

Proactive Joint Load Balancing of Controllers, Servers, and Links in SD-IoMT Networks: A Fuzzy-LSTM Approach

Somaye Imanpour, Ahmadreza Montazerolghaem



Abstract—Internet of Multimedia Things (IoMT) extends the traditional Internet of Things (IoT) by enabling the transmission of rich multimedia flows. However, the exponential growth in connected devices, data volumes, and user demands increases traffic load, leading to server congestion and degraded Quality of Service (QoS). Legacy network architectures struggle to accommodate the scalability and dynamic nature of IoMT. Software-Defined Networking (SDN), with its centralized control and global network view, offers a promising solution for resource management and load balancing. Although prior research has addressed load balancing at the server, link, or controller level, either individually or in limited pairwise combinations, it lacks a holistic strategy that captures the interdependencies among all three. This work proposes a joint load-balancing approach across servers, links, and multi-domain SDN controllers. It employs Long Short-Term Memory (LSTM) networks to predict future server and controller loads, achieving a Root Mean Square Error (RMSE) as low as 0.3934, and integrates a fuzzy-logic system to categorize server levels and domain assignments. The results demonstrate significant improvements in intra-domain load distribution and show that balancing server and link loads directly reduces controller overhead. Furthermore, coordinated inter-domain load management helps minimize costly switch migrations, thereby enhancing overall network efficiency and reducing operational costs. The proposed approach enables more intelligent and proactive management of the Software-Defined Internet of Multimedia Things (SD-IoMT), improving QoS and overall performance. Quantitatively, the integrated multi-layer balancing improves the controller Jain's Fairness Index (JFI) by up to 20.6% (from 0.792 to 0.955).

Index Terms—Internet of Multimedia Things, Software-Defined Networking, Load Balancing, Fuzzy Logic, Long Short-Term Memory.

1 INTRODUCTION

WITH the exponential growth of multimedia traffic in IoT networks, achieving efficient multicomponent load balancing has emerged as a critical challenge. The rapid proliferation of heterogeneous IoT devices—including smart sensors, intelligent meters, and multimedia-enabled cameras—generates continuous, large-scale multimedia flows, making the management of massive data volumes a pressing concern [1]–[3].

Traditional network architectures struggle to meet the stringent scalability, latency, and QoS requirements of IoMT environments. To address these challenges, SDN has been introduced as a transformative paradigm, offering a global

and programmable view of the network by decoupling the control and data planes [4]. In SDN-based IoT networks, the architecture is logically divided into three distinct layers: the data, control, and application layers. This layered structure facilitates effective traffic management and enables optimized performance across network components, particularly for multimedia flows requiring strict QoS guarantees [5].

While numerous studies have addressed multicomponent load balancing within SDN-enabled IoT networks [6], most existing works have focused on individual components or limited pairwise combinations. A significant limitation of these approaches is the lack of a holistic strategy that accounts for the complex interdependencies among servers, links, and controllers. These interdependencies substantially impact overall network performance, particularly under the dynamic and heavy demands of multimedia traffic.

Therefore, this study proposes a holistic multicomponent load-balancing strategy that simultaneously distributes load across servers, links, and controllers, thereby ensuring consistent QoS and enhancing the performance, resilience, and throughput of IoMT networks.

1.1 Motivation and Contributions

The rapid growth of multimedia traffic in IoMT networks challenges scalable QoS management. Traditional SDN controllers face scalability bottlenecks, and existing solutions address server–link or controller–link interactions in isolation, neglecting controller overhead and holistic coordination. This paper proposes a multi-layer SDN framework integrating LSTM prediction, fuzzy-logic decisions, and intelligent switch migration to proactively balance server, controller, and link loads across domains with QoS-aware routing. The main contributions are:

- We design a novel SDN-based distributed control and management framework for IoMT networks.
- We address the dynamic multicomponent load balancing problem across servers, links, and controllers to enhance scalability and QoS.
- We formulate the server and controller load balancing as optimization problems and apply predictive and proactive strategies to handle them.
- We leverage time-series prediction LSTM and fuzzy logic to anticipate future resource usage and guide decision-making (LSTM captures long-term traffic patterns; fuzzy handles uncertainty).

