ML-Based Intrusion Detection as a Service: Traffic Split Offloading and Cost Allocation in a Multi-Tier Architecture

Didik Sudyana, Yuan-Cheng Lai, Ying-Dar Lin, *Fellow, IEEE*, and Piotr Chołda

***Abstract*—An Intrusion Detection System (IDS) employing machine learning (ML) solutions is crucial for identifying network intrusions. To minimize operational expenses and enhance performance, enterprises have begun outsourcing IDS management to service providers, giving rise to the concept of Intrusion Detection as a Service (IDaS). Earlier research primarily aimed at enhancing the accuracy of ML-based IDS models or expediting their computational process. However, from the service provider's perspective, an optimal architecture ensuring minimal computation cost and processing delay is crucial to increasing revenue. This study evaluates the performance of IDaS in a multi-tier architecture, utilizing traffic split offloading to enhance performance by mapping three in-sequence ML-based IDS tasks (pre-processing, binary detection, multi-class classification) to the architectures as the offloading destinations. We employ a simulated annealing-based traffic offloading and cost allocation (SA-TOCA) algorithm to determine the offloading ratio for each traffic path and the cost requirements for each tier. The results indicate that the edge-cloud architecture is 15% and four times more cost-effective compared to the fog-edge and fog-cloud architectures, respectively, and it demonstrates superior performance in minimizing processing delays. Offloading the majority of traffic to the edge and the remainder to the cloud proves to be an efficient strategy, reducing both computation costs and average delays.**

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

I. Introduction

AN INTRUSION Detection System (IDS) has historically relied on signature-based and anomaly-based methods. Over the past decade, machine learning (ML) has increasingly been adopted, offering enhanced capabilities for detecting varied and novel intrusion types [1]. Traditional ML-based IDS approaches have primarily focused on a single stage for either detection, which involves recognizing benign and malicious traffic using binary decisions, or classification, which refers to the process of classifying an object using a multi-class model and predefined classes. These configurations are prone to overfitting and model bias [2], and also can lead to increased computational costs and bandwidth requirements in distributed networks. To address these issues, combining binary and multi-class models has been proposed as a potential solution, with the aim of enhancing performance and improving model stability [2].

Furthermore, ML-based IDS performs several tasks, including preprocessing raw data and using a combined binary detection and multi-class classification approach. Preprocessing transforms raw input data for analysis. Binary detection classifies traffic as benign or malicious, reducing overfitting and bias. Malicious traffic is further classified by a multi-class system. This combination has been shown to increase accuracy [2], reduce false-positive rates [3], and minimize bandwidth utilization by selectively sending traffic [4].

Managing an IDS in-house is costly and time-consuming. Therefore, enterprises are outsourcing IDS management to service providers, reducing staffing needs and operational costs. This shift turns intrusion detection from a hardware-centric to a service-based model, enabling IDaS.

The architecture for IDaS should be considered from the perspective of the service provider as the operator. However, several previous studies [3], [7], [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18] have proposed distributed ML-based IDS without considering potential architecture issues. These studies aim to increase ML accuracy and response time in specific architectures. Although [19] addressed similar architectural concerns for implementing IDaS, our work not only differs in architecture models and configurations but also improves on them by enhancing system efficiency and adaptability to network conditions. Detailed comparisons and specific methodologies, including our unique approach to managing traffic and resources, are further elaborated in subsequent sections.

Traditional centralized architectures with cloud computing face limitations in supporting real-time, delay-sensitive ML services due to high network latency. A multi-tier distributed system, federating cloud, edge, and fog computing paradigms, can address this issue by reducing latency and enhancing performance.

However, different communities and standards may have varying definitions for edge and fog computing. While the IEEE definition, as part of the OpenFog Reference Architecture, uses the concept of cloud-fog-edge, we have based our architecture on the European Telecommunication Standards

Received 18 June 2023; revised 14 July 2024; accepted 19 August 2024. Date of publication 28 April 2025; date of current version 12 June 2025. *(Corresponding author: Didik Sudyana.)*

Didik 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.nycu.edu.tw; ydlin@cs.nycu.edu.tw).

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).

Piotr Chołda is with the Institute of Telecommunications, AGH University of Krakow, 30-059 Kraków, Poland (e-mail: piotr.cholda@agh.edu.pl).

Digital Object Identifier 10.1109/TSC.2025.3563680

Institute's (ETSI) [5] and 5G-Coral frameworks [6]. In our work, the cloud and edge belong to the service provider's coverage, whereas the fog is placed within the subscriber's domain. As a result, the edge computing we refer to aligns with the concept of Multi-access Edge Computing (MEC), which pertains to the edge of the service provider's network rather than the edge of the entire architecture.

The cloud tier is positioned at the top and is widely adopted for its scalability, rapid elasticity, resource pooling, cost-saving benefits, and ease of maintenance. Major providers, such as Google and Amazon, fall into this tier, handling large-scale data processing, business logic, and analytical databases. The edge tier serves as the intermediate layer, derived from the ETSI initiative to virtualize cloud capabilities within mobile networks. This tier refers to the edge of the service provider network, where edge servers are deployed near cellular base stations and central offices. Providers in this tier include Verizon, T-Mobile, and AT&T, utilizing local network assets, micro data centers, and base stations. The fog tier, located at the bottom, is situated near data sources and extends the cloud and edge infrastructures. This architecture belongs to the subscriber area and supports real-time data processing on industrial PCs, autonomous equipment, and local computing devices.

Within this work, while ML models form an integral part of the IDS, our primary focus is on the architecture that facilitates these models as a service. We examine the optimal architecture for deploying ML-based IDS, considering various computational and delay constraints. This paper emphasizes the architectural framework and operational strategies rather than detailing specific ML configurations.

We examine IDaS performance using multi-tier architecture and offer recommendations for service providers regarding management and control planes. These concepts serve different purposes in managing the performance of a network system. The management plane focuses on optimizing revenue by minimizing computation costs while adhering to delay constraints. On the other hand, the control plane is responsible for maintaining network stability and efficiency, as well as ensuring accurate and reliable traffic transmission by minimizing processing delays.

Efficiently managing operating expenses (OPEX) and capital expenditures (CAPEX) while maintaining service quality is crucial for IDaS under the management plane, as traffic workloads may increase. Constantly utilizing high capacity can lead to increased costs, so minimizing computation costs while ensuring delays remain below predefined thresholds is essential for maximizing service providers' revenue.

The control plane focuses on processing traffic swiftly, using offloading techniques for IDaS traffic. This involves distributing latency-sensitive workloads across tiers to maintain service quality. By leveraging a cloud-edge-fog federation, we split and distribute IDaS traffic across multiple paths to minimize delay. Tasks are separated and mapped to the architecture, creating various traffic path configurations. For example, traffic might be sent to the fog for preprocessing, the edge for binary detection, and the cloud for multi-class classification (illustrated in Fig. 1). The paper's key contribution is determining the optimal traffic ratio for offloading in each path.

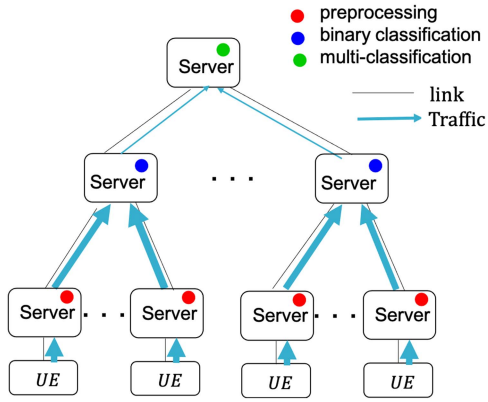


Fig. 1. Sketch of the offloading procedures.

To reduce computation cost or average delay, we allocate capacity to each tier during the offloading process. We model the IDaS system and optimize the offloading and cost allocation in two steps using the simulated annealing-based traffic offloading and cost allocation (SA-TOCA) algorithm. In the first step, we determine the offloading ratios for each traffic path with the goal of either minimizing computation cost (management plane) or minimizing average delay (control plane), depending on the optimization objective. In the second step, we assign the computational capacity for each task in each tier based on the current offloading ratios. Without appropriate offloading ratios and sufficient capacity allocation, the system may experience an increase in computation cost or average delay, making both steps essential to the optimization process.

The management plane's primary concern is identifying the architecture that achieves the lowest cost while maintaining delay thresholds. In contrast, the control plane focuses on determining the optimal architecture for IDaS performance while minimizing delays.

With relation to related work, this paper's contributions are three-fold: (1) to the best of our knowledge, while [19] also utilizes a mapping process of ML-based IDS tasks to architectures, they consider these mappings as mere configuration alternatives. In contrast, our study uniquely integrates these mappings as critical components of the traffic offloading process itself. This approach not only tailors the architecture to specific operational demands but also optimizes the overall system performance, (2) the use of an optimization process to calculate the ratio of offloading for each traffic path and the cost allocation for each tier, as well as the effect of various parameters on the performance of the architecture in running ML-based IDS, (3) the evaluation of the architecture for both management and control planes, focusing on minimizing cost and average delay, respectively, in order to provide service providers with appropriate recommendations.

In order to enhance our knowledge, we also examined (1) *architecture:* two-tier vs. three-tier, in order to identify the best architecture that has the minimum cost and minimum delay, (2) *traffic offloading ratios*, in order to determine the percentage of offloading in each traffic path, (3) *cost allocation*,

TABLE I
SUMMARY OF THE RESEARCH ON DISTRIBUTED IDS SUPPORTED WITH ML

Paper	IDS Type	Architecture	Jobs in IDS	Purpose	Method	A/S/I
[7]	Single-stage ML-based	Fog-Cloud	Full	Accuracy	Federated LSTM	I (ISCX intrusion dataset)
[8]		Fog-Cloud	Full	Accuracy	RNN	I (BoT-IoT dataset)
[9]		Fog	Full	Accuracy	RNN, LSTM, Bi-LSTM, GRU, CNN, CNN-LSTM	I (five datasets)
[10]		Fog	Full	Accuracy	Online Adaptive ML	I (Self-collected)
[11]		Dew-Cloud	Pre-processing/Binary	Accuracy	Deep belief network	I (UNSW-NB15 dataset)
[12]		IoT-Cloud	Full	Accuracy	FL	I (NSL-KDD dataset)
[13]		Two-level servers	Full	Accuracy	FL aided LSTM	I (AT&T SEA dataset)
[14]		Edge-Cloud	Full	Accuracy	FL with RL	I (UNSW-NB15 dataset)
[15]		Edge-Cloud	Full	Accuracy	FL with DNN for C-V2X networks	I (CIC-IDS-2018 dataset)
[3]	Multi-stage ML-based	IoT-Edge	Binary Classification/ Multi-classification	Accuracy	Multi-modal DAE, Soft output classifier	I (Bot-IoT Dataset)
[16]		Fog	Binary Classification/ Multi-classification	Accuracy	vAE with MLP	I (IoT-23 dataset)
[17]		Fog	2 Binary Classifications	Accuracy	DNN-KNN	I (NSL-KDD & CIC-IDS-2017 dataset)
[18]		Fog	2 Binary Classifications	Accuracy	RNN	I (NSL-KDD dataset)
[19]		Multi-tier	Pre-processing/ Binary Classification/ Multi-classification	Minimum Delay	Architecture selection with queueing theory	A
Ours SA-TOCA		Multi-tier	Pre-processing/ Binary Classification/ Multi-classification	Minimum Cost Minimum Delay	Traffic split offloading ratios and cost allocation with queueing theory	A

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

to determine how the algorithm distributes costs between tiers. We also evaluated the effects of several parameters on system performance: (4) *the effect of delay threshold*, which affects the allocated capacity in the architecture, (5) *the impact of malicious traffic*, which affects the performance of architecture when dealing with a huge volume of such traffic that must be handled, (6) *the impact of flow workload* on the performance of architecture while utilizing different ML models for binary and multi-class detection, (7) *the effect of computing cost*, specifically how the overall computation cost of a certain architecture might have an effect on its capacity and performance.

