Machine Learning based Intrusion Detection as a Service: Task Assignment and Capacity Allocation in a Multi-tier Architecture

Yuan-Cheng Lai
National Taiwan University of Science and Technology
Taipei, Taiwan
laiyc@cs.ntust.edu.tw

Didik Sudyana
National Yang-Ming Chiao-Tung University
Hsinchu, Taiwan
dsudyana@cs.nctu.edu.tw

Ying-Dar Lin
National Yang-Ming Chiao-Tung University
Hsinchu, Taiwan
ydlin@cs.nctu.edu.tw

Miel Verkerken
Ghent University-IMEC
Ghent, Belgium
miel.verkerken@ugent.be

Laurens D'hooge
Ghent University-IMEC
Ghent, Belgium
laurens.dhooge@ugent.be

Tim Wauters
Ghent University-IMEC
Ghent, Belgium
tim.wauters@ugent.be

Bruno Volckaert
Ghent University-IMEC
Ghent, Belgium
bruno.volckaert@ugent.be

Filip De Turck
Ghent University-IMEC
Ghent, Belgium
filip.deturck@ugent.be

ABSTRACT

Intrusion Detection Systems (IDS) play an important role for detecting network intrusions. Because the intrusions have many variants and zero days, traditional signature- and anomaly-based IDS often fail to detect it. Machine learning (ML), on the other hand, has better capabilities for detecting variants. In this paper, we adopt ML-based IDS which consists of three in-sequence tasks: pre-processing, binary detection, and multi-class detection. We proposed ten different task assignments, which map these three tasks into a three-tier network for distributed IDS. We evaluated these with queueing theory to determine which tasks assignments are more appropriate for particular service providers. With simulated annealing, we allocated the total capacity appropriately to each tier. Our results suggest that the service provider can decide on the task assignments that best suit their needs. Only edge or a combination of edge and cloud could be utilized due to their shorter delay and greater operational simplicity. Utilizing only the fog or a combination of fog and edge remains the most private, which allows tenants to not have to share their raw private data with other parties and save more bandwidth. A combination of fog and cloud is easier to manage while still addressing privacy concerns, but the delay was 40% slower than the fog and edge combination. Our results also indicate that more than 85% of the total capacity is allocated and spread across nodes in the lowest tier for pre-processing to reduce delays.

CCS CONCEPTS

• **Security and privacy → Intrusion detection systems**; • **Computer systems organization → Distributed architectures**.

KEYWORDS

ML-based IDS, multi-tier architecture, multi-stage machine learning

UCC '21 Companion, December 6–9, 2021, Leicester, United Kingdom
© 2021 Association for Computing Machinery.
ACM ISBN 978-1-4503-9163-4/21/12...$15.00
https://doi.org/10.1145/3492323.3495613

ACM Reference Format:
Yuan-Cheng Lai, Didik Sudyana, Ying-Dar Lin, Miel Verkerken, Laurens D'hooge, Tim Wauters, Bruno Volckaert, and Filip De Turck. 2021. Machine Learning based Intrusion Detection as a Service: Task Assignment and Capacity Allocation in a Multi-tier Architecture. In *2021 IEEE/ACM 14th International Conference on Utility and Cloud Computing (UCC '21) Companion (UCC '21 Companion), December 6–9, 2021, Leicester, United Kingdom.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3492323.3495613

1 INTRODUCTION

Nowadays, intrusion detection systems (IDS) are commonly used to protect network devices. In general, IDS can be classified as signature-based or anomaly-based. Signature-based IDSs often fail to detect an unknown attacks while anomaly-based IDS have high false positive (FP) rates. To address those problems, researchers have begun to use machine learning in IDS that has ability to detect more attack variants and novel attacks [7].

The use of IDS in a network is essential so as to ensure security. However, retaining an IT crew 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 service, and minimize operational costs. From the perspective of service providers, 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 of transferring traffic to centralized IDS.

