Task Assignment and Capacity Allocation for ML-based Intrusion Detection as a Service in a Multi-tier Architecture

Yuan-Cheng Lai, Didik Sudyana, Ying-Dar Lin, *Fellow, IEEE*, Miel Verkerken, Laurens D'hooge, Tim Wauters, *Member, IEEE*, Bruno Volckaert, *Senior Member, IEEE*, and Filip De Turck, *Fellow, IEEE*

***Abstract*—Intrusion Detection Systems (IDS) play an important role in detecting network intrusions. Because intrusions have many variants and zero-day attacks, traditional signature- and anomaly-based IDS often fail to detect them. On the other hand, solutions based on Machine Learning (ML), have better capabilities for detecting variants. In this work[1], we adopt an ML-based IDS which uses three in-sequence tasks, pre-processing, binary detection, and multi-class detection, with a multi-tier architecture with one-, two-, and three-tier architectural configurations. We then mapped three in-sequence tasks into these architectures, resulting in ten task assignments. We evaluated these with queueing theory to determine which tasks assignments were more appropriate for particular service providers. With simulated annealing, we obtained the computation capacity by allocating the total cost appropriate to each tier, based on the fixed parameter set with the objective of minimizing overall delay. These investigations showed that using only the edge and allocating all tasks to it gave the best performance. Furthermore, a two-tier architecture with edge and cloud components was also sufficient for IDS as a Service with the delay that was three times better than for other task assignments. Our results also indicate that more than 85% of the total capacity was allocated and spread across nodes in the lowest tier for pre-processing to reduce delays.**

***Index Terms*—ML-based IDS, multi-tier architecture, multi-stage machine learning, IDS as a Service.**

I. Introduction

Current intrusion detection systems (IDS) are commonly used to protect network devices by detecting specific attacks and raising alerts for network operators [2]. IDS can generally be classified as signature-based or anomaly-based. Signature-based IDSs often fail to detect unknown attacks while anomaly-based IDS result in high false positive (FP) rates. To address such problems, researchers have begun to use machine learning in IDS that has the ability to detect more attack variants and novel attacks [3].

Yuan-Cheng Lai is with the Department of Information Management, National Taiwan University of Science and Technology, Taipei 106, Taiwan (e-mail: laiyc@cs.ntust.edu.tw).

D. Sudyana and Ying-Dar Lin are with the Department of Computer Science, National Yang Ming Chiao Tung University, Hsinchu 300, Taiwan (e-mail: dsudyana@cs.nctu.edu.tw; ydlin@cs.nctu.edu.tw).

M. Verkerken, L. D'hooge, T. Wauters, B. Volckaert, and F. De Turck are with IDLab, Department of Information Technology, Ghent University, 9000 Ghent, Belgium (e-mail: miel.verkerken@ugent.be; laurens.dhooge@ugent.be; tim.wauters@ugent.be; bruno.volckaert@ugent.be; filip.deturck@ugent.be)

[1]This work is an extension of [1], with improved survey, considering computation cost, and more results.

The use of IDS in a network is essential to ensure security. Before now enterprises were responsible for managing their security system by purchasing an IDS hardware box from a vendor. However, retaining an IT team to manage and monitor these systems 24/7 is time-consuming and costly. As a result, many enterprises are outsourcing IDS management to service providers in order to reduce staff, obtain better services, and minimize operational costs. This trend shifts the intrusion detection paradigm from a hardware unit to a service, allowing for the possibility of intrusion detection as a service (IDaS).

From the perspective of service providers, who would operate such a service, the architecture and algorithm of IDS are the factors to consider. Previously, IDS was typically based on centralized cloud computing solutions. However, such an architecture is hard to scale due to significant latency and cost associated with transferring traffic to a centralized IDS. The use of a distributed architecture can be a solution to such problems in a centralized system, which is capable of distributing computing in parallel and closer to users. It can employ a number of nodes in a multi-tier system and then distribute the tasks to various components. Recently cloud, edge, and fog computing have become the prospective paradigms that can be employed as components of a multi-tier architecture system.

Cloud computing is one of the most widely-used computer concepts, and has also been leveraged by the European Telecommunication Standards Institute (ETSI) to develop edge computing. This has enabled mobile network operators to virtualize cloud computing capabilities and is known as Multi-access Edge Computing (MEC). Fog computing is similar to edge computing, but it is much closer to a user and is owned by subscribers. While cloud computing provides rapid availability, good reliability, and nearly limitless capacity, fog and edge computing contribute mobility and performance enhancement. By combining these three systems and establishing a multi-tier architecture based on a cloud-edge-fog hierarchy as defined in [4], enables the effective handling of a variety of services with different characteristics while preserving availability and reliability [5].

Furthermore, machine learning-based IDS can carry out several tasks, from pre-processing raw data to detecting attacks. There are two forms of attack detection in machine learning-based IDS, binary and multi-class. Many earlier studies only addressed a single-stage binary or multi-class model. A single-stage binary can suffer from overfitting and is a strongly biased model [6], and such existing multi-class detection is also not accurate enough [7]. Combining these attack detection approaches can help to reduce model instabilities [6].

Starting with pre-processing, we used three tasks to carry out ML-based IDS with binary and multi-class detection. Pre-processing involves transforming raw data into a tabular format and removing irrelevant data to improve classifier algorithm efficiency. Binary detection classifies traffic as benign or malicious and focuses on preventing overfitting and reducing bias towards normal traffic. Malicious traffic only will then be transferred to classify the attack class in a multi-class detection system. This combination has been shown to improve accuracy [6], reduce false-positive rates [8], and save bandwidth by not transferring all traffic.

In this study, we broke these three tasks down and mapped them to the architecture, resulting in several different configurations. The results of this mapping process were then referred to as task assignments. We then mapped those tasks onto proper architectures and analyzed which task assignments provided the best performance in terms of end-to-end delay. These task assignments are important for service providers. By analyzing such task assignments, service providers may determine which task assignments for intrusion detection as a service they could offer to their customers.

When we mapped the tasks to the architectures, we also had to properly allocate capacity by optimizing capacity allocation for each tier, based on overall computation cost. Certain service providers may be concerned with computation cost, since the cost for each tier may be different in order to obtain the desired capacity. We then modeled those task assignments with the objective of determining the minimum delay between task assignments. We used the simulated annealing algorithm to optimize and determine the cost of allocating the capacity for each architecture, based on arrival traffic rates and overall cost. Then, to analyze and classify the traffic, we employed a flow-based approach. This is an aggregation of transmitted network packets which share some properties. Flows are mostly employed in machine learning-based IDS and improve real-time traffic classification performance [9]. We also measured the computational load of machine learning-based IDS and used it in the simulation exercise. Such real-world data aids in bringing a model's behavior closer to a real-life situation.

To broaden our knowledge, we also investigated (1) one- vs. two- vs. three-tier architecture, to determine which architecture provided the best performance based on allocated capacity, (2) joint vs. separated task assignment, to evaluate the performance of task assignment when three tasks are combined in one place, against when three tasks are separated into different tiers, and (3) fog vs. edge vs. cloud capacity allocation, to identify how the algorithm allocates capacity between the tiers. We also investigated the effects of different parameters in task assignments performance: the effect of malicious traffic, which impacts the performance of task assignments when faced with a rising volume of such traffic that needs to be processed, the effect of flow workload, which influences the performance of task assignments when different machine learning models for binary and multi-class detection were employed, and the effect of computation cost, how a particular overall computation cost may affect the capacity and performance of task assignments. For example, which task assignments are better in a low-cost or high-cost scenario?