The remaining sections of this paper are structured as follows. Section II summarizes the previous works in distributed IDS. In Section III, the system architectures and problem formulation are defined. The solution algorithm is discussed in Section IV, followed by the simulation and results in Section V, and the conclusion in Section VI.

II. RELATED WORKS

Previous studies on utilizing distributed architecture for IDS are summarized in Table I, comparing IDS type, architecture, tasks, purpose, and approach. We differentiate between single-stage ML-based IDS, which use a single ML model for detection or classification, and multi-stage ML-based IDS that employ multiple ML models on the same data for enhanced accuracy.

Several studies have utilized single-stage ML-based approaches with distributed architecture for IDS. In [7], an LSTM model was used in a distributed setting, with training data spread over fog nodes and coordinated through a centralized node for better accuracy and scalability. [8] compared RNN and Bi-LSTM performance, proposing a framework where fog nodes handle pre-processing, feature extraction, and detection, while the cloud manages training, validation, and testing. [9] evaluated six DL models, finding LSTM the best, and deployed it in a distributed architecture with fog and cloud nodes, where fog handled detection and cloud received results upon attack detection. [10] applied an online adaptive ML model in fog nodes to enhance IDS quality and reduce workload, achieving faster response times. Studies in [12], [13], [14], [15] evaluated federated learning for distributed IDS, matching the accuracy of centralized solutions and preserving user privacy.

Several studies have utilized distributed IDS with fog nodes and two-stage ML detection for better accuracy and efficiency [3], [16], [17], [18]. In [17], binary classification was used, while [17] combined a variational AutoEncoder with a Multi-layer Perceptron to enhance detection efficiency. However, these models were only assessed for overhead in fog environments. Deploying IDS on fog nodes reduces response time by bringing processing closer to the user, but fog nodes often lack the power for extensive computations. [16] proposed a two-stage IDS for IoT-edge architecture but did not thoroughly investigate the architecture.

While previous research primarily focused on improving detection accuracy, our study, along with [3], emphasizes optimal architectures for running IDaS. Both works use a multi-tier architecture and multi-stage ML-based approach, adapting task assignment methodologies and modeling the architecture with queueing theory. However, our study differs in key aspects. (1) Unlike [19], which focuses solely on the control plane, our study addresses both control and management planes, aiming to minimize computation costs and system average delay through traffic split offloading and cost allocation. (2) We split incoming traffic and offload it to multiple destinations, employing all task assignments as offloading destinations, rather than using a single task assignment as in [19]. (3) Our approach uses a two-step optimization for offloading ratios and cost allocation with the SA-TOCA algorithm, while [19] employs a single-step

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_i^C	Capacity of cloud for i-th task
C_i^E	Capacity of edge for i-th task
C_i^F	Capacity of fog for i-th task
C_i^{UF}	Link bandwidth from UE to fog
C_i^{FE}	Link bandwidth from fog to edge
C_i^{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_i^C	Allocated cost to cloud for i-th task
U_i^E	Allocated cost to edge for i-th task
U_i^F	Allocated cost to fog for i-th task
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	Average delay
P^A	Probability of malicious flows
O_j	Offloading ratios of j-th destination
t	Temperature for Simulated Annealing Algorithm
t_{ini}	Initial Temperature
t_{trm}	Termination Temperature

optimization. (4) We model our architecture with sliced capacity for each ML-based IDS task, enhancing efficiency and scalability, which is not considered in [19]. We show the performance comparison between our work and [19] in Section V-D.

III. SYSTEM MODEL AND PROBLEM FORMULATION

This section provides an explanation of the traffic path for offloading, as well as multi-tier architectures, problem definitions, and a delay model. The variables and notations used are detailed in Table II.

A. System Architectures

The system architecture is presented in Fig. 2, where a multi-tier architecture with a three-tier network consisting of a cloud, N edge nodes E, and K fog nodes F under the edge is utilized. The IDS tasks are then represented by i and mapped to architectures.

The architecture has a total cost of S. Then, S is distributed to each tier based on the defined cost of each tier S^C, S^E, S^F to obtain the capacity of each tier for i-task C_i^C, C_i^E, and C_i^F with bandwidth capacities of each link for i-task C_i^{UF}, C_i^{FE}, and C_i^{EC}. The traffic passes at a rate of λ and is split into several directions to perform the offloading.

Traffic from user equipment (UE) arrives at the first node for pre-processing with a workload of $1/\mu_1^W$ and length $1/\mu_1^L$. This stage extracts features and cleans data, reducing the flow length to $1/\mu_2^L$. The data then undergoes binary detection to differentiate benign from malicious traffic, with a workload of

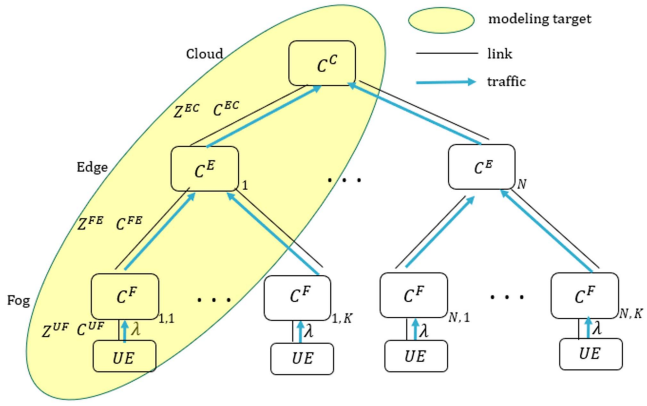


Fig. 2. Parameters employed in the architecture model.

$1/\mu_2^W$. We introduce a variable, P^A, representing the probability of malicious traffic, to quantify it. The malicious traffic is then forwarded for multi-class classification with a workload of $1/\mu_3^W$ and flow length $1/\mu_3^L$.

B. Traffic Path

We implement a configurable multi-tier architecture that supports traffic flow across one-, two-, or three-tier setups, resulting in seven architectural variants: fog, edge, cloud, fog-edge, fog-cloud, edge-cloud, and fog-edge-cloud. Within these architectures, we deploy three ML-based IDS tasks–pre-processing, binary detection, and multi-class classification–which are dynamically assigned across tiers. This setup yields 10 valid configurations, referred to as traffic paths, representing different combinations of task-to-tier mappings used as offloading destinations.

Each configuration is abbreviated for easy recognition (xxx/xxx/xxx), with sections representing fog, edge, and cloud, respectively. Symbols for pre-processing (p), binary detection (b), and multi-class classification (m) indicate task assignments. Unassigned sections are marked with "-". Fig. 1 shows traffic path 10, where tasks are assigned to fog, edge, and cloud.

It is important to note that our approach is based on traffic split offloading, where different tasks can be processed in different tiers. To enable this, the architecture must include more than one destination tier. Therefore, our focus is on identifying which multi-tier architectures benefit most from traffic split offloading, using the 10 predefined traffic destination combinations.

By splitting and distributing traffic across these configurations, our system reduces computation cost and average delay, while enhancing scalability and avoiding over-reliance on any single processing path.

C. Problem Formulation

In this study, we address two key challenges: minimizing computation costs in the management plane and minimizing delay in the control plane delay. We achieve this by optimizing traffic offloading ratios and cost allocation.

1) Cost Minimization in Management Plane: This work aims to minimize computation costs using traffic split offloading. Inputs include fog nodes, edge nodes, a cloud, traffic arrival rate, flow workloads and lengths, proportion of malicious traffic, link bandwidth, propagation delay, and cost per capacity unit for each tier. To achieve this, offloading ratios and costs must be properly allocated to each tier and task, while meeting delay threshold constraints. The problem statement is then formally defined as:

Input: A topology composed of N edge nodes, K fog nodes under an edge node, a flow arrival rate in each UE (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}), cost per unit capacity (S^C, S^E, S^F), and the delay threshold T.

Output: The allocated capacity to each tier in each task [C_i^F, C_i^E, C_i^C] based on the given cost and the offloading ratio O_j for each traffic path.

Objective: Minimize the total cost S, where S is equal to $\sum_{i=1}^{3}(C_i^F S^F + C_i^E S^E + C_i^C S^C)$.

Constraint: The average delay D must be less than the delay threshold T, and sum of offloading ratios must equal to 1, $\sum O_j = 1$.

2) Average Delay Minimization in Control Plane: In this second problem, the inputs are similar to the main problem, but the objective is to minimize the average delay. Proper allocation of offloading ratios and capacity for each tier and task, based on total computation cost, is required to achieve this. The problem statement 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 (λ), flow length of each task ($1/\mu_i^L$), flow workload of each task ($1/\mu_i^W$), the probability of malicious flows (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 in each task [C_i^F, C_i^E, C_i^C] based on the given cost, and the offloading ratio O_j for each traffic path.

Objective: Minimize the average delay D.

Constraint: The overall cost of each tier must equal the overall cost $\sum_{i=1}^{3}(C_i^F S^F + C_i^E S^E + C_i^C S^C) = S$, and the sum of offloading ratios must equal to 1, $\sum O_j = 1$.

D. Delay Model

In this work, we split traffic into multiple paths as detailed in Table III. To compute the delay, we calculate both communication and computation delays for each path using a queuing model. We then determine the average delay by weighting it according to the traffic volume on each path. Calculating the delay requires knowing the arrival rate and service capacity for each job (p, b, m) assigned to different nodes in the r-th source.

We construct four tables to obtain the necessary entities: Tables IV– VII represent task size, arrival rate, arrival rate definitions, and service capacity, respectively, for the r-th source. Values $r = 1$ to 7 represent communication resources (links between UE, fog, edge, and cloud), while $r = 8$ to 16 represent computation resources across the fog, edge, and cloud tiers. More specifically, $r = 8$ to 10 correspond to fog resources assigned to the three processing tasks: pre-processing, binary detection, and multi-class classification, respectively. Similarly, $r = 11$ to 13 represent edge resources for the same three tasks, and $r = 14$ to 16 represent cloud resources. This indexing structure helps organize the model's resource mapping across tiers and tasks in a unified way. Different traffic paths use combinations of these parameters to calculate delays, with ϕ indicating no traffic to specified resources.