A multi-tier architecture with bottom-tier, middle-tier, and top-tier is a potential solution to a scalable IDS. Computation can be distributed over tiers and brought closer to users. As a result, multi-tier architecture may solve the problems of high latency and bandwidth. The integration of each tier might also result in better performance, coverage, and scalability [13]. The combination of Fog with Multi-access Edge Computing (MEC), or Fog with Edge and Cloud, are illustrative of multi-tier architecture. For easier description and simplicity, this model will use the terms Fog, Edge, and Cloud throughout the article, and follows the tier hierarchy from [6].

Furthermore, machine learning based IDS consists of several tasks, from pre-processing the raw data to detecting an attack. Pre-processing is an important task for improving the quality of the data.

Table 1: 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

There are also 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. Such a single-stage binary suffers from overfitting and a strongly biased model [5]. Also, such existing multi-class detection is not accurate enough [2]. Combining these attack detection approaches can help to reduce model instabilities [5].

Starting with pre-processing, we use three tasks to perform ML-based IDS with binary and multi-class detection. Binary detection classifies traffic as benign or malicious and focuses on preventing overfitting and reducing the bias towards normal traffic. Malicious traffic only will then be transferred so as to classify the attack class in multi-class detection system. This combination has been shown to improve accuracy [5] and save bandwidth by not transferring all traffic. 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 employed in mostly machine learning-based IDS and improve real-time traffic classification performance [10]. Here, we break down these three tasks and map them to the architecture, resulting in ten different task assignments, as shown in Table 1. The values p, b, m represent the pre-processing, binary, and multi-class, respectively. We also used IDs and abbreviations to make it easier to recognize the ten task assignments.

A variety of research papers have dealt with distributed IDS from signature-based to machine learning-based. Mehmood et al. [9] considered a distributed IDS using a signature-based system in the cloud environment and applying it into a host-based IDS system.

Most studies focused on improving IDS accuracy and testing a model in a distributed architecture. For example, Diro and Chilamkurti [4] investigated applying LSTM model in a distributed environment, while Samy et al. [12] assessed six deep learning models before implementing them in a distributed architecture, and comparing their performance to a centralized design. Rahman et al. [11] and Zhao et al. [15] evaluated federated learning performance for distributed IDS and achieved some improvement in accuracy and training time performance.

Almiani et al. [1], Antonio et al. [2], Samy et al. [12] focused a one-tier architecture in fog, while other studies (Mehmood et al. [9], Rahman et al. [11], Zhao et al. [15]) focused on two-tier architecture. In a one-tier architecture, deploying on fog nodes reduces response time by bringing the processing node closer to the user. Fog has acceptable latency but typically insufficient processing power to deal with large computing processes. Furthermore, most research applied a second tier just as a coordinator or aggregation node.

Our goal, however, differs from this earlier research, and is to develop a multi-tier distributed IDS. We simulated and analyzed ten task assignments using queueing theory. Most previous studies focused just one architecture. To determine the minimum delay among the ten task assignment options, we optimized capacity allocation for each tier. We also measured the computational load of machine learning-based IDS and used it in the simulation exercise. This real-measurements helps in bringing the model to behave closer to real-life.

The system determined the capacity allocation for each architecture, based on arrival traffic rates and overall capacity. To broaden our knowledge, we also investigated (1) one- vs. two- vs. three-tier architecture; (2) joint vs. separated task assignment; (3) fog vs. edge vs. cloud capacity allocation.

The remainder of this paper is organized as follows. Section II defines the system architectures and problem formulation. The solution algorithm, is described in Section III. Section IV presents the simulation and results, and Section V concludes the work.

2 SYSTEM ARCHITECTURE AND PROBLEM FORMULATION

This section describes multi-tier architectures, gives problem statements, problem descriptions, and a delay model, with the variables and notations used are given in Table 2.