Some studies have discussed distributed ML-based IDS [7]-[8], [10]-[20] with no consideration of potential architecture problems. The primary objective of this work is to improve the accuracy of machine learning models while also evaluating the response time of such models when used in a particular architecture. With reference to related work, the contributions of this paper address three particular issues: (1) it is, to the best of our knowledge, the first study which considers a mapping process for tasks to architecture in an intrusion detection as a service and formally models its architecture, (2) the use of a simulated annealing method to determine how much capacity a particular architecture needs for task assignments, as well as the effect of computing cost and several parameters on task assignments performance, and (3) the analysis of those task assignments to provide appropriate recommendations to service providers.

Furthermore, compared with the previous conference version [1], this extension mainly has four improvements: (1) The related works are comprehensively surveyed and structured according to the IDS type. Also, their limitations and drawbacks are emphasized with respect to the paper. (2) The cost of computation on doing optimization is considered because the cost associated with each tier may be different. For example, the computation cost of a fog may be higher than that in the cloud. (3) In the solution, we simplify the calculation of the total delay by separating the transmission delay with the propagation delay. The algorithm is also changed to allocate the appropriate cost and the corresponding capacity to each tier. (4) We explore more issues, including the effect of malicious traffic, the effect of flow length, and the effect of communication cost, to obtain some insightful observations on these parameters.

The remainder of this paper is organized as follows. Section II reviews previous works in the distributed of IDS. Section III defines the system architectures and problem formulation. The solution algorithm, is described in Section IV; Section V covers the simulation and results, and Section VI concludes the work.

II. Related Work

Table I summarizes some previous works that have dealt with distributed IDS from signature-based to machine learning-based systems. The comparisons are based on the IDS type, architecture used, the IDS tasks considered, the purpose and method used.

There are three IDS types: signature-based, single-stage ML-based, and multi-stage ML-based approaches.

TABLE I: Summary of The Research on Distributed IDS

Paper	IDS Type	Architecture	Jobs in IDS	Purpose	Method	A/S/I
[10]	Signature-based	Flat network	Detection/Correlation	Accuracy	Cloud theory	I (HTTP-flood attacks)
[11]		Mobile Agent-Cloud	Detection/Correlation	Network load reduction	Suricata	I (Pytbull)
[12]	Single-stage ML-based	Fog-Cloud	Full	Accuracy	Federated LSTM	I (ISCX intrusion dataset)
[13]		Fog	Full	Accuracy	RNN, LSTM, Bi-LSTM, GRU, CNN, CNN-LSTM	I (five datasets)
[14]		Fog	Full	Accuracy	vAE with MLP	I (IoT-23 dataset)
[15]		Fog	Full	Accuracy	Online Adaptive ML	I (Self-collected)
[16]		Dew-Cloud	Pre-processing/Binary	Accuracy	Deep belief network	I (UNSW-NB15 dataset)
[17]		IoT-Cloud	Full	Accuracy	FL	I (NSL-KDD dataset)
[18]		Two-level servers	Full	Accuracy	FL aided LSTM	I (AT&T SEA dataset)
[19]		Edge-Cloud	Full	Accuracy	FL with RL	I (UNSW-NB15 dataset)
[8]	Multi-stage ML-based	IoT-Edge	Binary Classification/ Multi-classification	Accuracy	Multi-modal DAE, Soft output classifier	I (Bot-IoT Dataset)
[7]		Fog	2 Binary Classifications	Accuracy	DNN-KNN	I (NSL-KDD & CIC-IDS2017 dataset)
[20]		Fog	2 Binary Classifications	Accuracy	RNN	I (NSL-KDD dataset)
Ours		Multi-tier	Pre-processing/ Binary Classification/ Multi-classification	Minimum delay Optimal capacity	Architecture selection with queueing theory	A

A/S/I: Analysis/Simulation/Implementation, FL: Federated Learning

Signature-based approches rely on a database containing the patterns or signatures of known attacks. Single-stage ML-based approaches rely on a single machine learning model for either detection or classification. On the other hand, multi-stage ML-based approaches adopt a cascade of detection and classification methods using multiple machine learning models on the same input data to get a higher classification accuracy.

A distributed IDS using a signature-based system is considered in [10] and [11]. Their primary objective was to create a signature-based IDS algorithm and applied it to a host-based system. However, the utilization of host-based IDS systems is not suitable to large-scale environments and a signature-based system frequently fails to detect unknown attacks. In spite of the fact that the goal in [11] was to reduce network traffic, it only used mobile agents and cloud nodes, so it did not provide a wider perspective on which architecture is best for the intrusion detection as a service.

The single-stage ML-based approach with a distributed architecture is adopted in [12]-[19]. [12] investigated an Long Short-Term Memory (LSTM) model in a distributed environment. It distributed the training data over fog nodes and coordinated these nodes using parameter exchanges through a centralized node. All of these efforts were done to improve the detection accuracy and scalability. [13] assessed six deep learning models and found that LSTM had the best performance among these models. Then, it deployed the proposed model into a distributed architecture, which owns two fog nodes and a cloud node. The ML model was placed in fog, and when the fog detects an attack, the detection result will be sent to the cloud. [14] combined variational AutoEncoder with Multi-layer Perceptron and used it in a fog environment to increase the efficiency and accuracy of intrusion detection. It implemented the proposed model into the fog nodes and evaluated the overhead of the model. Similarly, [15] used an online adaptive machine learning model to improve the quality of ML-based IDS and reduce the workload. The proposed approach was then successfully applied to fog nodes, resulting in a faster response time. [17]-[19] evaluated distributed IDS for federated learning and achieved comparable accuracy with the centralized solution that has global knowledge of the whole system. Federated learning not only gets the benefit from distributed architecture, but also meets user privacy requirements.

Several works utilized a two-stage machine learning detection method and used fog as the computation nodes for a distributed IDS [7], [8], [20]. A two-stage binary classification is used in [7] to reduce model instability and obtain good results for accuracy and precision. Similarly, [20] also used a two-stage binary classification with a Recurrent Neural Network (RNN) and achieved a high level of sensitivity to certain attacks. Both studies then applied their models to fog nodes. However, they only tested the model's overhead in a fog environment. Deploying IDS on fog nodes in one-tier architecture reduces response time by bringing the processing node closer to the user. However, fog has acceptable latency but typically insufficient processing power to deal with large computing processes. Furthermore, [8] proposed a two-stage IDS, which includes binary classification and multi-classification. However, the authors only mentioned that their solution could be deployed in the IoT-Edge architecture, but did not conduct any further architecture-related experiments.

The main differences between this paper and the previous research are as follows: (1) The previous research mainly focused on improving the detection accuracy, but our paper focuses on minimizing the detection delay. (2) The previous research tried to improve the quality of ML models/signature patterns, but our paper wants to investigate the issues of task assignment, capacity allocation, and architecture selection, of ML-based IDS in a multi-tier architecture. (3) The previous research only used at most two stages, but our paper uses three stages: pre-processing, binary detection, and multi-class classification. Using three stages can speed up the detection because most traffic judged as the normal traffic in the binary detection stage do not need to be further handled.