TABLE III
POSSIBLE TRAFFIC PATH IN IDAS OFFLOADING

# 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

For example, in traffic path 1, where only fog processes the three tasks, traffic is sent from UE to fog with flow length $1/\mu_1^L$, arrival rate λ in $H[1,1]$, and capacity C^{UF} in $Y[1]$. The fog node then processes the tasks based on $V[1,4]$ flow workload, $H[1,4]$ arrival rate, and $Y[4]$ capacity. The delay for traffic path 1 is the sum of transmission time from UE to fog and processing time at the fog node.

The overall delay for each traffic path j is then calculated by adding all delay components using the queuing formula, which may be expressed as follows:

$$D_j = \sum_{r=1}^{16} b(j,r) \tag{1}$$

where

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

After obtaining the delay from all traffic paths, we can calculate the average delay by weighting it with the traffic volume on each path, which may be expressed as follows:

$$D = \frac{\sum_{j=1}^{10} D_j(\lambda_{O_j})}{\lambda} \tag{3}$$

Calculating the delay for computation nodes and communication links requires utilizing the (2) and then looking up the numbers in the Tables IV–VII. Moreover, the communication links consist of transmission delay and propagation delay. So, we also have to calculate the 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 $x = \phi$.

TABLE IV
TASK SIZE IN r-TH RESOURCE FOR THE j-TH TASK ASSIGNMENT, $V[j,r]$

$V[j,r]$	Communication							Computation								
	r=1	r=2	r=3	r=4	r=5	r=6	r=7	r=8	r=9	r=10	r=11	r=12	r=13	r=14	r=15	r=16
$V[1,r]$	$1/\mu_1^L$	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	$1/\mu_1^w$	$1/\mu_2^w$	$1/\mu_3^w$	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ
$V[2,r]$	$1/\mu_1^L$	$1/\mu_1^L$	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	$1/\mu_1^w$	$1/\mu_2^w$	$1/\mu_3^w$	ϕ	ϕ	ϕ
$V[3,r]$	$1/\mu_1^L$	$1/\mu_1^L$	ϕ	ϕ	$1/\mu_1^L$	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	$1/\mu_1^w$	$1/\mu_2^w$	$1/\mu_3^w$
$V[4,r]$	$1/\mu_1^L$	ϕ	ϕ	$1/\mu_3^L$	ϕ	ϕ	ϕ	$1/\mu_1^w$	$1/\mu_2^w$	ϕ	ϕ	ϕ	$1/\mu_3^w$	ϕ	ϕ	ϕ
$V[5,r]$	$1/\mu_1^L$	ϕ	ϕ	$1/\mu_3^L$	ϕ	ϕ	$1/\mu_3^L$	$1/\mu_1^w$	$1/\mu_2^w$	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	$1/\mu_3^w$
$V[6,r]$	$1/\mu_1^L$	$1/\mu_1^L$	ϕ	ϕ	ϕ	ϕ	$1/\mu_3^L$	ϕ	ϕ	ϕ	$1/\mu_1^w$	$1/\mu_2^w$	ϕ	ϕ	ϕ	$1/\mu_3^w$
$V[7,r]$	$1/\mu_1^L$	ϕ	$1/\mu_2^L$	ϕ	ϕ	ϕ	ϕ	$1/\mu_1^w$	ϕ	ϕ	ϕ	$1/\mu_2^w$	$1/\mu_3^w$	ϕ	ϕ	ϕ
$V[8,r]$	$1/\mu_1^L$	ϕ	$1/\mu_2^L$	ϕ	ϕ	$1/\mu_2^L$	ϕ	$1/\mu_1^w$	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	$1/\mu_2^w$	$1/\mu_3^w$
$V[9,r]$	$1/\mu_1^L$	$1/\mu_1^L$	ϕ	ϕ	ϕ	$1/\mu_2^L$	ϕ	ϕ	ϕ	ϕ	$1/\mu_1^w$	ϕ	ϕ	ϕ	$1/\mu_2^w$	$1/\mu_3^w$
$V[10,r]$	$1/\mu_1^L$	ϕ	$1/\mu_2^L$	ϕ	ϕ	ϕ	$1/\mu_3^L$	$1/\mu_1^w$	ϕ	ϕ	ϕ	$1/\mu_2^w$	ϕ	ϕ	ϕ	$1/\mu_3^w$

TABLE V
ARRIVAL RATE IN r-TH RESOURCE FOR THE j-TH TASK ASSIGNMENT, $H[j,r]$

$H[j,r]$	Communication							Computation								
	r=1	r=2	r=3	r=4	r=5	r=6	r=7	r=8	r=9	r=10	r=11	r=12	r=13	r=14	r=15	r=16
$H[1,r]$	λ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	λ_1^F	λ_2^F	$\lambda_3^F(P^A)$	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ
$H[2,r]$	λ	λ_1^{FE}	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	λ_1^F	λ_2^F	$\lambda_3^F(P^A)$	ϕ	ϕ	ϕ
$H[3,r]$	λ	λ_1^{FE}	ϕ	ϕ	λ_2^{EC}	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	λ_1^F	λ_2^F	$\lambda_3^F(P^A)$
$H[4,r]$	λ	ϕ	ϕ	$\lambda_3^{FE}(P^A)$	ϕ	ϕ	ϕ	λ_1^F	λ_2^F	ϕ	ϕ	ϕ	$\lambda_3^F(P^A)$	ϕ	ϕ	ϕ
$H[5,r]$	λ	ϕ	ϕ	$\lambda_3^{FE}(P^A)$	ϕ	ϕ	$\lambda_3^{EC}(P^A)$	λ_1^F	λ_2^F	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	$\lambda_3^F(P^A)$
$H[6,r]$	λ	λ_1^{FE}	ϕ	ϕ	ϕ	ϕ	$\lambda_3^{EC}(P^A)$	ϕ	ϕ	ϕ	λ_1^F	λ_2^F	ϕ	ϕ	ϕ	$\lambda_3^F(P^A)$
$H[7,r]$	λ	ϕ	λ_2^{FE}	ϕ	ϕ	ϕ	ϕ	λ_1^F	ϕ	ϕ	ϕ	λ_2^F	$\lambda_3^F(P^A)$	ϕ	ϕ	ϕ
$H[8,r]$	λ	ϕ	λ_2^{FE}	ϕ	ϕ	λ_2^{EC}	ϕ	λ_1^F	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	λ_2^F	$\lambda_3^F(P^A)$
$H[9,r]$	λ	λ_1^{FE}	ϕ	ϕ	ϕ	λ_2^{EC}	ϕ	ϕ	ϕ	ϕ	λ_1^F	ϕ	ϕ	ϕ	λ_2^F	$\lambda_3^F(P^A)$
$H[10,r]$	λ	ϕ	λ_2^{FE}	ϕ	ϕ	ϕ	$\lambda_3^{EC}(P^A)$	λ_1^F	ϕ	ϕ	ϕ	λ_2^F	ϕ	ϕ	ϕ	$\lambda_3^F(P^A)$

TABLE VI
ARRIVAL RATE DEFINITIONS

Arrival Rate	Definitions
λ_1^{FE}	$K(\lambda_{p2}+\lambda_{p3}+\lambda_{p6}+\lambda_{p9})$
λ_2^{FE}	$K(\lambda_{p7}+\lambda_{p8}+\lambda_{p10})$
λ_3^{FE}	$K(\lambda_{p1})$
λ_1^{EC}	$KN(\lambda_{p3})$
λ_2^{EC}	$KN(\lambda_{p8}+\lambda_{p9})$
λ_3^{EC}	$KN(\lambda_{p5}+\lambda_{p6}+\lambda_{p10})$
λ_1^F	$\lambda_{p1}+\lambda_{p4}+\lambda_{p5}+\lambda_{p7}+\lambda_{p8}+\lambda_{p10}$
λ_2^F	$\lambda_{p1}+\lambda_{p4}+\lambda_{p5}$
λ_3^F	λ_{p1}
λ_1^E	$\lambda_{p2}+\lambda_{p6}+\lambda_{p9}$
λ_2^E	$K(\lambda_{p2}+\lambda_{p6}+\lambda_{p7}+\lambda_{p10})$
λ_3^E	$K(\lambda_{p2}+\lambda_{p4}+\lambda_{p7})$
λ_1^C	$KN(\lambda_{p3})$
λ_2^C	$KN(\lambda_{p8}+\lambda_{p9})$
λ_3^C	$KN(\lambda_{p3}+\lambda_{p5}+\lambda_{p6}+\lambda_{p8}+\lambda_{p9}+\lambda_{p10})$

IV. SOLUTION APPROACH: SA-BASED TRAFFIC OFFLOADING AND COST ALLOCATION (SA-TOCA)

The SA-TOCA algorithm solves the problem by minimizing computation cost and average delay in the management and control planes. First, it defines the traffic offloading ratio for each path, allocates costs to each tier, and calculates capacity. Average latency is then computed using (3). The algorithm optimizes the offloading ratio and cost allocation based on the objective of minimizing computation costs while maintaining the average delay threshold in the management plane or minimizing the average delay in the control plane by adjusting specific lines in the algorithm.

Algorithm 1 outlines the details of the SA-TOCA. It employs a probabilistic approach to minimize computation costs by iteratively optimizing traffic offloading ratios and cost allocations. The algorithm initiates by setting parameters, including the offloading ratios and tier-specific costs, with initial temperatures t_1 and t_2 set to 1000, and a cooling rate $\alpha = 0.085$. The process starts by generating initial offloading solutions constrained by the condition that the sum of the offloading ratios equals one, as indicated in line 3 ($\sum_{j=1}^{10} O'_j = 1$).

During the iterative process, while the outer temperature t_1 exceeds the termination threshold t_{trm}, random offloading solutions are generated for O'_j. Within this loop, for each offloading solution, new cost allocations for each tier and task are generated by randomizing within specified minimum and maximum bounds for U_i^F, U_i^E, and U_i^C, as detailed in lines 6-8. For example, when generating a new solution for U_1^F, it first randomizes the cost for U_1^F, with a minimum number of $U_1^{F_{\min}}$ to prevent lack of capacity, which might result in a negative delay. $U_1^{F_{\min}}$ is calculated as $(H[j,8] \times V[j,8] \times S^F)$. The maximum number is used to narrow the search space, which can speed up the optimization time. Once we obtain U_i^F, U_i^E, and U_i^C, the next step is to determine the capacity for each tier and task (line 9). This can be achieved using the equation $Cx_i = (U_i^x \times sx)/(1/\mu_i^W)$, where x can be either F, E, or C.

Following this, the algorithm computes the average delay D and the total computation cost S for the new solutions (line

TABLE VII
COMPUTING CAPACITY IN THE r-TH RESOURCE (QUEUE), $Y[r]$

r=1	r=2	r=3	r=4	r=5	r=6	r=7	r=8	r=9	r=10	r=11	r=12	r=13	r=14	r=15	r=16
C^{UF}	C_1^{FE}	C_2^{FE}	C_3^{FE}	C_1^{EC}	C_2^{EC}	C_3^{EC}	C_1^{F}	C_2^{F}	C_3^{F}	C_1^{E}	C_2^{E}	C_3^{E}	C_1^{C}	C_2^{C}	C_3^{C}

10). The change in computation cost ($\Delta S = S' - S$) is then evaluated (line 11). If the new solution improves the system by reducing S and maintaining D below a delay threshold T, it is accepted (lines 12-13). Otherwise, it may still be accepted with a probability dependent on the temperature and cost change ($\exp(\Delta S/t)$), which helps in avoiding local minima (lines 14-15).