2.1 System architecture

The system architecture is shown in Figure 1 where the multi-tier architecture is composed of a cloud, N number of edge servers E, and K number of fog servers F beneath the edge. We then map the IDS tasks as i into architectures.

As can be seen in table 1, 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 $1/\mu_1^{\mathrm{L}}$, it is immediately processed in the pre-processing stage. This stage is used to extract features and clean the data before performing machine learning detection, and reduces the flow length into $1/\mu_2^{\mathrm{L}}$. The data is then sent through binary detection to detect benign and malicious traffic. Only malicious traffic with a probability of p^A and flow length $1/\mu_3^{\mathrm{L}}$ will then be passed to the multi-class detection system for the attack types to be classified.

Each of the task assignments utilized the same amount of total capacity C. Then, depending on the task assignment chosen, C will be allocated to 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.

2.2 Problem statement

Our objective is to minimize total delay and determine the best of the ten possible task assignments. To achieve this, capacity has to be appropriately allocated. The problem statement is then defined as:

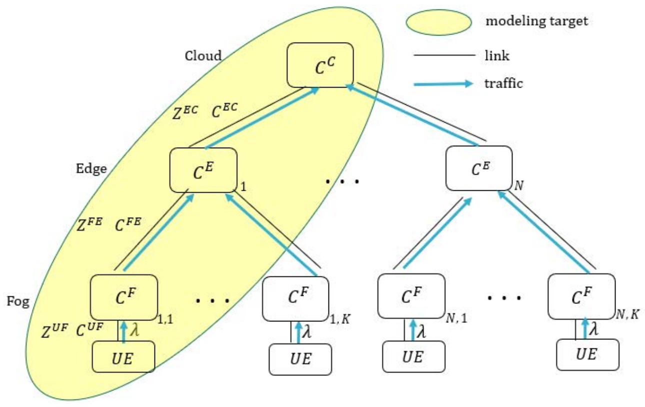

Figure 1: Parameter employed in the architecture model

Table 2: Model Notations

Notation	Meaning
N	The number of edge nodes
K	The number of fog nodes per edge node
$i = \{i \in 1, 2, 3\}$	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 node
C^E	Capacity of edge nodes
C^F	Capacity of fog nodes
C^{UF}	Link bandwidth from UE to fog
C^{FE}	Link bandwidth from fog to edge
C^{EC}	Link bandwidth from edge to cloud
λ	Arrival traffic rate from UE
Z^{UF}	Distance between UE and fog
Z^{FE}	Distance between fog and edge
Z^{EC}	Distance between edge and cloud
X	Speed of light
D	Total delay
p^A	Probability of malicious flows

Given: a topology composed of N edge servers, K fog servers under an edge server, 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}), and a distance between tiers (Z^{UF}, Z^{FE}, and Z^{EC}).

We need to determine the capacity allocations [C^F, C^E, C^C] which depend on architecture used as outputs, with the objective of minimizing the total delay D and subject to the constraints $C^F + C^E + C^C = C$.

2.3 Delay model

In this model, the total capacity for each of the task assignments is equal, and each tier's node will then have the same capacity. The delay will be calculated using an M/M/1 queueing model. The delay components, such as flow workload and length used, arrival rate, and capacity, vary among task assignments j in the r-th resource, as listed in Table 3 $V[j,r]$, Table 4 $H[j,r]$, and Table 5 $Y[r]$, respectively. For example, in task assignment 1, traffic is transmitted from UE to a fog link with flow length $1/\mu_1^L$ in $V[1,1]$, and arrival rate λ in $H[1,1]$, and with C^{UF} capacity in $Y[1]$. The task is handled at the fog node with $V[1,4]$ flow workload, $H[1,4]$ arrival rate, and $Y[4]$ computing capacity.