TABLE II: Model Notations

Notation	Meaning
N	The number of edge nodes
K	The number of fog nodes per edge node
$i = 1, 2, 3$	IDS tasks in attack detection process (1: pre-processing, 2: Binary, 3: Multi-class)
$1/\mu_i^W$	Flow workload of i-th task
$1/\mu_i^L$	Flow length of i-th task
C^C	Capacity of cloud
C^E	Capacity of edge
C^F	Capacity of fog
C^{UF}	Link bandwidth from UE to fog
C^{FE}	Link bandwidth from fog to edge
C^{EC}	Link bandwidth from edge to cloud
S	Total computation cost
S^C	Cost per unit of capacity at cloud
S^E	Cost per unit of capacity at edge
S^F	Cost per unit of capacity at fog
U^C	Allocated cost to cloud
U^E	Allocated cost to edge
U^F	Allocated cost to fog
λ	Arrival traffic rate from UE
Z^{UF}	Propagation delay between UE and fog
Z^{FE}	Propagation delay between fog and edge
Z^{EC}	Propagation delay between edge and cloud
D	Total delay
p^A	Probability of malicious flows
t	Temperature for Simulated Annealing Algorithm
t_{ini}	Initial Temperature
t_{trm}	Termination Temperature

However, the three-stage task assignment in a multi-tier architecture becomes more complex. (4) In order to minimize the detection delay, we deduce the load and capacity in each network node and link to calculate the delay using queueing theory. However, the previous research did not consider this.

III. System Model and Problem Formulation

This section describes task assignments, multi-tier architectures, problem descriptions, and a delay model, with the variables and notations used given in Table II.

A. System architectures

The system architecture is shown in Figure 1 where the multi-tier architecture is composed of a cloud, N number of edge nodes E, and K number of fog nodes F beneath the edge. We then map the IDS tasks as i into architectures. $1/\mu$ is workload of the traffic. There are two types of workloads: computation and communication. The former is the workload of computation tasks (pre-processing, binary, or multi-class) for processing traffic in the unit of the number of instructions and represented as $1/\mu_i^W$. The latter is the communication length or the flow traffic size for each task when transferring the data across the network in the unit of Kilobit, and it is denoted as $1/\mu_i^L$. The reason of using the denominator for μ is that when the capacity is C, we can get a service rate $C\mu$, which is a commonly used notation.

The lowest tier for pre-processing could be the fog, edge, or cloud. When traffic flows from user equipment (UE) arrive at the lowest-tier node with a length of $1/\mu_1^L$, it is immediately processed in the pre-processing stage with a workload $1/\mu_1^W$. This stage is used to extract features and clean the data before performing machine learning detection, and reduces the flow length into $1/\mu_2^L$. The data is then sent through binary detection to detect benign and malicious traffic with a workload $1/\mu_2^W$. Only malicious traffic with a probability of p^A and flow length $1/\mu_3^L$ will then be passed to the multi-class detection system with a workload $1/\mu_3^W$ for the attack types to be classified.

Each of the task assignments utilized the same total cost S. Then, depending on the task assignment chosen, S can be distributed to each tier based on S^C, S^E, S^F to obtain C^C, C^E, and C^F with bandwidth capacities of C^{UF}, C^{FE}, and C^{EC}. The traffic then passes through the fog at a rate of λ.

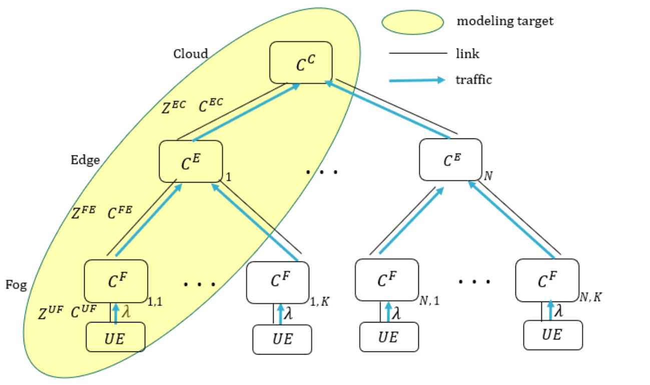


Fig. 1: Parameters employed in the architecture model

B. Task Assignment

In this work, we used a multi-tier architecture that can be utilized in seven different configurations, and can utilize one-, two-, or three-tier architecture. In one-tier, we can use only fog, edge, cloud. Two-tier can have several configurations such as fog-edge, fog-cloud, or edge-cloud, and three-tier architecture is a combination of fog-edge-cloud.

The three tasks can be broken down and mapped to each of the seven architectures, resulting in 10 possible task assignments, as illustrated in Table III. We used IDs and abbreviations to make it easier to recognize the task assignments. The abbreviation (xxx/xxx/xxx), which has three parts separated by slashes, illustrates where tasks are assigned. The left, middle, and right parts represent fog, edge and cloud, respectively. Then symbols p, b, and m, which indicate pre-processing, binary detection, and multi-class classification, respectively, are filled in these parts to exhibit task assignments. If no task is placed in a part, a symbol '-' will be filled. For example, task assignment 10 with abbreviation p/b/m means pre-processing, binary, and multi-class, are placed in the fog, edge, and cloud, respectively. Another example is the task assignment 7 with abbreviation p/bm/-, which indicates that we only adopt a two-tier architecture and place pre-processing in the fog, followed by binary and multi-class classification in the edge. The illustration of these task assignments are shown in Figure 2.

TABLE III: Possible assignments of preprocessing, binary and multi-class detection tasks over tiers

# of tiers	Architecture	Fog	Edge	Cloud	ID	Abbr.
1	Fog	p, b, m			1	pbm/-/-
	Edge		p, b, m		2	-/pbm/-
	Cloud			p, b, m	3	-/-/pbm
2	Fog-Edge	p, b	m		4	pb/m/-
	Fog-Cloud	p, b		m	5	pb/-/m
	Edge-Cloud		p, b	m	6	-/pb/m
	Fog-Edge	p	b, m		7	p/bm/-
	Fog-Cloud	p		b, m	8	p/-/bm
	Edge-Cloud		p	b, m	9	-/p/bm
3	Fog-Edge-Cloud	p	b	m	10	p/b/m

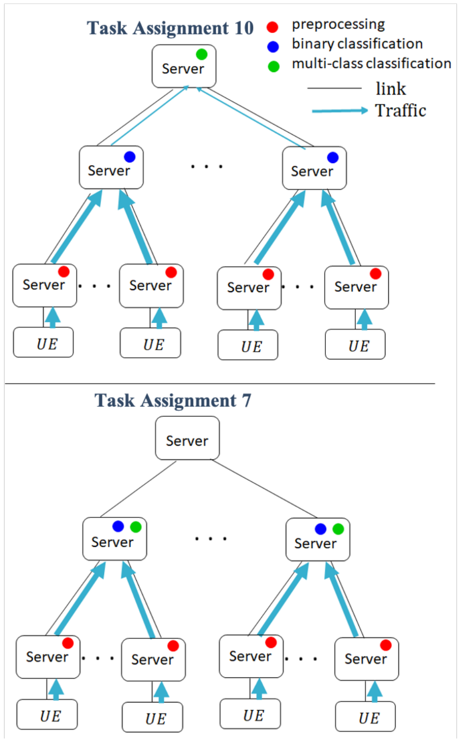


Fig. 2: Task Assignment Illustration

C. Problem statement

A cloud, edge nodes, fog nodes, a flow arrival rate, flow lengths and flow workloads, the probability of malicious flows, a link bandwidth, propagation delay between links, and cost per unit capacity for cloud, edge, and fog are the inputs for the objective of minimizing total delay and determining the best of the ten possible task assignments with the constraints of overall computation costs. To achieve this, capacity has to be appropriately allocated to each tier based on computation cost. The problem statement for the main problem is then formally defined as:

Input: A topology composed of N edge nodes, K fog nodes per edge node, a flow arrival rate in each UE of (λ), flow length of $(1/\mu_i^L)$, flow workload of $(1/\mu_i^W)$, the probability of malicious flows of (p^A), a link bandwidth $(C^{UF}$, C^{FE}, and $C^{EC})$, propagation between tiers $(Z^{UF}$, Z^{FE}, and $Z^{EC})$, and cost per unit capacity $(S^C$, S^E, $S^F)$.