The inner loop decreases the temperature t_2 iteratively (line 16), repeating the process until t_2 falls below t_{trm}. After optimizing the cost allocations for new O_j, the algorithm performs a final calculation of D and S', comparing them with previous values (lines 19-20). If the new S is lower and D is within the acceptable range, the new ratios replace the current ones (lines 21-22). This iterative refinement continues until the outer temperature t_1 also falls below t_{trm}, resulting in optimized traffic offloading ratios and cost allocations with minimized computation costs.

Moreover, to achieve the goal of minimizing the average delay in the control plane optimization process, we reuse Algorithm 1 with some modifications. The first change involves the maximum value for new cost allocation randomization at lines 6-8, which are defined as follows:

$$\begin{cases} U_i^{F_{\max}} = S - \sum_{i=1}^{3} U_i^{F_{\min}} - \sum_{i=1}^{3} U_i^{E_{\min}} - \sum_{i=1}^{3} U_i^{C_{\min}} \\ U_i^{E_{\max}} = S - \sum_{i=1}^{3} U_i^{F} - \sum_{i=1}^{3} U_i^{E_{\min}} - \sum_{i=1}^{3} U_i^{C_{\min}} \\ U_i^{C_{\max}} = S - \sum_{i=1}^{3} U_i^{F} - \sum_{i=1}^{3} U_i^{E} - \sum_{i=1}^{3} U_i^{C_{\min}} \end{cases}$$

The second modification involves adjusting the criteria for accepting new solutions at lines 12 and 21. Instead of comparing the new S value, we now compare the new average delay D and accept the new solutions if the D value is lower than the previous one.

Algorithm 1: SA-TOCA.

initialize $U_i^F, U_i^E, U_i^C, O_j, t_{\text{ini}} = 1000, \alpha = 0.085$;
$t_1, t_2 = t_{\text{ini}}$; //
1 calculate D, C_i^F, C_i^E, C_i^C, S;
2 **while** $t_1 > t_{trm}$ **do**
3 generate random offloading solutions for O'_j with constraint $\sum_{j=1}^{10} O'_j = 1$;
4 **while** $t_2 > t_{trm}$ **do**
5 generate new solutions for U_i^F, U_i^E, U_i^C, where $i \in 1, 2, 3$ by;
6 $U_i^{F'} = rand(U_i^{F\min}, U_i^{F\max})$;
7 $U_i^{E'} = rand(U_i^{E\min}, U_i^{E\max})$;
8 $U_i^{C'} = rand(U_i^{C\min}, U_i^{C\max})$;
9 calculate C_i^F, C_i^E, C_i^C based on $U_i^{F'}, U_i^{E'}, U_i^{C'}$;
10 calculate D and S';
11 $\Delta S = S' - S$;
12 **if** $S' \leq S \wedge D \leq T$ **then**
13 accept new $U_i^F = U_i^{F'}, U_i^E = U_i^{E'}, U_i^C = U_i^{C'}$, $S = S'$;
14 **else**
15 accept new $U_i^F = U_i^{F'}, U_i^E = U_i^{E'}, U_i^C = U_i^{C'}$, $S = S'$, with probability $\frac{1}{exp(\Delta S/t)}$;
16 decrease the temperature: $t_2 = \alpha \times t_2$;
17 **end**
18 **end**
19 calculate D and S';
20 $\Delta S = S' - S$;
21 **if** $S' \leq S \wedge D \leq T$ **then**
22 accept new O'_j, $S = S'$;
23 **else**
24 accept new $O_{j'}$, $S = S'$, with probability $\frac{1}{exp(\Delta S/t)}$;
25 **end**
26 decrease the temperature: $t_1 = \alpha \times t_1$;
27 **end**

V. SIMULATION AND RESULTS

This section presents the numerical results obtained from the simulation tests that were carried out in order to evaluate the performance of our IDaS architecture with traffic split offloading. Each experiment was repeated ten times to ensure the reliability and robustness of the results. This repetition is crucial as the simulated annealing algorithm employed in our study uses a probabilistic approach to generate solutions, aiming to achieve a near-global optimum. The inherent randomness in solution generation means that each run can yield slightly different outcomes. To confirm that our results are consistently close to the optimal solution, we computed the standard deviation of the results across repetitions. These deviations are represented as error bars in the figures, illustrating the variation in each experiment's results.

A. Measurements on Binary and Multi-Class Algorithm

In this study, we utilize a multi-stage ML-based IDS pipeline with binary detection and multi-class classification, which has demonstrated a better accuracy compared to baseline models and existing research [4]. This foundation allows our work to focus primarily on the architecture selection for running an ML-based IDS pipeline rather than on model development. However, we also conducted a comprehensive ML-based IDS training process. This is necessary to measure the workload of these models during their detection tasks.

The workload parameters in this study are based on a real system, bringing the model closer to real-life behavior. We built an ML-based IDS pipeline with three functions: preprocessing, binary detection, and multi-class classification. We executed the IDS tasks on an Intel Xeon E5 processor, using PERF tools to count CPU instructions.

Preprocessing involves extracting features, selecting them, and parsing raw data. We used CIC-FlowMeter [20] to extract features, transforming raw data into a table with 80 features. Using CIC-IDS-2017 as the training dataset, we performed feature selection based on [21] by removing certain features. Finally, we normalized the data using the MinMax Scaler from

TABLE VIII
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
Multi-class	Decision tree	114,849
	Random forest	167,493
	Gradient boosting	197,266
	MLP	312,614
	DNN	915,083

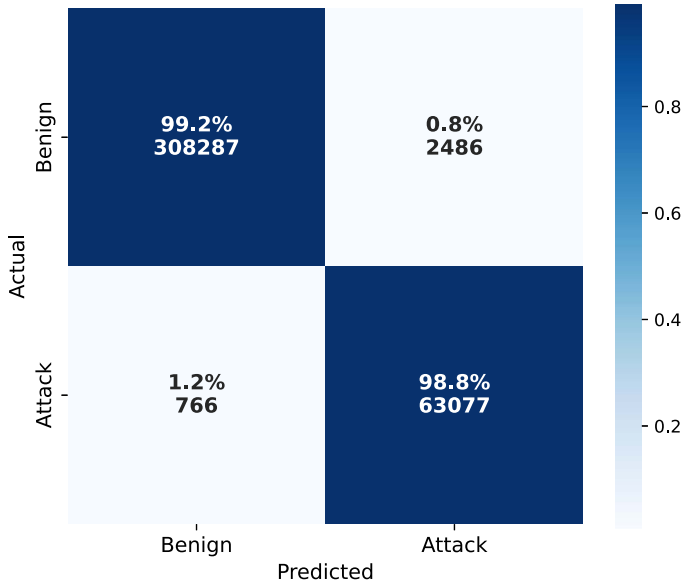
Scikit-Learn to maintain good detection performance, as recommended by [22].

In binary detection, we have to detect the attack quickly with the low computation resources needed. To fulfill this requirement, we utilized a decision tree (DT) since it is fast and accurate enough [22]. We constructed this decision tree with the hyperparameters derived from [22]. Furthermore, in multi-class classification, we utilized DL since it is better to distinguish the normal and malicious traffic [23]. Specifically, we implemented a 5-layer deep neural network (DNN) based on the model proposed by [23]. Since we utilized a similar dataset to train the DNN model, we were able to leverage the existing architecture and hyperparameters.

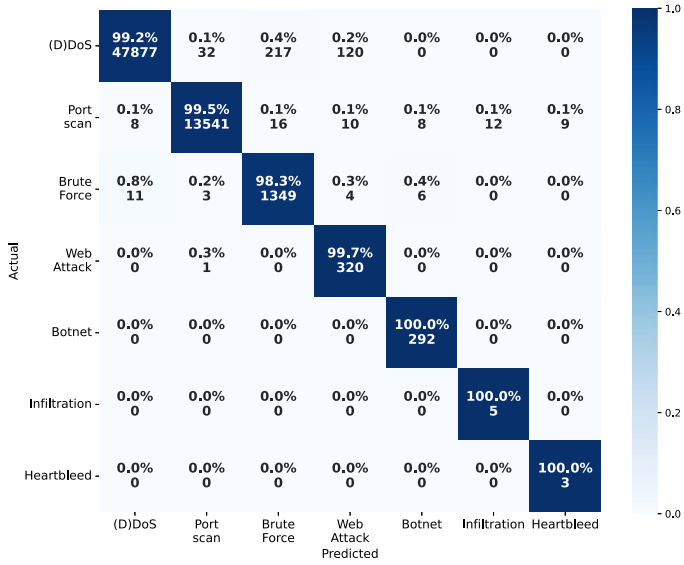
For the further experiment, we also used various ML and DL models, including random forest and gradient boosting, using the hyperparameters derived from [22], and a single-layer multilayer perceptron (MLP), as proposed by [23].

All the machine learning models were trained on the CIC-IDS-2017 dataset and achieved an average F1 score of 99%, with 99.50% precision, 98.50% recall, and 98.75% accuracy in testing dataset. Fig. 3 presents the confusion matrices for both binary detection and multi-class classification, using the DT and DNN models as the example, respectively.

Next, we measured the workload of those ML and DL models using PERF and used it to perform the analytical of the effect of flow workload. Table VIII lists the results of measurements. Finally, we also measured the flow length by monitoring the changes in file size in each stage. We observed that after pre-processing, flow length decreased from 100% to 5.3%.



(a) Binary Detection



(b) Multi-class Classification

Fig. 3. The confusion matrices of the detection models.

B. Parameter Settings

Table IX lists the simulation parameters. The parameters were chosen based on references from previous works and our measurements of the real ML-based IDS system. Such settings bring the model's behavior closer to real-life scenarios. The architecture was configured with a cloud node, 400 edge nodes, and 20 fog nodes under each edge node. We also considered 5G bandwidth for each link. The pre-processing flow length is derived from [24], which provides real experimental data on benign traffic flow size. From their statistics (3,876,982 flows and 65.6 GB of raw data), we calculated an average of approximately 135 kilobits per benign flow. To obtain a more representative value, we conducted our own experiment to measure attack traffic, which averaged around 114.98 kilobits per flow. By combining both benign and attack traffic, we derived a final average flow length of 124.99 kilobits, ensuring that our pre-processing parameters are grounded in both literature and experimental data.

Additionally, the distance between each tier is derived from [25]. These distances were used to calculate the propagation delay by dividing the distance by the speed of light in the given medium. We considered using the fiber as the medium where the speed of light in fiber is approximately 2×10^8 meters per second. This calculation ensures that our delay settings accurately reflect real-world conditions.