Depending on the task assignment utilized, the computation and communication delays occur at each of the fog, edge, and cloud based on the tasks that are assigned. The total delay for each task assignment j may then be determined by adding up all delay components, which can be expressed as:

$$D_j = \sum_{r=1}^{6} b(j,r) + \sum_{r=7}^{9} m(j,r) \tag{1}$$

where

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

$$m(j,r) = \begin{cases} \frac{V[j,r]}{X}, & \text{if } V[j,r] \neq \phi, \\ 0, & \text{if } V[j,r] = \phi. \end{cases} \tag{3}$$

3 SOLUTION APPROACH: CAPACITY ALLOCATION OPTIMIZATION

The problem set out above is solved by implementing the Capacity Allocation Optimization Algorithm. First, we provide initial capacity for each tier and calculate the average delay using M/M/1. The algorithm then determines the best capacity allocation for all task assignments, and allocates the capacity to each tier based on the overall capacity C and, adjusts the capacity until the system exhibits a minimum delay.

The 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 the new capacity allocation. We generate a new capacity allocation for C^F, C^E, and C^C in each iteration by applying randomization with specified constraints to minimum and maximum values. When generating a new solution for C^F, the minimum number is C^F_{min} to prevent running out of capacity, which might result in a negative delay. C^F_{min} is calculated as ($C^F_{min} = H[j,4] \times V[j,4]$). Then, the maximum number is C^F_{max} and can be calculated as ($C^F_{max} = C - (C^E_{min} + C^C_{min})$), which provides sufficient space capacity for C^E and C^C. Further, when producing a random number for C^E, the minimum values of C^E_{min} can be calculated as ($C^E_{min} = H[j,7] \times V[j,7]$), and the maximum values of C^E_{max} can be calculated as ($C^E_{max} = C - (C^F + C^C_{min})$). The residual number is therefore the value of C^C.

4 SIMULATION AND RESULTS

4.1 Measurements for Realistic Parameters Values

To run the simulation, certain parameters from a real system must be collected. The workloads used in this study come from a real IDS system. We built a machine learning-based intrusion detection

Table 3: Task Size in r-th resource for the j-th task assignment, $V[j,r]$

$V[j,r]$	Communication			Computation (workload)			Distance		
	$r=1$	$r=2$	$r=3$	$r=4$	$r=5$	$r=6$	$r=7$	$r=8$	$r=9$
	D^{UF}	D^{FE}	D^{EC}	D^{F}	D^{E}	D^{C}			
$V[1,r]$	$1/\mu_1^L$	ϕ	ϕ	$1/\mu_1^w + 1/\mu_2^w + (1/\mu_3^w.p^A)$	ϕ	ϕ	Z^{UF}	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	Z^{UF}	Z^{FE}	ϕ
$V[3,r]$	$1/\mu_1^L$	$1/\mu_1^L$	$1/\mu_1^L$	ϕ	ϕ	$1/\mu_1^w + 1/\mu_2^w + (1/\mu_3^w \cdot p^A)$	Z^{UF}	Z^{FE}	Z^{EC}
$V[4,r]$	$1/\mu_1^L$	$1/\mu_3^L$	ϕ	$1/\mu_1^w + 1/\mu_2^w$	$1/\mu_3^w \cdot p^A$	ϕ	Z^{UF}	Z^{FE}	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	$1/\mu_3^w \cdot p^A$	Z^{UF}	Z^{FE}	Z^{EC}
$V[6,r]$	$1/\mu_1^L$	$1/\mu_1^L$	$1/\mu_3^L$	ϕ	$1/\mu_1^w + 1/\mu_2^w$	$1/\mu_3^w \cdot p^A$	Z^{UF}	Z^{FE}	Z^{EC}
$V[7,r]$	$1/\mu_1^L$	$1/\mu_2^L$	ϕ	$1/\mu_1^w$	$1/\mu_2^w + (1/\mu_3^w.p^A)$	ϕ	Z^{UF}	Z^{FE}	ϕ
$V[8,r]$	$1/\mu_1^L$	$1/\mu_2^L$	$1/\mu_2^L$	$1/\mu_1^w$	ϕ	$1/\mu_2^w + (1/\mu_3^w \cdot p^A)$	Z^{UF}	Z^{FE}	Z^{EC}
$V[9,r]$	$1/\mu_1^L$	$1/\mu_1^L$	$1/\mu_2^L$	ϕ	$1/\mu_1^w$	$1/\mu_2^w + (1/\mu_3^w \cdot p^A)$	Z^{UF}	Z^{FE}	Z^{EC}
$V[10,r]$	$1/\mu_1^L$	$1/\mu_2^L$	$1/\mu_3^L$	$1/\mu_1^w$	$1/\mu_2^w$	$1/\mu_3^w \cdot p^A$	Z^{UF}	Z^{FE}	Z^{EC}