Output: The allocated capacity to each tier $[C^F$, C^E, $C^C]$ and its corresponding task assignment based on the given cost.

Objective: Minimize the total delay D.

Constraint: The overall cost of each tier must equal the overall cost $C^F S^F + C^E S^E + C^C S^C = S$.

D. Delay model

In this model, the total delay in each component is composed transmission delay, computation delay and propagation delay. To calculate computation and transmission delays, we considered using a queuing model in each resource. In order to compute the delay while using the r-th source, we had to know the arrival rate, service capacity, and task size, since different task assignments assign these tasks (p, b, m) into different nodes.

For clearer a picture, we constructed three tables which are respectively represent the task size (table IV), arrival rate (table V), and service capacity (table VI) of different task assignments in the r-th source. $r = 1,2,3$ denotes the communication resource - the link from UE to fog, fog to edge, and edge to cloud, respectively, while $r = 4,5,6$ represents the computing resource of fog, edge, and cloud, respectively. Task assignments then utilized different combinations of those parameters to calculate delays.

For example, task assignment 1 which utilizes only fog to process three tasks, traffic is transmitted from UE to a fog link with flow length $1/\mu_1^L$, arrival rate λ in $H[1,1]$, and C^{UF} capacity in $Y[1]$. Traffic is then handled at fog node to process those tasks with $V[1,4]$ flow workload, $H[1,4]$ arrival rate, and $Y[4]$ fog node capacity. Finally, we can determine the delay associated with task assignment 1 by adding the communication delay from UE to fog and the computation delay at that fog node.

The total delay for each task assignment j may then be determined by adding up all delay components, which can be expressed as:

$$D_j = \sum_{r=1}^{6} b(j,r) \tag{1}$$

where

$$b(j,r) = \begin{cases} \dfrac{1}{\frac{Y[r]}{V[j,r]} - H[j,r]}, & \text{if } V[j,r] \neq \phi, \\ 0, & \text{if } V[j,r] = \phi. \end{cases} \tag{2}$$

The delay for communication links and computation nodes is calculated using equation 2 by looking up the values in Tables IV, V, and VI. Furthermore, when it comes to communication delay, it is not only the transmission delay, but also the propagation delay, that must be taken into account. We thus need to add propagation delay, which can be expressed as $Z^{UF} \times f(V[j,1]) + Z^{FE} \times f(V[j,2]) + Z^{EC} \times f(V[j,3])$, where $f(x) = 1$, if $x \neq \phi$, or $f(x) = 0$, if $x = \phi$.

IV. Solution Approach: Simulated Annealing for Optimizing Capacity based on Cost Allocation

The problem set out above is solved by implementing the SA for optimizing capacity based on a cost allocation

TABLE IV: Task Size in r-th resource for the j-th task assignment, $V[j,r]$

$V[j,r]$	Transmission			Computation (workload)		
	$r=1$	$r=2$	$r=3$	$r=4$	$r=5$	$r=6$
$V[1,r]$	$1/\mu_1^L$	ϕ	ϕ	$1/\mu_1^w + 1/\mu_2^w + (1/\mu_3^w.p^A)$	ϕ	ϕ
$V[2,r]$	$1/\mu_1^L$	$1/\mu_1^L$	ϕ	ϕ	$1/\mu_1^w + 1/\mu_2^w + (1/\mu_3^w.p^A)$	ϕ
$V[3,r]$	$1/\mu_1^L$	$1/\mu_1^L$	$1/\mu_1^L$	ϕ	ϕ	$1/\mu_1^w + 1/\mu_2^w + (1/\mu_3^w \cdot p^A)$
$V[4,r]$	$1/\mu_1^L$	$1/\mu_3^L$	ϕ	$1/\mu_1^w + 1/\mu_2^w$	$1/\mu_3^w \cdot p^A$	ϕ
$V[5,r]$	$1/\mu_1^L$	$1/\mu_3^L$	$1/\mu_3^L$	$1/\mu_1^w + 1/\mu_2^w$	ϕ	$1/\mu_3^w \cdot p^A$
$V[6,r]$	$1/\mu_1^L$	$1/\mu_1^L$	$1/\mu_3^L$	ϕ	$1/\mu_1^w + 1/\mu_2^w$	$1/\mu_3^w \cdot p^A$
$V[7,r]$	$1/\mu_1^L$	$1/\mu_2^L$	ϕ	$1/\mu_1^w$	$1/\mu_2^w + (1/\mu_3^w.p^A)$	ϕ
$V[8,r]$	$1/\mu_1^L$	$1/\mu_2^L$	$1/\mu_2^L$	$1/\mu_1^w$	ϕ	$1/\mu_2^w + (1/\mu_3^w \cdot p^A)$
$V[9,r]$	$1/\mu_1^L$	$1/\mu_1^L$	$1/\mu_2^L$	ϕ	$1/\mu_1^w$	$1/\mu_2^w + (1/\mu_3^w \cdot p^A)$
$V[10,r]$	$1/\mu_1^L$	$1/\mu_2^L$	$1/\mu_3^L$	$1/\mu_1^w$	$1/\mu_2^w$	$1/\mu_3^w \cdot p^A$

TABLE V: Arrival rate in the r-th resource, $H[j,r]$

$H[j,r]$	Transmission			Computation		
	$r=1$	$r=2$	$r=3$	$r=4$	$r=5$	$r=6$
$H[1,r]$	λ	ϕ	ϕ	λ	ϕ	ϕ
$H[2,r]$	λ	λK	ϕ	ϕ	λK	ϕ
$H[3,r]$	λ	λK	λKN	ϕ	ϕ	λKN
$H[4,r]$	λ	$\lambda K(p^A)$	ϕ	λ	λK	ϕ
$H[5,r]$	λ	$\lambda K(p^A)$	$\lambda KN(p^A)$	λ	ϕ	λKN
$H[6,r]$	λ	λK	$\lambda KN(p^A)$	ϕ	λK	λKN
$H[7,r]$	λ	λK	ϕ	λ	λK	ϕ
$H[8,r]$	λ	λK	λKN	λ	ϕ	λKN
$H[9,r]$	λ	λK	λKN	ϕ	λK	λKN
$H[10,r]$	λ	λK	$\lambda KN(p^A)$	λ	λK	λKN

TABLE VI: Capacity in the r-th resource, $Y[r]$

$r=1$	$r=2$	$r=3$	$r=4$	$r=5$	$r=6$
C^{UF}	C^{FE}	C^{EC}	C^F	C^E	C^C

algorithm. First, we have to allocate the cost to each tier based on the total cost, then convert the allocated cost to computation capacity depending on the cost per computation in each tier and calculate the average delay using queueing theory. The algorithm then determines the best cost allocation to obtain the capacity for all task assignments based on the overall cost S, and adjusts it until the system exhibits a minimum delay.

Algorithm 1 describes a simulated annealing technique which finds an approximate globally-optimum solution for the least delay using a probabilistic technique, based on random movements of a new cost allocation. This technique was selected because it avoids being stranded at a local minimum by accepting worse variants as new solutions with a probability. Using this algorithm, a control parameter t, (often known as the 'temperature') is used to control the randomness of the search process. With every iteration, the algorithm then decreases temperature t according to α. The algorithm stops the iteration when temperature t reaches the value of t_{trm}.