The parameters in Algorithm 1 were selected based on empirical testing based on random search. For instance, the initial

TABLE IX
PARAMETER SETTINGS

Notations	Value	Unit
N	400	Nodes
K	20	Nodes
$1/\mu_1^W$	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
C^{UF}	1	Gbps
C^{FE}	100	Gbps
C^{EC}	100	Gbps
S	11×10^{13}	Money units
S^C	10	Money units/instruction
S^E	15	Money units/instruction
S^F	20	Money units/instruction
λ	500	Flows/second
Z^{UF}	3.335×10^{-6}	Seconds. Distance: 1 km
Z^{FE}	3.335×10^{-5}	Seconds. Distance: 10 km
Z^{EC}	3.335×10^{-3}	Seconds. Distance: 1000 km
P^A	0.2	

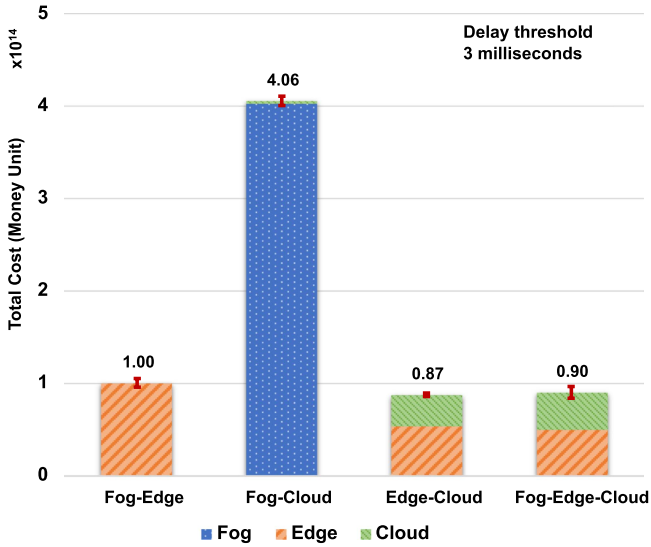


Fig. 4. Architecture: two- vs. three-tier in minimum cost.

temperature t_{ini} and the cooling rate α were chosen to balance the exploration and exploitation phases of the simulated annealing process. These settings were fine-tuned through extensive experimentation to achieve a good trade-off between convergence speed and solution quality. After exhaustive experiments, we found that setting $t_{\text{ini}} = 1000$ and $\alpha = 0.085$ provided the best parameter configurations for Algorithm 1.

Furthermore, we decide to set the computation cost at the lower tiers more expensive than higher tiers. This is due to the fact that the lowest tier has a higher operational cost and maintenance due to the effect of distributed data centers in comparison to centralized data centers in cloud computing [26]. The costs S^C, S^E, and S^F are then set to [10, 15, 20] in money units. All the code to run the optimization can be retrieved from the GitHub repository [27].

C. Results for Management Plane

In this section, we present and discuss the results on the management plane. We organize our discussion based on the issues that we investigate in this management plane: (1) two- vs. three-tier architecture, (2) traffic offloading ratios in each path, (3) cost allocation in each tier, and (4) effect of delay thresholds on the computation cost.

1) Architecture: Two- Vs. Three-Tier: Fig. 4 shows the comparative results of two- and three-tier architectures with the total allocated cost with the delay threshold is 3 milliseconds. According to the results, edge-cloud and fog-edge-cloud architectures are 15% and 4 times more cost-effective than fog-edge and fog-cloud architectures, respectively.

In this scenario, the fog-cloud combination is not ideal. Despite the cloud's superior processing capabilities, it suffers from high propagation delay. As a result, the algorithm directed all traffic to be processed in the fog, leading to a higher total cost. Similarly, in the fog-edge architecture, although forwarding all traffic to the edge takes advantage of the shorter distance and effectively meets the delay constraints, it incurs higher costs. However, by offloading some of the IDS traffic from the edge to the cloud, the allocated cost can be significantly reduced while still meeting the delay threshold. It can happen because cloud resources are usually more cost-effective than edge resources. Cloud providers can offer large-scale resources at a lower cost due to economies of scale, while edge devices have to scatter the resources to the distributed edge nodes, which can be more expensive. By offloading some of the IDS traffic to the cloud, edge devices can reduce their workload, thereby increasing their processing efficiency.

Additionally, implementing traffic split offloading combined with capacity slicing in the IDaS architecture greatly improves the efficiency of managing computation costs. This method offers a clear improvement over the closest competitor, [19], which does not incorporate these techniques into its IDaS model. By applying the same parameters—such as traffic arrival rates, the number of fog and edge nodes, and the delay threshold—our approach achieves a four times lower computation cost while meeting the required delay thresholds for processing traffic. Fig. 5 compares the computation cost allocation between our approach and [19], specifically for the edge-cloud and fog-edge-cloud architectures. In contrast, the fog-edge and fog-cloud architectures in [19] yielded computation costs that were more than six times higher than our approach, making them less suitable for meaningful comparison. The results clearly demonstrate that our proposed method achieves consistently lower computation costs in both architectures, emphasizing the value of traffic split offloading and capacity slicing in optimizing resource allocation and ensuring cost-effectiveness.

We conclude that using edge-cloud is the best option in considering the lowest computation cost scenario for running IDaS service.

2) Traffic Offloading Ratios: In this section, we present a comprehensive analysis of offloading ratios in various architectures. Our findings indicate that offloading traffic to a higher

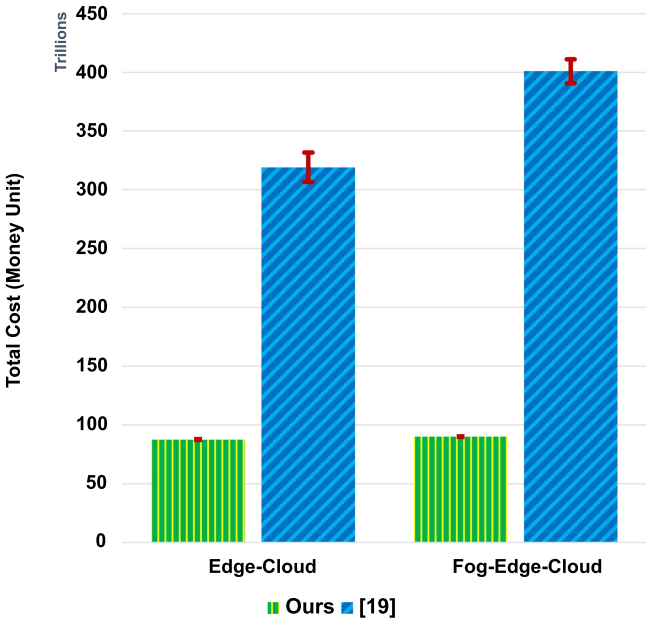


Fig. 5. Average cost comparisons: ours vs. [19].

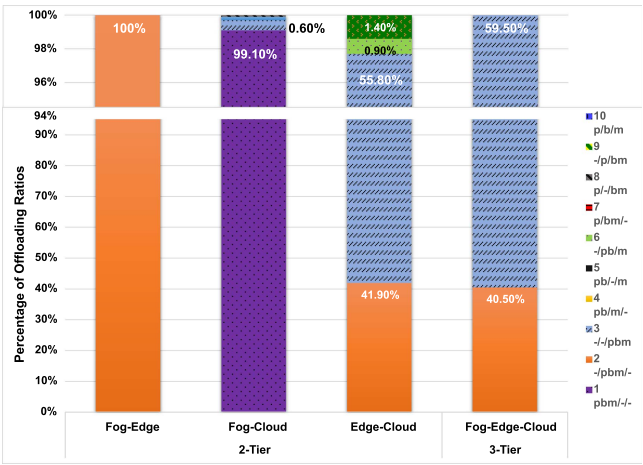


Fig. 6. Traffic offloading ratios in minimum cost.

tier in edge-cloud, fog-edge-cloud, and fog-edge can effectively decrease computation costs. As illustrated in Fig. 6, edge-cloud and fog-edge-cloud architectures offloaded a significant proportion of their traffic, 55.8%, and 59.5%, respectively, to the cloud as their highest tier. This offloading strategy demonstrates the importance of leveraging higher-tier resources to optimize processing capabilities.

In contrast to the previously discussed architectures, fog-cloud architecture processes 99% of its traffic in the fog rather than offloading it to the cloud as its highest tier. This counter intuitive strategy stems from the longer transmission and propagation delays encountered when transferring traffic between the fog and cloud. As a result, the algorithm must allocate more cost to the fog to accommodate the delay threshold, which is worsened by the longer transmission delay.

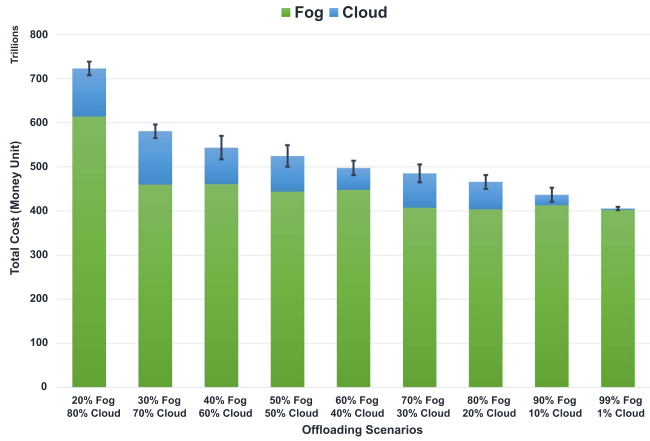


Fig. 7. Computation cost in different offloading scenarios of fog-cloud.

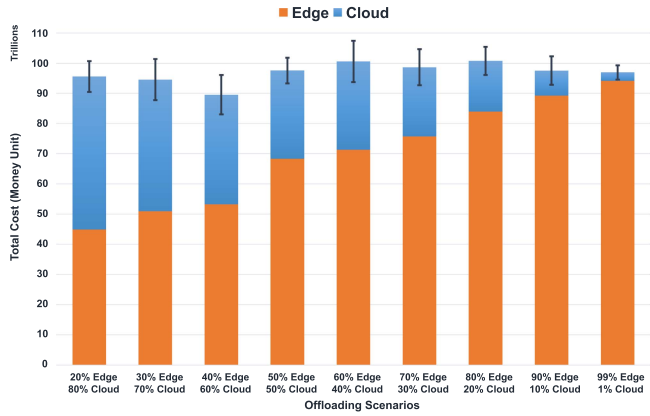


Fig. 8. Correlation between total computation cost and traffic offloading ratios in the edge-cloud architecture.

Consequently, forcing the computation delay lower in fog significantly increases the cost. This phenomenon is illustrated in Fig. 7, where we analyzed the impact of offloading ratios on computation costs by manually adjusting the traffic distribution between the cloud and fog. Our findings reveal that the more traffic offloaded to the cloud, the more cost is allocated to the fog in order to minimize the average delay by accelerating processing time within the fog.

For example, when comparing the performance of offloading 80% of the traffic to the cloud with processing 99% of the traffic in the fog, we found that the former scenario incurs a 78.2% higher computation cost. In this case, the average delay of processing time in the fog is reduced to 1.49 ms, whereas when 99% of the traffic is processed in the fog, the average delay of processing time in the fog increases to 2.99 ms, further emphasizing our previous statement.