Table 4: Arrival rate in the r-th resource, $H[j,r]$

$H[j,r]$	Communication			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	$\lambda K(p^A)$		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 5: 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 1: SA for Capacity Allocation

```
initialize $C^F, C^E, C^C, t_{ini} = 1000, \alpha = 0.085$;
$t = t_{ini}$;
calculate $D$;
while $t > t_{trm}$ do
    generate new solutions for $C^F$, $C^E$, $C^C$ by;
        $C^{F'} = rand(C^F_{min}, C^F_{max})$;
        $C^{E'} = rand(C^E_{min}, C^E_{max})$;
        $C^{C'} = C - C^{F'} - C^{E'}$;
    calculate $D'$;
    $\Delta D = D' - D$ ;
    if $D' \leq D$ then
        accept new $C^F = C^{F'}, C^E = C^{E'}, C^C = C^{C'}, D = D'$;
    else
        accept new $C^F = C^{F'}, C^E = C^{E'}, C^C = C^{C'}, D = D'$, with
          probability $\frac{1}{exp(\Delta D/t)}$;
    end
    decrease the temperature: $t = \alpha \times t$;
end
```

pipeline that consists 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 Core i5-3470 3.2GHz CPU.

For binary detection, we had to identify malicious quickly while maintaining performance. A decision tree is suitable for this criterion because it is fast and accurate [3]. For multi-class detection, we used deep learning since it is more capable of distinguishing an intrusion and also effective for detecting unforeseen intrusions [14].

We constructed Decision Trees with default hyperparameters and Deep Neural Networks from [14] with only malicious traffic trained using CICIDS-2017 as the dataset. Pre-processing involved extracting features from raw traffic using CICFlowMeter, cleaning the data, and parsing it correctly. The data were also rescaled using Scikit-MinMaxScaler. After 10 tests, we found that pre-processing took 1,129,740 instructions, binary detection 113,989, and multi-class detection 1,053,851 instructions. We also observed that after pre-processing, flow length decreased from 100% to 5.3%.

4.2 Parameters settings

Table 6 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 [13] and the flow length for pre-processing was taken from [8].

4.3 Results

4.3.1 One- vs. two- vs. three-tier architecture. Figure 2 shows the comparative results of the ten task assignments with the total delay for processing a flow. The result shows that utilizing only the edge as in task assignment 2 had the shortest delay in processing the traffic. This task assignment 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 produced poor performance which increases computation time by 15. On the other hand, using just cloud node maximizes computing capacity, speeds up computing

Table 6: Parameter settings

Notations	Value	Unit
N	400	Nodes
K	20	Nodes
λ	100	Flows/second
p^A	0.2	
$1/\mu_1^W$	1,129,740	Instructions per flow
$1/\mu_2^W$	113,989	Instructions per flow
$1/\mu_3^W$	1,053,851	Instructions per flow
$1/\mu_1^L$	124.99	Kilobit
$1/\mu_2^L$, $1/\mu_3^L$	6.62	Kilobit
C	2.4×10^6	MIPS
Z^{UF}	1	Kilometer
Z^{FE}	10	Kilometers
Z^{EC}	1000	Kilometers
C^{UF}	1	Gbps
C^{FE}, C^{EC}	100	Gbps

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 the lower-tier would reduce a large transmission delay. 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 had 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 16% reduction in delay. This combination also saves bandwidth because traffic only needs to travel a short distance. However, maintaining a high number of nodes will be a strain for providers. Finally, 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.