We then generate a new capacity for C^F, C^E, and C^C each iteration, based on the allocated cost, by applying randomization with specified constraints to minimum and maximum values. When generating a new solution for C^F, it first randomizes the cost for U^F, with a minimum number of U^F_{min} to prevent running out of capacity, which might result in a negative delay. U^F_{min} is calculated as $(H[j,4] \times V[j,4] \times S^F)$. The maximum number is U^F_{max} and can be calculated as $S - (U^E_{min} + U^C_{min})$, which provides sufficient space capacity for C^E and C^C. Once the allocated cost is determined, we divide U^F by S^F to get the capacity for C^F. Further, when generating capacity for C^E, we must assign the cost for U^E using randomization, with the minimum value of U^E_{min} calculated as $(H[j,5] \times V[j,5] \times S^E)$ and the maximum value of U^E_{max} calculated as $(U^E_{min} + U^C_{min})$. The capacity for C^E is then determined by dividing the allocated cost U^E by S^E. Thus, the residual number is the value of U^C that will be divided by S^C in order to obtain the capacity for C^C.

For illustrative purposes, we use the task assignment 4 to illustrate how to obtain the computation capacity with the shortest delay. The task assignment 4, pb/m/-, employs a two-tier architecture. Thus, the algorithm must assign capacity to fog and edge. Using algorithm 1, we begin the optimization with given overall computation cost S as an input. The capacity for fog, C^F, is then produced by randomly selecting cost for U^F between the minimal and maximum values, U^F_{min} and U^F_{max}. The U^F is then divided by S^F to get C^F. Therefore, the residual is the value of U^E that will be divided by S^E to obtain the capacity, C^E. Then, we will use the obtained C^F and C^E values to compute the total delay utilizing equation 1 and 2 with the assistance of tables IV, V, and VI. After calculating the overall delay, the result is compared to the current solution. If the new delay is less than the previous delay, the new capacity allocation will replace the current one. The algorithm will continuously iterate and generate new capacity allocation until the temperature reaches t_{trm}. Then we will obtain the optimal capacity allocation, which has the smallest delay, for the task assignment 4.

V. Simulation and Results

This section contains the numerical results of the simulation tests conducted to evaluate the performance of our task assignments.

Algorithm 1: SA for Optimizing Capacity based on Cost Allocation

```
initialize $C^F, C^E, C^C, t_{ini}, \alpha$;
$t = t_{ini}$;
calculate $D_j$;
while $t > t_{trm}$ do
    generate new solutions for $C^F$, $C^E$, $C^C$ by;
        $U^F = rand[U^F_{min}, U^F_{max}]$;
        $C^{F'} = \frac{U^F}{S^F}$;
        $U^E = rand[U^E_{min}, U^E_{max}]$;
        $C^{E'} = \frac{U^E}{S^E}$;
        $U^{C'} = S - U^F - U^E$;
        $C^{C'} = \frac{U^C}{S^C}$;
    calculate $D'_j$;
    $\Delta D_j = D'_j - D_j$ ;
    if $D'_j \leq D_j$ then
        accept new $C^F = C^{F'}$, $C^E = C^{E'}$, $C^C = C^{C'}$, $D_j = D'_j$;
    else
        accept new $C^F = C^{F'}$, $C^E = C^{E'}$, $C^C = C^{C'}$, $D_j = D'_j$,
          with probability $\frac{1}{exp(\Delta D_j / t)}$;
    end
    decrease the temperature: $t = \alpha \times t$;
end
```

A. Measurements on Binary and Multi-class algorithms

To run the simulation, certain parameters from a real system had to be collected. The workloads used in this study come from a real IDS system. We built a machine learning-based intrusion detection pipeline that consisted of the three tasks. Once the flow is sent, we kept track of how many instructions were needed. We used PERF on Linux to monitor instruction numbers, and the tests were run using an Intel Xeon E5-2609 2.4GHz CPU.

For binary detection, we had to identify malicious traffic quickly while maintaining performance. A decision tree is suitable for this criterion because it is fast and accurate [21]. For multi-class detection, we used deep learning as it is more capable of distinguishing an intrusion and also more effective for detecting unforeseen intrusions [22].

We constructed Decision Trees with default hyperparameters and utilized Deep Neural Networks from [22]. We trained the models by using CICIDS-2017 as dataset. In training DNN as the multi-class detection, we only used the malicious data.

Pre-processing involved extracting features from raw traffic using CICFlowMeter [23], cleaning the data, and parsing it correctly. The data were also rescaled using Min-Max Scaler with the help from Scikit library.

In addition to investigate the effect of flow workload, we tested several machine learning models, Random Forest, Gradient Boosting, and Multilayer Perceptron. All of these machine learning models were trained on CICIDS-2017 and used for binary and multi-class detection. Table VII lists the average of flow workload measurements for 10 tests. We also observed that after pre-processing, flow length decreased from 100% to 5.3%.

TABLE VII: Flow Workload

Stage		Flow Workload (Instructions)
Preprocessing		1,137,406
Binary	Decision Tree	114,549
	Random Forest	153,288
	Gradient Boosting	121,488
	MLP	304,367
	DNN	908,214
Multi-class	Decision Tree	114,849
	Random Forest	167,493
	Gradient Boosting	197,266
	MLP	312,614
	DNN	915,083

B. Parameters settings

Table VIII lists the parameters used in the simulations, comprising a cloud node, 400 edge nodes, and 8000 fog nodes. Each edge node covered 20 fog nodes and link bandwidth was considered as a 5G network. The distance between each tier was taken from [5].

Furthermore, to investigate the effect of computing cost, we set up the costs of computation capacity at the lower tiers to be more expensive than the higher tiers because the lowest tier has greater relative costs for real estate, cooling, maintenance employees, and other applications. It is possible to obtain significantly lower prices for server hardware and software licensing by making use of the economies of scale that result from using huge hyper-scale data centers rather than by distributing the data over several data centers [24]. We then set the costs S^C, S^E, S^F to [10, 15, 20] in money units.

The results of the experiments could be generalized to real scenarios. To make the experiment more realistic, we did a lot of efforts, including (1) Traffic parameters have been referred from a real system. For example, to obtain the parameter of flow workload, we did the measurements by developing a machine learning-based intrusion detection pipeline that consists of the three tasks. For the flow length, we took the number from the previous paper that practically measures the average flow size in a network [25]. (2) Based on literature, our simulation constructs a realistic network architecture. In this architecture, the fog is the closest thing to UEs, it has lower computation capacity than edge and cloud, and it has a higher computation cost.

C. Results

1) One- vs. two- vs. three-tier architecture: Figure 3 shows the comparative results of the ten task assignments with the total delay for processing a flow. These result shows that utilizing only the edge as in task assignment 2 had the shortest delay in processing the traffic, which is ideal in terms of computing capacity and distance between the UE and the computing node. When we moved the computing node closer to the UE by utilizing only the fog as in task assignment 1, the coverage area was too small and we then had to scatter more resources to cover the area. Scattering too many resources in queueing systems results in poor performance which increases computation time. On the other

TABLE VIII: Parameter settings

Notations	Value	Unit
N	400	Nodes
K	20	Nodes
λ	100	Flows/second
p^A	0.2	
$1/\mu_1^W$	1,137,406	Instructions per flow
$1/\mu_2^W$	114,549	Instructions per flow
$1/\mu_3^W$	915,083	Instructions per flow
$1/\mu_1^L$	124.99	Kilobit
$1/\mu_2^L$	6.62	Kilobit
$1/\mu_3^L$	6.62	Kilobit
Z^{UF}	3.335×10^{-6}	Seconds. Distance: 1Km
Z^{FE}	3.335×10^{-5}	Seconds. Distance: 10Km
Z^{EC}	3.335×10^{-3}	Seconds. Distance: 1000Km
C^{UF}	1	Gbps
C^{FE}	100	Gbps
C^{EC}	100	Gbps
S	50×10^{12}	Money units
S^C	10	Money units/instruction
S^E	15	Money units/instruction
S^F	20	Money units/instruction
t_{ini}	10000	
α	0.085	

hand, using just cloud node maximizes computing capacity, speeds up computing time, and simplifies management. However, because a large amount of raw traffic from all UEs must be delivered to the farthest node, it results in a transmission bottleneck.