Furthermore, selecting the appropriate traffic offloading ratio is essential for achieving cost efficiency and maintaining performance in the IDaS architecture. Fig. 8 illustrates the correlation between computation cost and offloading ratios in the edge-cloud architecture, highlighting how traffic distribution significantly impacts overall costs. The figure shows that

offloading more traffic to edge nodes leads to higher costs due to the distributed nature and operational expenses of edge resources. By contrast, optimizing the offloading ratio can reduce computation cost by around 10%, with 40% and 60% of the traffic being offloaded to the edge and cloud, respectively, proving to be the most cost-effective balance. This underscores the importance of optimization in determining the best offloading strategy to efficiently utilize resources while meeting delay constraints.

Our comprehensive analysis of offloading ratios in various architectures reveals that strategically offloading traffic to higher tiers effectively decreases computation costs

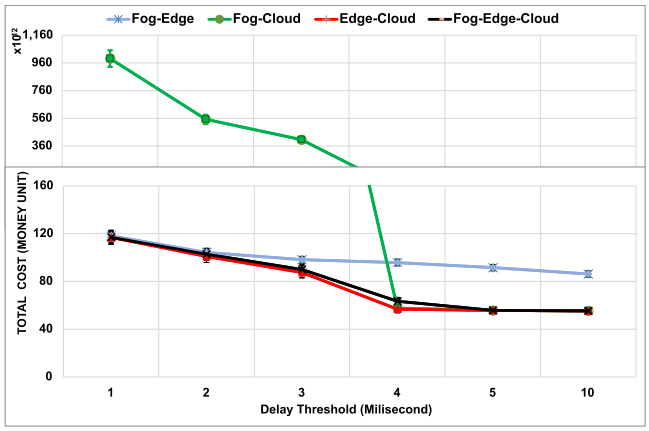
3) Cost Allocation: In this section, we analyze cost allocation across various tiers in each architecture and optimize the allocation based on traffic offloading solutions. For the fog-edge and fog-cloud architectures, more than 99% of the traffic is offloaded exclusively to edge and fog tiers, respectively, resulting in cost allocation primarily to those tiers. Conversely, in edge-cloud and fog-edge-cloud architectures, even though over 55% of the traffic is offloaded to the cloud, a larger proportion of the cost must be allocated to the edge tier. As shown in Fig. 8, 40% of the cost is allocated to the cloud, while 60% is allocated to the edge.

We determined this allocation through extensive simulations using the SA-TOCA algorithm, which employs a probabilistic approach to explore various offloading and cost allocation scenarios. Over several iterations, the algorithm converged to the 40%-60% split between the cloud and edge, yielding the lowest overall cost while maintaining the required delay threshold.

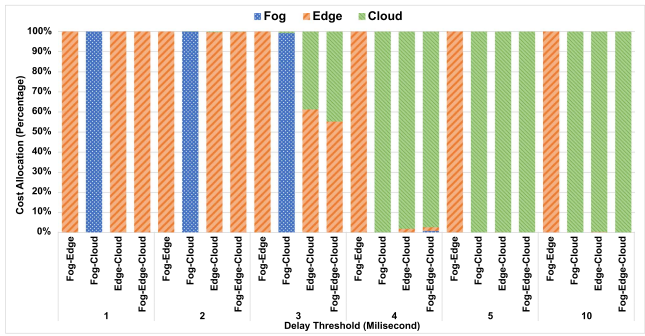
This allocation reflects a trade-off between capacity and proximity. Cloud resources provide cost-efficient, large-scale capacity, but their higher propagation delays make them less suitable for handling latency-sensitive traffic. Edge resources, on the other hand, offer lower propagation delays due to their proximity to users, but incur higher operational costs because of their distributed nature. The SA-TOCA algorithm balances these factors by probabilistically adjusting the offloading ratios and the cost allocation to each tier. This configuration optimally reduces the edge's workload by offloading part of the traffic to the cloud, thereby minimizing the total computation cost without exceeding the delay constraint. This result underscores the importance of carefully considering the costs associated with managing traffic in different tiers.

Based on our analysis, we have found that allocating 40% of the cost to the cloud and 60% to the edge is the optimal solution to achieve the minimum computation cost while still satisfying the delay constraint.

4) Effect of Delay Threshold: The delay threshold plays a crucial role in the performance of the IDaS system, as shown in Fig. 9(a). To provide a clearer view of the cost trends, we split the figure into two parts. The first part presents the fog-edge and fog-edge-cloud architectures, where the cost trends are smoother and more comparable. The second part focuses on the fog-cloud architecture, which exhibits a sharp decrease in cost under tight and loose delay constraints. This separation allows for better readability and ensures that the distinct behavior of each



(a) Total cost



(b) Allocated cost

Fig. 9. Effect of delay threshold.

architecture is clearly illustrated without one overshadowing the others.

When the delay threshold is tight (e.g., 3 ms), the fog-cloud architecture can be up to seven times more expensive than other architectures due to the high propagation delay between the fog and cloud tiers. This delay makes it difficult to meet strict delay constraints if traffic is offloaded to the cloud. As a result, the algorithm prioritizes processing all traffic in the fog to reduce delay, requiring full cost allocation to the fog tier. However, processing traffic solely in the fog increases costs significantly because fog nodes are more distributed and require higher operational expenses than centralized cloud nodes. Although offloading computation to the cloud can reduce total capacity requirements, the long latency between tiers in the fog-cloud architecture forces the system to rely heavily on the fog tier under tight constraints, resulting in a sharp increase in computation cost.

Using the edge as the offloading destination is more appropriate when the delay threshold is tight because it can satisfy the delay threshold while still minimizing the cost. This is because the edge is a suitable location where the distance is still close to UE and has a more centralized capacity than the fog. As a result, all architectures that utilize the edge allocate all traffic to be processed there, as shown in Fig. 9(b).

On the other hand, when the delay threshold is loose, it is preferable to serve traffic using the highest tier instead of

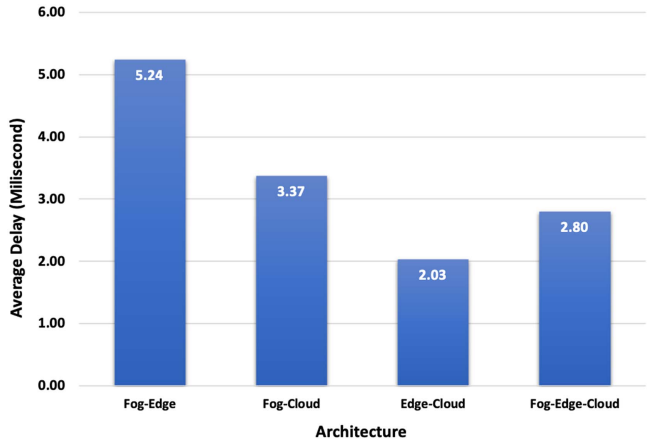


Fig. 10. Architecture: two- vs. three-tier in minimum delay.

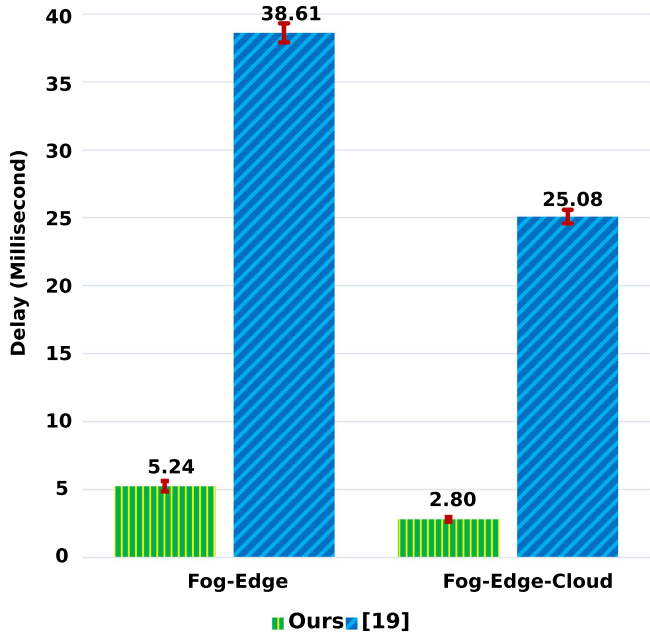


Fig. 11. Average delay comparisons: ours vs. [19].

the lower tier to reduce capacity. Fig. 9(b) demonstrates that offloading IDS traffic to the cloud in edge-cloud architecture can lower the cost by 55.6% compared to offloading the traffic to the edge in fog-edge architecture. This is because the edge-cloud architecture provides a more efficient way of utilizing the resources. Thus, when the delay threshold is loose, it is more efficient to offload traffic to the cloud than to process it at the edge.

We conclude that a combination of edge and cloud architecture offers flexibility in minimizing computation costs.

D. Results for Control Plane

In this section, we present and discuss the results on the control plane. We organize our discussion based on the issues that we investigate in this control plane: (1) two- vs. three-tier architecture, (2) cost allocation in each tier, (3) traffic offloading ratios in each path, (4) effect of malicious traffic, (5) effect of flow workload, and (6) effect of computation cost. In running the experiment for these results, we kept the same computation cost for all architectures.

1) Architecture: Two- Vs. Three-Tier: Fig. 10 illustrates a comparison of average delays for processing a flow in two- and three-tier architectures. Our analysis demonstrates that employing an edge-cloud federation for offloading IDS traffic yields the shortest average delay. This allocation strategy proves ideal in terms of distance, coverage, and cost, resulting in the minimum delay.

When considering alternative architecture within a two-tier architecture, such as fog-cloud and fog-edge, it was observed that the average delay increased by 66% and more than two times, respectively. Although fog-edge excels in distance, it falls short in coverage and cost allocation. The fog layer offers the closest proximity to UE, but its highly distributed nature leads to the dispersion of resources across various areas, causing ineffective cost allocation and increased delays.

Cloud computing, on the other hand, offers the lowest cost utilization due to its more centralized traffic handling. However, it has the longest distance to the UE, which contributes to additional propagation delays. Introducing an edge tier in a three-tier architecture improves the delay by 20.36% compared to fog-cloud. However, as this architecture still necessitates cost allocation to the fog layer, its efficiency is 27.5% lower than that of the edge-cloud configuration.

Furthermore, employing traffic split offloading and allocating sliced capacity not only proves effective in managing computation costs but also enhances performance in managing traffic to deliver services with minimum processing delay. This approach outperforms the closest competitor, [19], which configured its IDaS model without implementing traffic split offloading or capacity slicing. When using the same parameters—such as arrival traffic rate, the number of fog and edge nodes, and total computation cost—its delay increased by more than seven times compared to ours. Fig. 11 illustrates a comparison of the average delay in processing traffic between our approach and [19], focusing on both the fog-edge and fog-edge-cloud architectures. The results clearly show that their system becomes unable to handle traffic effectively, as evidenced by the significant increase in average delay.

We conclude that leveraging edge-cloud as the architecture for IDaS offers the most effective approach for handling IDS traffic.