Service providers can choose the task assignments that best suit their needs. Utilizing only edge or edge with cloud may be the fastest and simplest to use. Concerns about privacy may be addressed by utilizing fog with edge. However, the delay was two times slower than utilizing edge with cloud and maintenance was challenging because 8000 fog and 400 edge nodes were involved. Therefore, utilizing fog with cloud simplifies maintenance while maintaining privacy, but is 16% slower than fog with edge.

4.3.2 Joint vs. separated task assignment. This section evaluates the effects of task assignment, whether the three tasks are combined in a single tier or separated into several tiers. Figure 3 shows the computation delays based on task assignment. The computation time was faster when three tasks were separated. Comparing the joint configuration in task assignment 1 to the separated configuration in task assignments 5, 8, and 10, the separated configuration was 11-27% faster. A higher-tier node has more capacity than a lower-tier

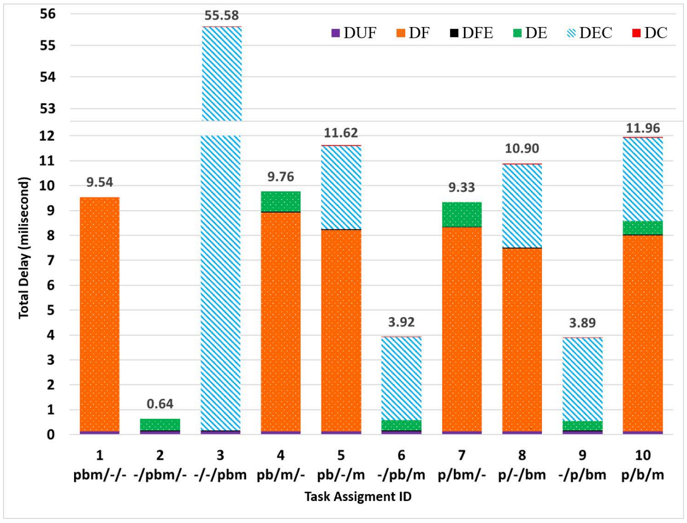

Figure 2: One- vs. two- vs. three-tier architecture

node. Thus, assigning some tasks to a higher-tier would increase performance. Despite the increased propagation delay caused by higher-tier distance, the separated configuration showed improved system utilization, indicating better efficiency in managing traffic.

However, when separating the tasks, we must consider workload and tier location. As shown in task assignment 4, where we split multi-class task into edge, separating too few workloads make a separated configuration less effective than a joint configuration.

Furthermore, a separated configuration that places pre-processing alone in the lower-tier and a binary with multi-class configuration in the upper-tier will be easier for maintaining a machine learning model. For example, to update or retrain it, a provider just needs to update a new model with fewer nodes than when dividing binary and multi-class models into different tiers.

On the other hand, allocating pre-processing with binary detection to the same tier and multi-class to an upper-tier increases privacy since the raw traffic is handled closer to the users and not aggregated. Furthermore, since only a very small amount of attacked traffic is sent to the farthest node, it may be able to save bandwidth. For example, in task assignment 6, we only had to send 1.05 gigabits of traffic per second to the cloud, but in task assignment 9, we had to send 5.29 gigabits per second.

4.3.3 Fog vs. edge vs. cloud capacity allocation. Figure 4 shows the capacity distribution for each tier in all task assignments. Task assignments 1-3 had no distribution because they only had a single-tier configuration, hence the entire capacity was dedicated to that tier. However, with task assignments 1 and 2, it was necessary to spread capacity over 8000 and 400 nodes, respectively. By comparison, task assignment 3 had the fastest processing time since all capacity was allocated to a single node.