• S. Imanpour and A. Montazerolghaem are with the Faculty of Computer Engineering, University of Isfahan, Isfahan, Iran. E-mail: s.imanpour@eng.ui.ac.ir and a.montazerolghaem@comp.ui.ac.ir
Copyright (c) 2026 IEEE. Personal use of this material is permitted. However, permission to use this material for any other purposes must be obtained from the IEEE by sending a request to pubs-permissions@ieee.org.

- We develop an intelligent switch migration mechanism based on predicted resource loads to maintain balanced control plane operations (Proactive migration avoids controller saturation).
- We implement a QoS-aware inter- and intra-domain routing mechanism that selects optimal paths according to the traffic class.
- We conduct extensive experiments to evaluate the performance of the proposed framework under various network conditions.

1.2 Organization

This paper is structured as follows: In Section 2, we review the related literature. The model and framework that we propose are presented in Section 3. Section 4 covers the implementation details and the evaluation of the framework's performance. Finally, Section 5 concludes the paper and outlines future research directions.

2 RELATED WORK

In the domain of SDN load balancing, existing literature primarily focuses on optimizing individual components servers, links, or controllers or explores limited pairwise interactions. Despite these efforts, a holistic strategy that accounts for the complex interdependencies across all three elements remains underexplored. We classify contemporary simultaneous load-balancing research into three categories: (i) server and link, (ii) server and controller, and (iii) controller and link domains.

2.1 Simultaneous Load Balancing Between Servers and Links

Research in this domain often integrates traffic prediction with decision-making modules. Montazerolghaem et al. [7], [8] introduced frameworks leveraging Normalized Least Mean Square (NLMS) prediction, fuzzy logic, and QoS classification to distribute IoT traffic across servers and paths, targeting multi-objective optimization for both load and energy consumption. Similarly, Abhishek et al. [9] utilized Artificial Neural Networks (ANNs) trained on resource metrics (CPU, memory, request rates) to facilitate traffic distribution within the Abilene topology. Furthermore, Zheng et al. [10] proposed IACO-LB, which selects optimal servers and paths based on real-time global resource status. While these strategies advance adaptive server-link balancing, they largely disregard the computational burden imposed on the SDN controller, thereby limiting their viability in controller-constrained network environments.

2.2 Simultaneous Load Balancing Between Servers and Controllers

Joint server-controller balancing is critical for multi-domain SDN deployments under high-density IoT traffic. To address this, Guo et al. [11] proposed a load-variance-based synchronization (LVS) mechanism designed to mitigate synchronization overhead via threshold-triggered updates. Li et al. [12] combined the Load-Oriented Control Placement (LOCP) for delay minimization with a Hierarchical Adaptive Controller Allocation (HACA) strategy for resource assignment under reliability constraints. Additionally, Babbar et al. [13] developed a QoS-aware switch migration algorithm for intra- and inter-domain load redistribution. Although these methods enhance synchronization efficiency and dynamic mi-

gration, they frequently overlook controller bottlenecks during persistent flows, potentially underestimating the actual resource pressure in heterogeneous network environments.

2.3 Simultaneous Load Balancing Between Controllers and Links

Research focusing on controller and link coordination is relatively nascent. Wang et al. [14] formulated the link-controller load balancing routing (LBR-LC) problem as an NP-hard optimization and proposed a rounding-based approximation algorithm, achieving a 70% reduction in maximum controller response time with a marginal link load increase of 3%. Despite their efficacy in link-controller coordination, such approaches often neglect the aggregate controller processing load in distributed, high-volume scenarios, which hinders scalability in large-scale networks.

2.4 Summary of Limitations and Proposed Approach

Existing pairwise load-balancing solutions, while effective within specific domains, fail to provide comprehensive coordination across the entire network infrastructure. By focusing on isolated interactions, they overlook the aggregate impact of controller processing, which is exacerbated by persistent multimedia flows. The proposed multi-layer framework addresses these gaps by integrating LSTM-based prediction with fuzzy-logic migration decisions. This approach enables accurate load estimation, enhanced inter-domain fairness, and improved Quality of Experience (QoE) in multi-domain SD-IoMT networks under dynamic and heterogeneous traffic conditions.

3 THE PROPOSED MODEL AND FRAMEWORK

3.1 The System Model

To formally describe the network model, let $C = \{c_1, \dots, c_K\}$ and $S = \{s_1, \dots, s_N\}$ denote the sets of all controllers and switches, respectively, where K and N represent their total counts. Each controller c_j manages N_j' switches, with s_n^j denoting the n -th switch in domain j . Similarly, let $H = \{h_1, \dots, h_M\}$ represent the set of M total servers, where M_j' denotes the number of servers in domain j , and h_i^j represents the i -th server therein. All notations are summarized in Table 1.

