natural Language Processing.

Abdelhak Bouayad received the M.S. degree in Big Data Analytics & Smart Systems from Sidi Mohamed Ben Abdellah University, Fes, Morocco, in 2018. He is currently pursuing the Ph.D. degree in Data science, Networking and Algorithmic thinking from Mohammed VI Polytechnic University (UM6P), CC, Benguerir, Morocco. His research interests include privacy-preserving machine learning, network intrusion detection systems, and federated learning.

Zakaria Yartaoui is a cybersecurity engineer at the National Computer Emergency Response Team of Morocco (maCERT) since 2019. He holds a degree in aeronautical engineering from the Royal Air Academy of Morocco and a master's degree in cybersecurity from the National Institute of Posts and Telecommunications (INPT-Morocco). He is pursuing a Ph.D. at UM6P, with a research focus on Artificial Intelligence and Machine Learning-based computer network anomaly detection systems.

Ismail Berrada is currently an associate professor in computer science, at Mohammed VI Polytechnic University (UM6P), SCCS, Benguerir, Morocco. In 2005, he received his PHD degree in computer science from University of Bordeaux 1, France. His research mainly focuses on Artificial Intelligence's (AI) applications in multiple domains such as cognitive radio networks, radio resource management, vehicular ad hoc network, road safety, software testing and verification. Dr. Berrada had also worked for 5 years as an assistant professor in the University of La Rochelle, France, and for 10 years in USMBA, Fez, Morocco.