To reduce the delay in the utilizing cloud, it should be combined with other tiers and form a two- or three-tier system. Utilizing cloud with edge, might reduce total delay by more than 14 times. Pre-processing data in a lower-tier would reduce a large transmission delay and, as a result, we save a lot of bandwidth and enhance processing capability.

Utilizing cloud with fog, as in task assignments 5 and 8, could also be a solution. Although these task assignments resulted in three times the total latency compared to cloud with edge, they provided greater privacy. Sending raw traffic to a higher tier implies aggregating traffic with other tenants.

Furthermore, utilizing fog with edge, as in task assignments 4 and 7, also could handle privacy issues with shorter delays. This combination also saves bandwidth because traffic only needs to travel a short distance. However, maintaining a high number of nodes will be stressful for providers. Last, forming a three-tier architecture by combining all tiers appears less suitable because it results in greater latency and because too many federations are difficult to manage and control.

Current research on intrusion detection as a service architecture is still in its infancy. However, a comparison with relevant studies is already possible. [7] and [20], which used fog as the architecture for their multi-stage ML-based IDS, were able to finish processing 12597 events in 28.57 and 66 microseconds, respectively, on an NSL-KDD dataset. Our task assignment 1 resembles configurations in [7] and [20], but the capacity of those architectures is unclear because they only implemented the system on a personal computer and referred to it as fog computing. They also did not consider a pre-processing stage, which is required to extract features from raw data.

We conclude that a two-tier architecture with edge and cloud is sufficient for Intrusion Detection as a Service, and assigning all tasks to the edge is the most efficient task assignment.

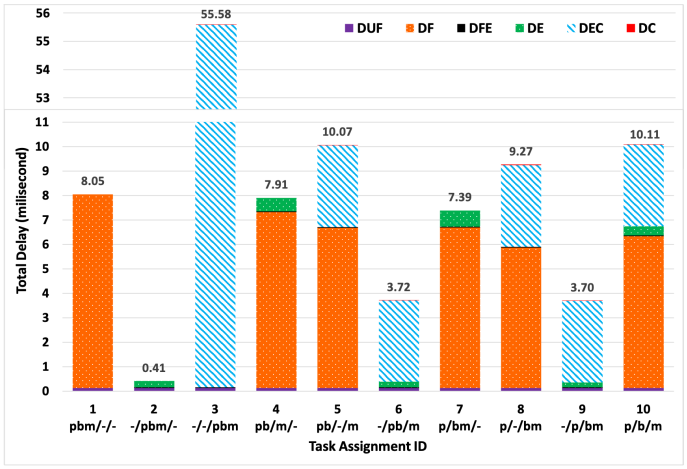


Fig. 3: One- vs. two- vs. three-tier architecture

2) Joint vs. separated task assignment: This investigation evaluates the effects of task assignments, whether the three tasks are combined in a single tier or separated into different tiers. Figure 3 shows that the performance of a joint configuration was better than separated task assignment in terms of total delay. Using task assignment 2 that used joint configuration as an example, performance was six times faster than the separated configuration that was used in task assignments 6 and 9. In a separated configuration, it is necessary to transmit traffic to the following tiers for further processing, which increases the total delay as a result of transmission and propagation delays.

Furthermore, because it does not need to transmit any traffic to the next tiers, a joint configuration may be able to save more bandwidth. For example, in task assignment 2, we just had to send traffic to edge. However, a separated configuration in task assignments 6 and 9 would require an additional 1.05 gigabits per second and 5.29 gigabits per second of traffic, respectively, to be sent to cloud.

We thus conclude that a joint configuration is preferable to separated configurations because it provides better performance while also utilizing less bandwidth.

3) Capacity allocation: upper vs. lower: Figure 4 shows the allocation of capacity based on the given total cost for each tier in all task assignments. The allocated capacity in fog and edge was then scattered between 8000 and 400 nodes, respectively. For example, in task assignment 1, the allocated capacity was scattered across 8000 nodes, which resulted in each fog node getting 312.5 MIPS.

Furthermore, in two and three-tiers, total capacity had to be allocated between the tiers. The results show that the algorithm tends to assign nodes more than 85% of the total capacity to the lowest-tier. When utilizing cloud as the highest-tier, the algorithm allocated 5% more capacity to its lowest tier nodes. As we know from queueing systems, even though a higher-tier computation node that is more centralized may result in longer communication delays, it requires less capacity than a distributed node in the lowest-tier.

The most important observation in this investigation is that more than 85% of the capacity is thus assigned to the lowest tier for pre-processing in order to reduce delays.

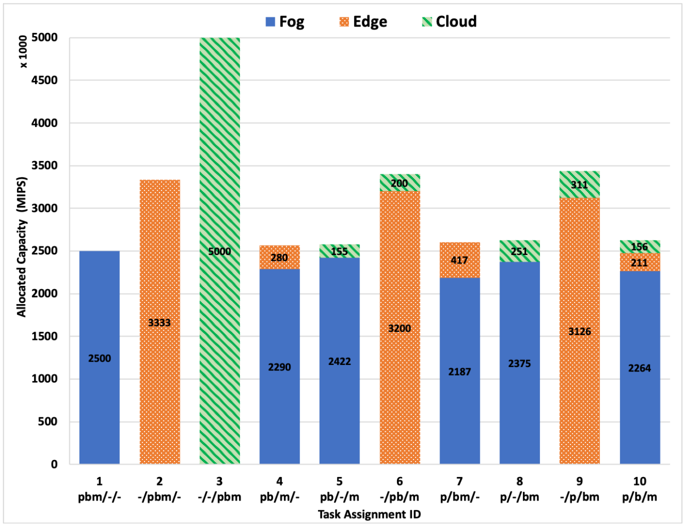


Fig. 4: Capacity Allocation: upper vs. lower

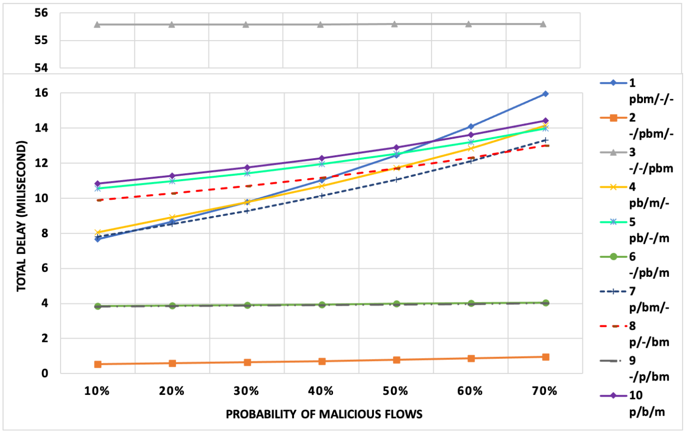
4) Effect of Malicious Traffic: This part of the investigation examines the effect of malicious traffic on the performance of task assignments when we increased malicious flows from 10% to 70%. As seen in Figure 5a, increasing the probability of malicious traffic had a direct impact on multi-class detection for several task assignments. Using only fog, or a fog-edge combination, had such a significant impact that the latency rose by 40-50%. On the other hand, combining fog with cloud had a lower impact than the previous combination.