Furthermore, in two and three-tiers, total capacity had to be allocated among the tiers. The results show that the algorithm tends to assign the lowest-tier nodes more than 85% of the total capacity. In task assignments 5 and 6, the algorithm allocated 5%

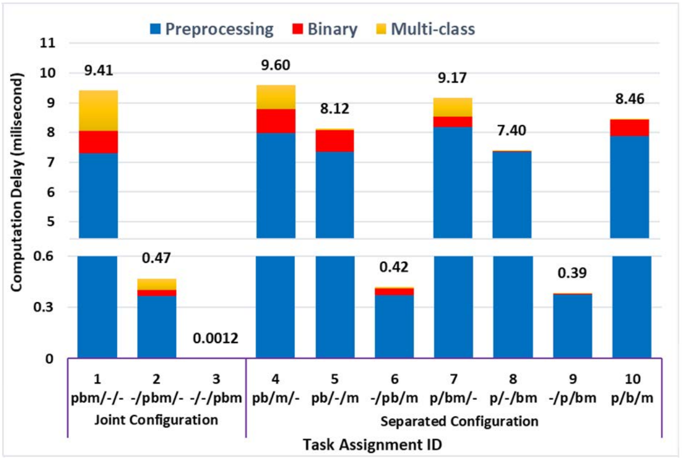

Figure 3: Joint vs. separated task assignment

more capacity to the lowest-tier when the cloud was the highest-tier, compared to task assignment 4, which used the edge as the highest-tier. Because the cloud suffers from long propagation delays, the algorithm assigns higher capacity to the lowest-tier to reduce delays. Further, 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. We also observed that moving binary detection from lowest-tier to higher-tier and combining it with multi-class, as in task assignment 4 compared to task assignment 7, only required 4-6% more capacity compared to a higher-tier. This means that pre-processing task should be given as much capacity as possible to minimize delay.

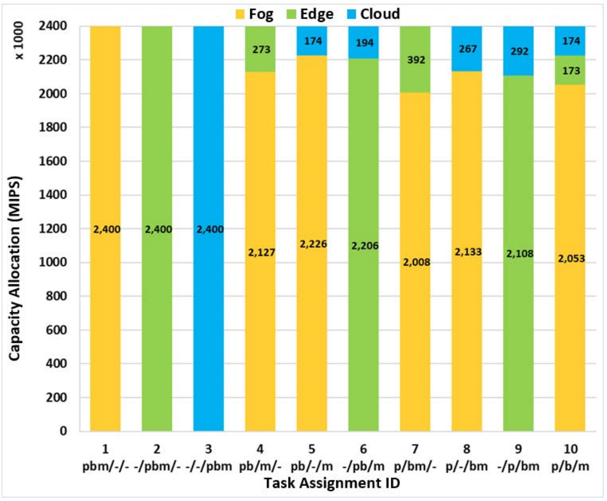

Figure 4: Fog vs. edge vs. cloud capacity allocation

5 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 for each tier. The results suggest that a service providers' concerns can influence the task assignments adopted. Each task assignment has its own advantages and disadvantages. Service providers should consider utilizing those task assignments that use only edge or edge with cloud for lower latency and easier management. 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. Moreover, the separated task assignment can be 11-27% faster than combining three tasks and placing them on the same tier. In separating the tasks, we need to consider the workload and tier location to make them more efficient. Separating the tasks into pre-processing alone or pre-processing with binary classification combined increases privacy and uses less bandwidth. Lastly, to reduce delays, the results show that 85% of total capacity is provided to the lowest-tier.

REFERENCES

[1] Muder Almiani, Alia AbuGhazleh, Amer Al-Rahayfeh, Saleh Atiewi, and Abdul Razaque. 2020. Deep recurrent neural network for IoT intrusion detection system. *Simulation Modelling Practice and Theory* 101 (2020), 102031.