IoMT applications, including video surveillance, remote health monitoring, and smart traffic management, generate delay-sensitive, high-bitrate multimedia flows with stringent QoS requirements. Figure 1 illustrates the proposed multi-domain SD-IoMT architecture, in which controllers classify flows, switches prioritize queues, and servers distribute multimedia workloads.

The proposed framework maximizes flow utility by selecting paths that balance the load across servers, controllers, and links while satisfying bandwidth, delay, and hop-count constraints. The objective function balances QoS gains against penalties associated with overloaded resources, thereby favoring underloaded paths and domains.

The constraints ensure that each flow is assigned to one feasible path, each switch is associated with one controller, server and controller capacities are respected, link QoS requirements are satisfied, and performance metrics are normalized.

The resulting Mixed-Integer Nonlinear Programming (MINLP) problem involves discrete and continuous variables under capacity and responsiveness constraints. Given the

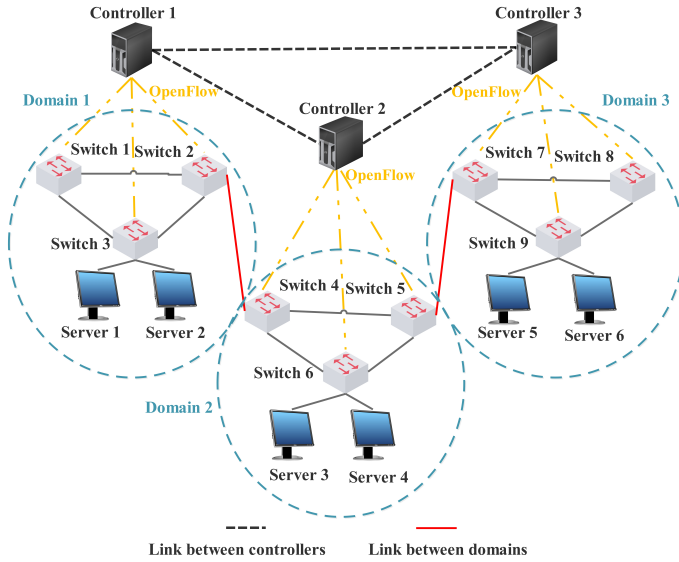


Figure 1. Multi-domain SD-IoMT network model with K controllers managing N switches and M servers in total. Inter-domain links support coordination and load redistribution across domains.

coupling between the control and data planes and the real-time requirements of SD-IoMT, exact optimization is computationally impractical. Therefore, a scalable, low-complexity heuristic is developed to achieve near-optimal load balancing while preserving QoS.

The objective maximizes network utility by selecting paths that balance load and QoS (Eq. 1). For each path $p \in P$, utility f_p trades off the QoS score q_p against the load penalty r_p , weighted by β_1 and β_2 (Eq. 2). The QoS score combines normalized bandwidth b_p , delay d_p , and hop count h_p using weights γ , δ , and ϕ (Eq. 3).

The path set P comprises inter-domain routes O and intra-domain routes I_j (Eq. 4). Path bandwidth is limited by the minimum link bandwidth, whereas delay and hop count are accumulated over the path links (Eqs. 5–10).

The load penalty combines server load L_{h_i} and controller load L_{c_j} (Eq. 11). Server load aggregates normalized CPU, memory, hard disk, and bandwidth utilizations using weights a_1 – a_4 , whose sum equals one (Eq. 12).

The resource metrics are bounded by their capacities in Eqs. (17)–(18), while the load and QoS variables are normalized to $[0, 1]$ in Eq. (19). The corresponding weighting constraints are given in Eqs. (20)–(21).

As shown in Figure 2, the framework integrates server, controller, and link load balancing through LSTM prediction and fuzzy decision-making. Server and controller balancing estimate CPU, memory, hard disk, bandwidth, and PACKET-IN loads and trigger switch migration when needed. Link balancing evaluates bandwidth, utilization, and delay, classifies traffic into three classes, and selects optimal intra- or inter-domain paths. This model underpins the three-layer framework described next.