Furthermore, employing cloud, or a combination of cloud and edge, had the least impact on their total delay. Cloud is more capable than fog and edge for maintaining consistent performance while dealing with an increasing volumes of malicious traffic.

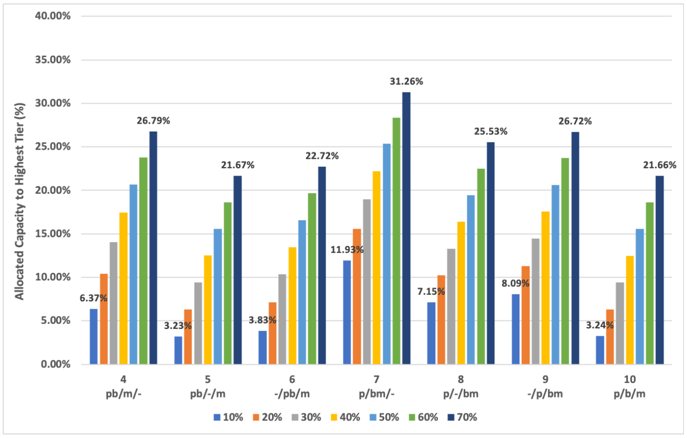
Figure 5b shows how much of the capacity of highest-tier was allocated to each task assignment, which is responsible for multi-class detection. Based on that figure, cloud requires 5% less capacity than the other tiers. As a result, by allocating additional capacity to a lower tier, the overall delay could be reduced. Despite the fact that the cloud is far away, processing multi-class traffic on the cloud helps to minimize delay better in situations where there is a very large volume of malicious traffic.

We conclude that processing multi-class traffic on the cloud reduces delay and keeps performance stable in cases where malicious traffic is high.

5) Effect of Flow Workload: The purpose of this part of the investigation was to examine the effects of flow workload on task assignment performance by utilizing a different sets of a machine learning algorithm for binary and multi-class detection. As seen in Figure 6, we noticed the effect on some task assignments only when we utilized DNN as the machine learning model for binary and multi-class detection.



(a) Total Delay



(b) Allocated Capacity to Highest-tier

Fig. 5: Effect of Malicious Traffic

Task assignments that utilize pre-processing and binary detection in fog (task assignments 1, 4, 5) had a greater impact than others. On the other hand, it was relatively stable for task assignments that used edge, cloud, and edge-cloud combinations (task assignments 2, 3, 6, and 9). Binary and multi-class detection also performed 40%–70% better on edge or cloud than in fog. It is reasonable because edge and cloud nodes have a greater capacity than fog nodes. As a result, it is preferable to handle binary and multi-class detection in an upper-tier, such edge or cloud when using machine learning models that need much computation.

The most important finding in this investigation is, when using computationally intensive machine learning models, it is preferable to handle binary and multi-class detection at the higher-tier.

6) Effect of Computation Cost: The investigation examined the effect of computation cost on the performance of task assignments. We used different amounts of total cost to determine its effects performance. Figure 7a shows the direct impact of computation cost on some task assignments. In the low-cost scenario, task assignments that use the edge or the edge-cloud combination, resulted in minimum delay.

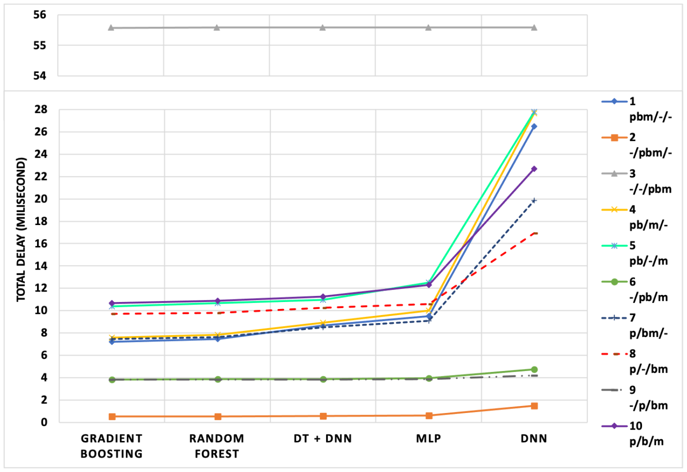


Fig. 6: Effect of Flow Workload

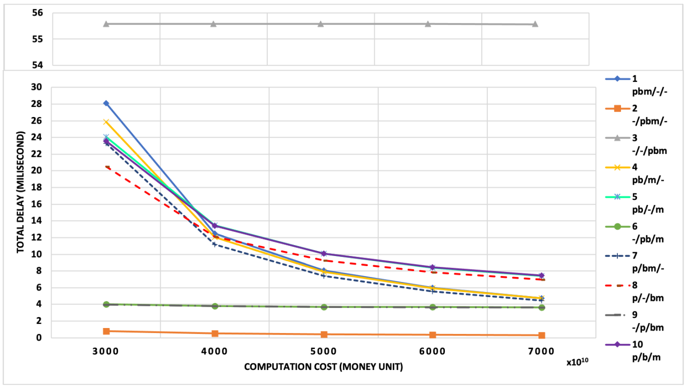
Task assignments that only use fog, resulted in a much higher delay - 5 times more than the task assignment that use edge-cloud combination. The reason for this is, as the computing costs decreases, the resource utilization increases. According to queuing theory, increased resource utilization extends the time spent in queues. Thus, offloading the task to a higher tier, such as an edge-cloud combination, is more effective in minimizing queuing time and speeding up computation. Figure 7b, which shows the allocation of capacity to each tier based on overall cost, illustrates that the task assignments that use edge and cloud, have at least 30 % more capacity than the task assignments that use fog. Utilizing the task assignments that use the edge or the edge-cloud combination, thus guarantee better performance in a low-cost scenario.

Furthermore, a fog, or fog-edge combination, can achieve comparable performance to an edge-cloud combination at highest computation costs. This is possible because higher computation costs result in increased capacity while decreasing resource utilization. As a result, no traffic will be queued in the system. Thus, the lowest tier is capable to handle the workloads.

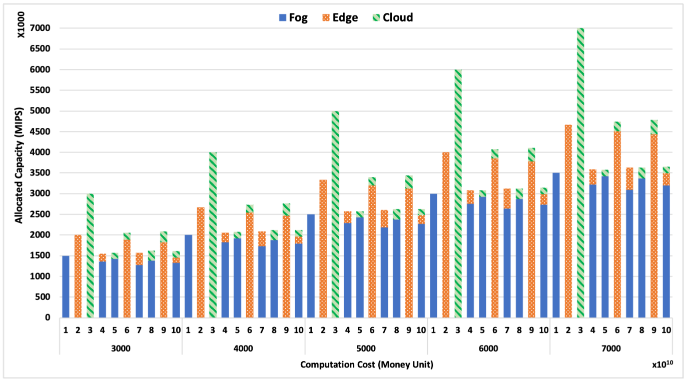
The results of this investigation thus show that using edge or edge-cloud combination are the best task assignment that have the consistent performance in all cases.

VI. Conclusions

We considered ten task assignments for distributed machine learning based IDS as a service. We used queueing theory to calculate the total delay, and a simulated annealing approach to allocate the optimum capacity to each tier. The results suggest that a service provider's concerns can influence the task assignments adopted. Each task assignment has its own advantages and disadvantages. The best task assignment with lower delay and stable performance under any conditions is achieved by utilizing only edge or edge with cloud. However, utilizing fog with edge may be considered by service providers concerned about tenant data privacy. Utilizing fog with cloud could potentially also be considered as they are easier to manage than fog with edge.