[2] Cristiano Antonio, De Souza, Carlos Becker, Renato Bobsin, João Bosco, Mangueira Sobral, and Santos Vieira. 2020. Hybrid approach to intrusion detection in fog-based IoT environments. *Computer Networks* 180, May (2020), 107417.

[3] Laurens D'hooge, Tim Wauters, Bruno Volckaert, and Filip De Turck. 2020. Inter-dataset generalization strength of supervised machine learning methods for intrusion detection. *Journal of Information Security and Applications* 54 (2020).

[4] Abebe Diro and Naveen Chilamkurti. 2018. Leveraging LSTM Networks for Attack Detection in Fog-to-Things Communications. *IEEE Communications Magazine* 56, 9 (2018), 124–130.

[5] Farrukh Aslam Khan, Abdu Gumaei, Abdelouahid Derhab, and Amir Hussain. 2019. A Novel Two-Stage Deep Learning Model for Efficient Network Intrusion Detection. *IEEE Access* 7 (2019), 30373–30385.

[6] Ping-Heng Kuo, Alain Mourad, Chenguang Lu, Miguel Berg, Simon Duquennoy, Ying-Yu Chen, Yi-Huai Hsu, Aitor Zabala, Riccardo Ferrari, Sergio Gonzalez, Chi-Yu Li, and Hsu-Tung Chien. 2018. An integrated edge and Fog system for future communication networks. In *2018 IEEE Wireless Communications and Networking Conference Workshops (WCNCW)*. 338–343. https://doi.org/10.1109/WCNCW.2018.8369023

[7] Hongyu Liu and Bo Lang. 2019. Machine learning and deep learning methods for intrusion detection systems: A survey. *Applied Sciences (Switzerland)* 9, 20 (2019).

[8] Péter Megyesi and Sándor Molnár. 2013. Analysis of Elephant Users in Broadband Network Traffic. In *Advances in Communication Networking*, Thomas Bauschert (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 37–45.

[9] Yasir Mehmood, Muhammad Awais Shibli, Ayesha Kanwal, and Rahat Masood. 2015. Distributed intrusion detection system using mobile agents in cloud computing environment. In *2015 Conference on Information Assurance and Cyber Security (CIACS)*. 1–8.

[10] Camila F. T. Pontes, Manuela M. C. de Souza, João J. C. Gondim, Matt Bishop, and Marcelo Antonio Marotta. 2021. A New Method for Flow-Based Network Intrusion Detection Using the Inverse Potts Model. *IEEE Transactions on Network and Service Management* 18, 2 (2021), 1125–1136.

[11] Sawsan Abdul Rahman, Hanine Tout, Chamseddine Talhi, and Azzam Mourad. 2020. Internet of Things Intrusion Detection: Centralized, On-Device, or Federated Learning? *IEEE Network* 34, 6 (2020), 310–317.

[12] Ahmed Samy, Haining Yu, and Hongli Zhang. 2020. Fog-Based Attack Detection Framework for Internet of Things Using Deep Learning. *IEEE Access* 8 (2020), 74571–74585.

[13] Minh Tuan Thai, Ying Dar Lin, Yuan Cheng Lai, and Hsu Tung Chien. 2020. Workload and Capacity Optimization for Cloud-Edge Computing Systems with Vertical and Horizontal Offloading. *IEEE Transactions on Network and Service Management* 17, 1 (2020), 227–238.

[14] R. Vinayakumar, Mamoun Alazab, K. P. Soman, Prabaharan Poornachandran, Ameer Al-Nemrat, and Sitalakshmi Venkatraman. 2019. Deep Learning Approach for Intelligent Intrusion Detection System. *IEEE Access* 7 (2019), 41525–41550.

[15] Ruijie Zhao, Yue Yin, Yong Shi, and Zhi Xue. 2020. Intelligent intrusion detection based on federated learning aided long short-term memory. *Physical Communication* 42 (2020), 101157.