2) Traffic Offloading Ratios: This section provides the analysis of the traffic offloading ratios for each architecture. In performing these experiments, we utilized the same computation cost for all architectures. Similar to the management plane problem, our analysis of the control plane problem reveals that, despite having 10 traffic paths available for offloading, all architectures utilize no more than six traffic paths, as shown in Fig. 12. Notably, 90% of the traffic is directed to traffic paths number 2

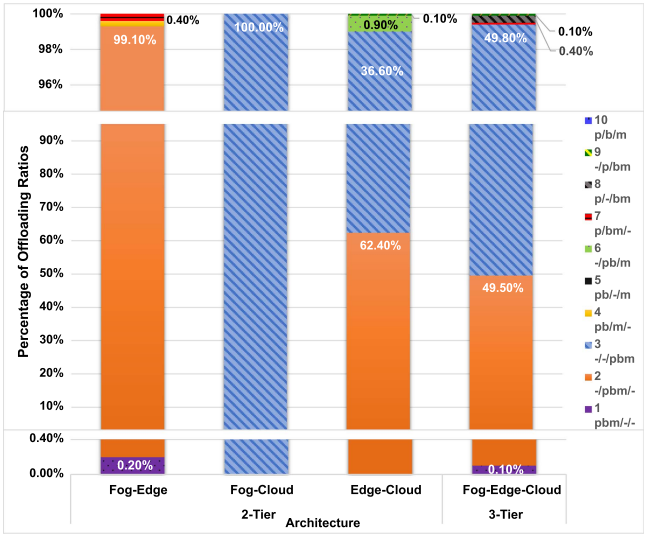


Fig. 12. Traffic offloading ratios in minimum delay.

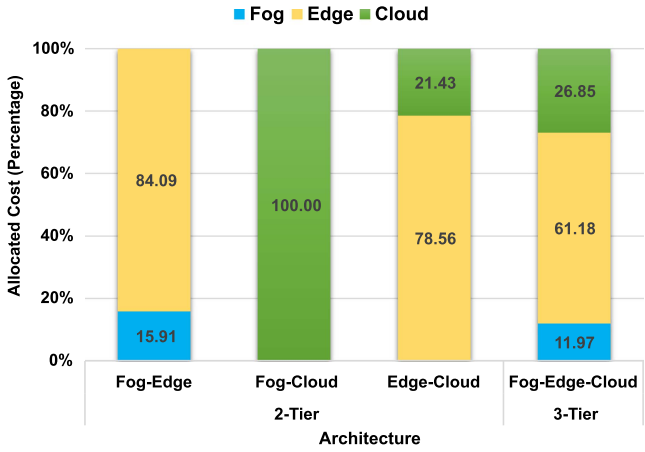


Fig. 13. Cost allocation in minimum delay.

and 3, which process all tasks—including pre-processing, binary detection, and multi-class classification—in the edge or the cloud, respectively. This observation demonstrates that joining tasks within a single processing node yields better results than separating tasks across several nodes. The rationale behind this phenomenon is that in a separated configuration, traffic must be transmitted to subsequent tiers for further processing, thereby increasing the total delay due to transmission delay. Consequently, the SA-TOCA algorithm favors the joint configuration and offloads more traffic onto it to minimize delays.

Furthermore, an examination of offloading ratios in each architecture, as illustrated in Fig. 12, reveals that the SA-TOCA algorithm tends to avoid processing traffic in the fog tier. In both fog-edge and fog-cloud architectures, the majority of the traffic is processed at their highest tiers. This can be attributed to the high computation cost associated with utilizing fog. Therefore, offloading traffic to a higher tier proves more effective in reducing the average delay.

In conclusion, our findings suggest that joining all tasks to be processed in a single node is a more efficient approach than separating them across multiple nodes as it reduces delays and conserves bandwidth.

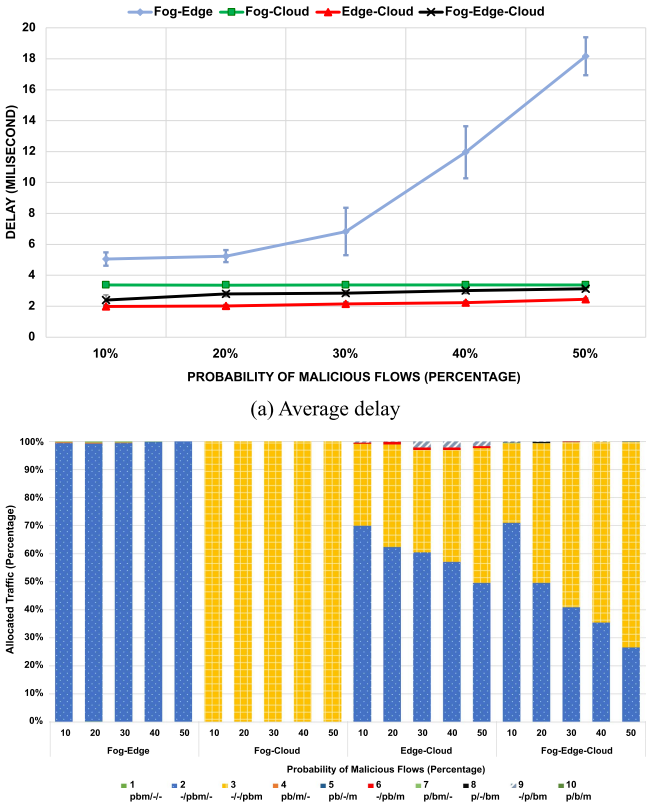
3) Cost Allocation: Fig. 13 illustrates the allocation of the cost to the architecture. Edge computing has emerged as the most utilized tier in various architectures, with more than 61% of costs being allocated to the edge. Although the edge is efficient, it must be appropriately combined with other tiers to offload processing tasks effectively. However, using only the edge without appropriate offloading to other tiers can result in a high computation delay, especially when the computation cost is tight. Allocating 100% cost and forwarding 100% traffic to the edge, in this case, can push a higher resource utilization, resulting in a computation delay that is 10 times higher. This observation differs from that of [19], where they concluded that using only the edge was the optimal approach. However, our findings indicate that offloading a portion of the traffic to higher tiers and allocating the appropriate computation cost, such as in edge-cloud and fog-edge-cloud architectures where 26% of the cost is allocated to the cloud, enhances the processing delay performance.

Over-allocating costs to the fog tier, on the other hand, is not an ideal option. Due to its highly distributed nature, the fog tier requires the cost to be scattered across more than 8,000 nodes, necessitating substantial resources. According to queueing theory, a large centralized capacity offers better utilization than a small distributed capacity [28]. For example, the results regarding the utilization of fog computing in the fog-edge and fog-edge-cloud architectures. Although no more than 1% of traffic is directed to these paths, 15% of the cost must be allocated to fog nodes. While the benefits of fog computing, such as improved latency, reduced bandwidth usage, and better privacy and security, are undeniable, the costs associated with this technology should be carefully considered by service providers before utilizing it.

In conclusion, to achieve optimal performance, it is important to allocate costs appropriately between the edge and cloud tiers while also avoiding the over-allocation of capacity to the fog tier.

4) Effect of Malicious Traffic: In this section, we analyze how the performance of architectures is impacted as the proportion of malicious traffic is increased from 10% to 50%. As illustrated in Fig. 14(a), a rise in the probability of malicious traffic has a direct impact on architectures without cloud federation, such as fog-edge, causing the average delay to rise up to three times. In contrast, architectures with cloud federation demonstrate stable performance in handling increased traffic requests. Offloading computations to the highest tier can effectively maintain performance. As demonstrated in Fig. 14(b), when the probability of malicious flows is low, edge-cloud and fog-edge-cloud architectures direct approximately 70% of the traffic to the edge. Moreover, as traffic requests increase, around 50% of the traffic is processed in the cloud. Although the cloud is more distant, offloading computations to the cloud in cases of increased traffic helps maintain performance.

In conclusion, utilizing a cloud proves beneficial as an offloading destination when IDS traffic requests increase.



(a) Average delay

(b) Traffic ratios

Fig. 14. Effect of malicious traffic.

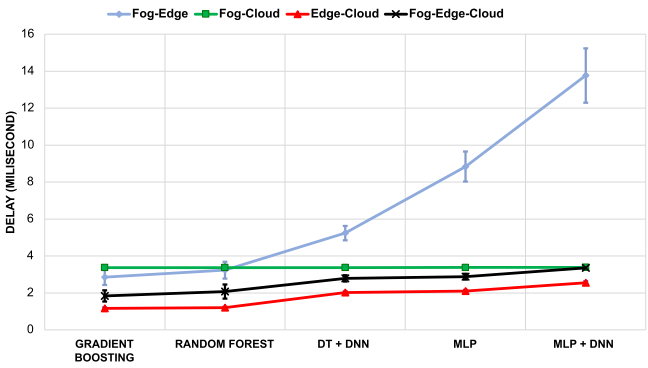
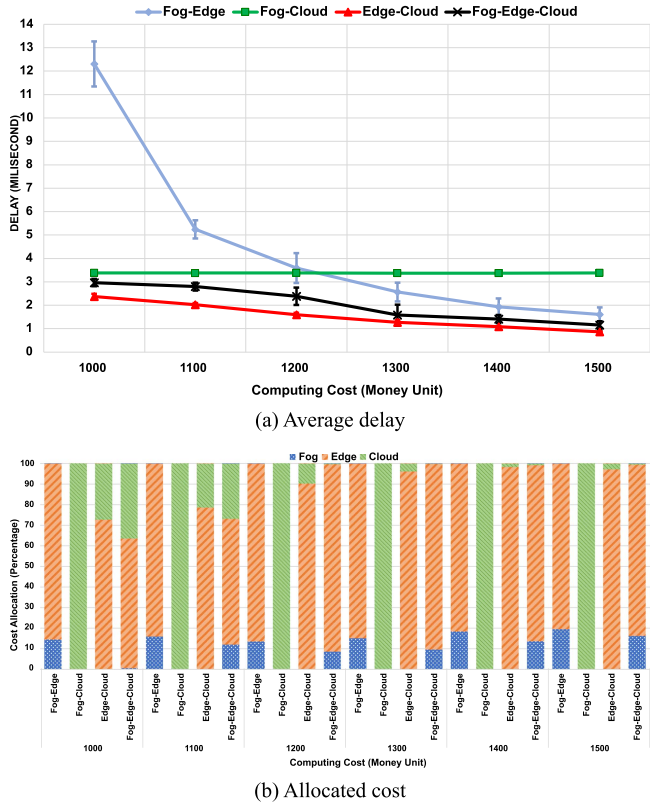


Fig. 15. Effect of flow workload.



(a) Average delay

(b) Allocated cost

Fig. 16. Effect of computation cost.

5) Effect of Flow Workload: The objective of this investigation was to explore the impact of flow workload on average delay using different ML algorithms for binary and multi-class detection. Fig. 15 shows that changing the ML model does not significantly affect the fog-cloud, edge-cloud, and fog-edge-cloud architectures. Employing low-computation ML models like gradient boosting or random forest results in the optimization algorithm allocating more capacity to the edge tier, reducing delay. Conversely, using high-computation ML models like MLP for binary and multi-class detection and combining MLP with DNN shifts more cost to the cloud to maintain low-delay performance. Offloading to the cloud enhances scalability and performance, particularly under high workload conditions, helping to prevent the overloading of lower-tier nodes. For low-workload computations, fog, and edge tiers are adequate; for higher workloads, offloading to the cloud is preferable.