(a) Total Delay



(b) Allocated Capacity based on Computation Cost

Fig. 7: Effect of Computation Cost

Moreover, a joint configuration could have a better delay and save more bandwidth than separate task assignment. In order to to reduce delays, our results show that 85% of total capacity is provided to the lowest-tier for pre-processing.

Furthermore, when there is more than 50% malicious traffic, processing multi-class traffic in the cloud can effectively reduce latency and maintain a fairly stable performance. Next, when utilizing machine learning models, such as DNN, that need a large amount of computation, by handling binary and multi-class detection in the upper-tier can result in 40%-70% better performance. Finally, when computing costs are minimal, offloading the task to a higher tier, such as an edge-cloud combination, is more beneficial in terms of reducing queueing time and maximizing computing performance. Fog or fog-edge combinations, on the other hand, can achieve an equivalent performance to the prior combination by doubling the computation cost.

Work that needs to be considered in the future includes the following: First, the use of a three-tier network with slicing the nodes and offloading the traffic in ten directions based on ten task assignments, followed by the optimization of traffic split offloading ratios, should be investigated. Second, it is possible to do further research into the optimization of cost allocation with the goal of reducing the overall computation cost, subject to the delay threshold.

Furthermore, the real implementation of task assignment and capacity allocation for IDS is complicated and out of scope of this paper. In the future, we will consider developing a service framework on top of the Kubernetes platform, which will include intrusion detection as a service as application scenario.

References

[1] Y.-C. Lai, D. Sudyana, Y.-D. Lin, M. Verkerken, L. D'hooge, T. Wauters, B. Volckaert, and F. De Turck, "Machine learning based intrusion detection as a service: Task assignment and capacity allocation in a multi-tier architecture," in *Proceedings of the 14th IEEE/ACM International Conference on Utility and Cloud Computing Companion*, ser. UCC '21, Leicester, United Kingdom: Association for Computing Machinery, 2021.

[2] S. Gamage and J. Samarabandu, "Deep learning methods in network intrusion detection: A survey and an objective comparison," *Journal of Network and Computer Applications*, vol. 169, no. July, p. 102 767, 2020.

[3] H. Liu and B. Lang, "Machine learning and deep learning methods for intrusion detection systems: A survey," *Applied Sciences (Switzerland)*, vol. 9, no. 20, 2019.

[4] P.-H. Kuo, A. Mourad, C. Lu, *et al.*, "An integrated edge and fog system for future communication networks," in *2018 IEEE Wireless Communications and Networking Conference Workshops (WCNCW)*, 2018, pp. 338–343.

[5] M. T. Thai, Y. D. Lin, Y. C. Lai, and H. T. Chien, "Workload and Capacity Optimization for Cloud-Edge Computing Systems with Vertical and Horizontal Offloading," *IEEE Transactions on Network and Service Management*, vol. 17, no. 1, pp. 227–238, 2020.

[6] F. A. Khan, A. Gumaei, A. Derhab, and A. Hussain, "A novel two-stage deep learning model for efficient network intrusion detection," *IEEE Access*, vol. 7, pp. 30 373–30 385, 2019.

[7] C. Antonio, D. Souza, C. Becker, R. Bobsin, J. Bosco, M. Sobral, and S. Vieira, "Hybrid approach to intrusion detection in fog-based IoT environments," *Computer Networks*, vol. 180, no. May, p. 107 417, 2020.

[8] G. Bovenzi, G. Aceto, D. Ciuonzo, V. Persico, and A. Pescapé, "A hierarchical hybrid intrusion detection approach in iot scenarios," in *GLOBECOM 2020 - 2020 IEEE Global Communications Conference*, 2020, pp. 1–7.

[9] C. F. T. Pontes, M. M. C. de Souza, J. J. C. Gondim, M. Bishop, and M. A. Marotta, "A new method for flow-based network intrusion detection using the inverse potts model," *IEEE Transactions on Network and Service Management*, vol. 18, no. 2, pp. 1125–1136, 2021.

[10] H. Li and Q. Wu, "A distributed intrusion detection model based on cloud theory," in *2012 IEEE 2nd International Conference on Cloud Computing and Intelligence Systems*, vol. 01, 2012, pp. 435–439.

[11] Y. Mehmood, M. A. Shibli, A. Kanwal, and R. Masood, "Distributed intrusion detection system using mobile agents in cloud computing environment," in *2015 Conference on Information Assurance and Cyber Security (CIACS)*, 2015, pp. 1–8.

[12] A. Diro and N. Chilamkurti, "Leveraging LSTM Networks for Attack Detection in Fog-to-Things Communications," *IEEE Communications Magazine*, vol. 56, no. 9, pp. 124–130, 2018.

[13] A. Samy, H. Yu, and H. Zhang, "Fog-Based Attack Detection Framework for Internet of Things Using Deep Learning," *IEEE Access*, vol. 8, pp. 74 571–74 585, 2020.

[14] Y. Labiod, A. Amara Korba, and N. Ghoualmi, "Fog Computing-Based Intrusion Detection Architecture to Protect IoT Networks," *Wireless Personal Communications*, 2022.

[15] S. Li, Y. Lu, and J. Li, "Cad-ids: A cooperative adaptive distributed intrusion detection system with fog computing," in *2022 IEEE 25th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, 2022, pp. 635–640. DOI: 10.1109/CSCWD54268.2022.9776147.

[16] P. Singh, A. Kaur, G. S. Aujla, R. S. Batth, and S. Kanhere, "DaaS: Dew Computing as a Service for Intelligent Intrusion Detection in Edge-of-Things Ecosystem," *IEEE Internet of Things Journal*, p. 1, 2020.

[17] S. A. Rahman, H. Tout, C. Talhi, and A. Mourad, "Internet of Things Intrusion Detection: Centralized, On-Device, or Federated Learning?" *IEEE Network*, vol. 34, no. 6, pp. 310–317, 2020.

[18] R. Zhao, Y. Yin, Y. Shi, and Z. Xue, "Intelligent intrusion detection based on federated learning aided long short-term memory," *Physical Communication*, vol. 42, p. 101 157, 2020.

[19] W. Lalouani and M. Younis, "Robust distributed intrusion detection system for edge of things," in *2021 IEEE Global Communications Conference (GLOBECOM)*, 2021, pp. 01–06.

[20] M. Almiani, A. AbuGhazleh, A. Al-Rahayfeh, S. Atiewi, and A. Razaque, "Deep recurrent neural network for IoT intrusion detection system," *Simulation Modelling Practice and Theory*, vol. 101, p. 102 031, 2020.

[21] L. D'hooge, T. Wauters, B. Volckaert, and F. De Turck, "Inter-dataset generalization strength of supervised machine learning methods for intrusion detection," *Journal of Information Security and Applications*, vol. 54, 2020.

[22] R. Vinayakumar, M. Alazab, K. P. Soman, P. Poornachandran, A. Al-Nemrat, and S. Venkatraman, "Deep learning approach for

intelligent intrusion detection system," *IEEE Access*, vol. 7, pp. 41 525–41 550, 2019.

[23] A. Lashkari, G. Drapper, and M. S. Islam, *Cicflowmeter*. [Online]. Available: https://github.com/ahlashkari/CICFlowMeter.

[24] J. Madden, *Analysis: The economics of edge computing*, 2020. [Online]. Available: https://edgecomputing-news.com/2020/10/29/analysis-economics - of - edge - computing/ (visited on 02/01/2022).

[25] P. Megyesi and S. Molnár, "Analysis of elephant users in broadband network traffic," in *Advances in Communication Networking*, T. Bauschert, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 37–45.