We conclude that when using computationally intensive ML models, offloading computation to cloud computing is preferable to maintain performance.

6) Effect of Computation Cost: The investigation analyzed the impact of computation cost on the architecture's performance by varying the total cost and observing its effects on performance. The effect of computation cost has a significant impact on the fog-edge architecture, causing a substantial increase in delay while being relatively stable for other architectures, as shown in Fig. 16(a). The cause for this can be attributed to the fact that as computation costs decrease, resource utilization increases. According to queuing theory, increased resource utilization prolongs the time spent in queues [29]. Consequently, in a distributed architecture with limited capacity, such as fog-edge, the queuing time increases substantially. In a low-cost scenario, offloading traffic to the cloud and allocating 40% cost to it can help reduce the delay by 75%, as shown in Fig. 16(b). However, by increasing the computation cost from 1000 to 1300, processing more traffic at the edge results in better delay than offloading it to the cloud.

In conclusion, edge-cloud and fog-edge-cloud architectures are the most effective choices for IDaS due to their stable performance in both low and high-cost scenarios.

VI. Conclusions and Future Works

This study evaluates the effectiveness of IDaS within a multi-tier architecture. We map tasks to the architecture, enabling offloading based on this mapping and model the architecture. An optimization process calculates offloading ratios and cost requirements for each traffic path, tier, and task. By evaluating the architecture and the impact of various parameters on performance (minimizing cost and delay), we provide recommendations for service providers.

The results of the study indicate that in terms of architectures with the lowest computational cost, the edge-cloud architectures provide 15% and 4 times more cost-effective than fog-edge and fog-cloud architectures, respectively. Especially when the delay threshold is loose. This can be achieved by allocating 40% of the cost to the cloud and 60% to the edge.

Similar to the previous objective, when service providers offer the IDaS with minimum processing time, using edge-cloud with 78.5% cost allocated to edge and 21.43% to the cloud offers the most effective architecture with 66% and 100% better average delay than fog-cloud and fog-edge, respectively. Furthermore, in defining the traffic offloading destination, joining all tasks to be processed in a single node is a more efficient approach than separating them across multiple nodes. Then, in the case of the increased number of traffic requests and workload, offloading the traffic to the cloud has proven to maintain the delay performance still low.

While some results, such as the reduction in computation costs with increased cloud processing, may appear intuitive, they are crucial for validating the practical effectiveness of our proposed multi-tier architecture. These findings provide empirical support for the utility and novelty of our approach, demonstrating its capability to balance cost and performance in real-world ML-based IDS deployments. The detailed analysis of offloading ratios further illustrates the significant performance gains achievable through our optimized traffic split offloading strategy.

Work that needs to be considered in the future includes the following: First, developing more sophisticated optimization algorithms, such as Reinforcement Learning, that take into account dynamic changes in network conditions and traffic patterns with faster decision time. Moreover, evaluating the performance of IDaS in real-world scenarios with varying levels of network traffic and attack types also could be explored.

References

[1] A. Halbouni, T. S. Gunawan, M. H. Habaebi, M. Halbouni, M. Kartiwi, and R. Ahmad, "Machine learning and deep learning approaches for cybersecurity: A review," *IEEE Access*, vol. 10, pp. 19572–19585, 2022.

[2] 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. 30373–30385, 2019.

[3] G. Bovenzi, G. Aceto, D. Ciuonzo, V. Persico, and A. Pescapé, "A hierarchical hybrid intrusion detection approach in IoT scenarios," in *Proc. GLOBECOM 2020-2020 IEEE Glob. Commun. Conf.*, 2020, pp. 1–7.

[4] M. Verkerken et al., "A novel multi-stage approach for hierarchical intrusion detection," *IEEE Trans. Netw. Service Manage.*, vol. 20, no. 3, pp. 3915–3929, Sep. 2023.

[5] S. Kekki et al., "MEC in 5G networks," *ETSI White Paper*, vol. 28, no. 2018, pp. 1–28, 2018.

[6] P.-H. Kuo et al., "An integrated edge and fog system for future communication networks," in *Proc. 2018 IEEE Wireless Commun. Netw. Conf. Workshops*, 2018, pp. 338–343.

[7] A. Diro and N. Chilamkurti, "Leveraging LSTM networks for attack detection in fog-to-things communications," *IEEE Commun. Mag.*, vol. 56, no. 9, pp. 124–130, Sep. 2018.

[8] N. F. Syed, M. Ge, and Z. Baig, "Fog-cloud based intrusion detection system using recurrent neural networks and feature selection for IoT networks," *Comput. Netw.*, vol. 225, 2023, Art. no. 109662.

[9] A. Samy, H. Yu, and H. Zhang, "Fog-based attack detection framework for Internet of Things using deep learning," *IEEE Access*, vol. 8, pp. 74571–74585, 2020.

[10] S. Li, Y. Lu, and J. Li, "CAD-IDS: A cooperative adaptive distributed intrusion detection system with fog computing," in *Proc. IEEE 25th Int. Conf. Comput. Supported Cooperative Work Des.*, 2022, pp. 635–640.

[11] 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 Things J.*, vol. 8, no. 16, pp. 12569–12577, Aug. 2021.

[12] S. A. Rahman, H. Tout, C. Talhi, and A. Mourad, "Internet of Things intrusion detection: Centralized, on-device, or federated learning?," *IEEE Netw.*, vol. 34, no. 6, pp. 310–317, Nov./Dec. 2020.

[13] R. Zhao, Y. Yin, Y. Shi, and Z. Xue, "Intelligent intrusion detection based on federated learning aided long short-term memory," *Phys. Commun.*, vol. 42, 2020, Art. no. 101157.

[14] W. Lalouani and M. Younis, "Robust distributed intrusion detection system for edge of things," in *Proc. 2021 IEEE Glob. Commun. Conf.*, 2021, pp. 1–6.

[15] A. Selamnia, B. Brik, S. M. Senouci, A. Boualouache, and S. Hossain, "Edge computing-enabled intrusion detection for C-V2X networks using federated learning," in *Proc. GLOBECOM 2022-2022 IEEE Glob. Commun. Conf.*, 2022, pp. 2080–2085.

[16] Y. Labiod, A. Amara Korba, and N. Ghoualmi, "Fog computing-based intrusion detection architecture to protect IoT networks," *Wireless Pers. Commun.*, vol. 125, pp. 231–259, 2022.

[17] C. Antonio et al., "Hybrid approach to intrusion detection in fog-based IoT environments," *Comput. Netw.*, vol. 180, 2020, Art. no. 107417.

[18] M. Almiani, A. AbuGhazleh, A. Al-Rahayfeh, S. Atiewi, and A. Razaque, "Deep recurrent neural network for IoT intrusion detection system," *Simul. Modelling Pract. Theory*, vol. 101, 2020, Art. no. 102031.

[19] Y. -C. Lai et al., "Task assignment and capacity allocation for ML-based intrusion detection as a service in a multi-tier architecture," *IEEE Trans. Netw. Service Manage.*, vol. 20, no. 1, pp. 672–683, Mar. 2023.

[20] A. Lashkari, G. Drapper, and M. S. Islam, "CICFlowMeter," 2018. [Online]. Available: https://github.com/ahlashkari/CICFlowMeter

[21] M. Verkerken, L. D'hooge, T. Wauters, B. Volckaert, and F. De Turck, "Unsupervised machine learning techniques for network intrusion detection on modern data," in *Proc. 4th Cyber Secur. Netw. Conf.*, 2020, pp. 1–8.

[22] L. D'hooge, T. Wauters, B. Volckaert, and F. De Turck, "Inter-dataset generalization strength of supervised machine learning methods for intrusion detection," *J. Inf. Secur. Appl.*, vol. 54, 2020, Art. no. 102564.

[23] 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. 41525–41550, 2019.

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

[25] 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 Trans. Netw. Service Manage.*, vol. 17, no. 1, pp. 227–238, Mar. 2020.

[26] J. Madden, "Analysis: The economics of edge computing," 2020. [Online]. Available: https://edgecomputing-news.com/2020/10/29/analysis-economics-of-edge-computing/

[27] D. Sudyana, "Github repository containing code for the paper 'ml-based intrusion detection as a service: Traffic split offloading and cost allocation in a multi-tier architecture'," 2024. [Online]. Available: https://github.com/nycu-hsl/idas-optimization-paper

[28] W. Yahya, E. Oki, Y.-D. Lin, and Y.-C. Lai, "Scaling and offloading optimization in pre-cord and post-cord multi-access edge computing," *IEEE Trans. Netw. Service Manage.*, vol. 18, no. 4, pp. 4503–4516, Dec. 2021.

[29] Z. Sharif, L. T. Jung, I. Razzak, and M. Alazab, "Adaptive and priority-based resource allocation for efficient resources utilization in mobile-edge computing," *IEEE Internet Things J.*, vol. 10, no. 4, pp. 3079–3093, Feb. 2023.

Didik Sudyana received the Ph.D. degree from the Department of Electrical Engineering and Computer Science, National Yang Ming Chiao Tung University (NYCU), Hsinchu, Taiwan, in 2024. He is currently a Postdoctoral Fellow with High-Speed Network Lab, NYCU. His research interests include cybersecurity, machine learning, and network design and optimization.

Yuan-Cheng Lai received the Ph.D. degree from the Department of Computer and Information Science, National Chiao Tung University, Hsinchu, Taiwan, in 1997. In 2001, he joined the Faculty of the Department of Information Management at the National Taiwan University of Science and Technology, Taipei, Taiwan, and where he has been a distinguished Professor since 2012. His research interests include performance analysis, software-defined networking, wireless networks, and IoT security.

Ying-Dar Lin (Fellow, IEEE) received the Ph.D. degree in computer science from the University of California at Los Angeles, Los Angeles, CA, USA, in 1993. He was a Visiting Scholar with Cisco System, San Jose, CA, during 2007–2008; CEO with Telecom Technology Center, Taiwan, during 2010–2011; and Vice President of National Applied Research Labs, Taiwan, during 2017–2018. He cofounded L7 Networks Inc. in 2002 and O'Prueba Inc. in 2018. He is currently a Chair Professor of computer science with National Yang Ming Chiao Tung University, Hsinchu, Taiwan. His research interests include cybersecurity, wireless communications, network softwarization, and machine learning for communications. He has served or is serving on the editorial boards of several IEEE journals and magazines, including the Editor-in-Chief of IEEE COMMUNICATIONS SURVEYS AND TUTORIALS (COMST, 2017–2020).

Piotr Chołda received the Ph.D. degree in telecommunications from the AGH University of Science and Technology, Krakow, Poland, in 2006. He is currently an Associate Professor with the Institute of Telecommunications, AGH University of Krakow, Krakow. His research focuses on network management of networks (risk-aware performance evaluation and design; optimization of resources usage; and control with machine learning methods).