3.2 Server Load Balancing

The primary component of the SD-IoMT network load balancing architecture is server load balancing. Within each domain, the controller executes a dedicated algorithm de-

$$\text{maximize } \sum_{p \in P} f_p u_p \quad (1)$$

subject to:

$$f_p = \sum_p (\beta_1 q_p - \beta_2 r_p), \quad \forall p \in P \quad (2)$$

$$q_p = \gamma \frac{b_p}{B_p} - \delta \frac{d_p}{D_p} - \phi \frac{h_p}{H_p}, \quad \forall p \in P \quad (3)$$

$$D_p = \sum_{x \in (P=O \cup I_j)} d_x, \quad \forall j = 1, \dots, K \quad (4)$$

$$b_{p_{jk}} \leq b_{nm}^{jk} u_{p_{jk}}, \quad (n, m) \in p_{jk}, \quad \forall p_{jk} \in O \quad (5)$$

$$d_{p_{jk}} = \sum_{(n, m) \in p_{jk}} d_{nm}^{jk} u_{p_{jk}}, \quad \forall p_{jk} \in O \quad (6)$$

$$h_{p_{jk}} = \sum_{(n, m) \in p_{jk}} |\{n, m\}| u_{p_{jk}}, \quad \forall p_{jk} \in O \quad (7)$$

$$b_{p_j} \leq b_{nm}^j u_{p_j}, \quad (n, m) \in p_j, \quad \forall p_j \in I_j \quad (8)$$

$$d_{p_j} = \sum_{(n, m) \in p_j} d_{nm}^j u_{p_j}, \quad \forall p_j \in I_j \quad (9)$$

$$h_{p_j} = \sum_{(n, m) \in p_j} |\{n, m\}| u_{p_j}, \quad \forall p_j \in I_j \quad (10)$$

$$r_p = L_{h_i} + L_{c_j} \quad (11)$$

$$L_{h_i} \leq \left(a_1 \frac{Ld_i^{CPU}}{\vartheta_i^x} + a_2 \frac{Ld_i^{mem}}{\vartheta_i^y} + a_3 \frac{Ld_i^{hdd}}{\vartheta_i^z} + a_4 \frac{Ld_i^{bw}}{\vartheta_i^w} \right) u_p \quad (12)$$

$$\sum_{j=1}^K s_n^j = 1, \quad \forall n = 1, \dots, N \quad (13)$$

$$L_{c_j} = \sum_{s_n^j \in j} L_{s_n^j}, \quad \forall j = 1, \dots, K, \quad \forall n = 1, \dots, N'_j \quad (14)$$

$$L_{c_j} \leq \Theta_j, \quad \forall j = 1, \dots, K \quad (15)$$

$$\sum_{p \in P} u_p = 1 \quad (16)$$

$$b_p \leq B_p, \quad d_p \leq D_p, \quad h_p \leq H_p \quad (17)$$

$$Ld_i^{CPU} \leq \vartheta_i^x, \quad Ld_i^{mem} \leq \vartheta_i^y, \quad Ld_i^{hdd} \leq \vartheta_i^z, \quad Ld_i^{bw} \leq \vartheta_i^w \quad (18)$$

$$0 \leq q_p \leq 1, \quad 0 \leq L_{h_i}, L_{c_j} \leq 1, \quad 0 \leq L_{s_n^j}, \Theta_j \leq 1 \quad (19)$$

$$0 \leq \beta_1, \beta_2 \leq 1, \quad a_1 + a_2 + a_3 + a_4 = 1 \quad (20)$$

$$\sum_p (\gamma + \delta + \phi) u_p = 1 \quad (21)$$

Variables: $u_p \in \{0, 1\}$, $s_n^j \in \{0, 1\}$, $q_p, r_p, b_p, d_p, h_p \geq 0$

signed to distribute computational tasks effectively across the available servers. This component is structured into several functional submodules, as detailed below.

3.2.1 Server Load Measurement

As outlined in Algorithm 1, the process begins by quantifying the load of all M'_j servers within domain j . Specifically, normalized utilization ratios for CPU, memory, hard disk, and bandwidth—denoted as x_i, y_i, z_i , and w_i , respectively—are computed, each ranging from 0 to 1. Subsequently, an LSTM model processes a historical $P \times W$ matrix (where $P = 4$ represents the resource types and W denotes the

Table 1
Key Nomenclatures

System Entities	
C, c_j, K	Set of controllers, controller j , and total number of controllers
S, s_n^j, N, N_j', Q	Set of switches, switch n in domain j , total number of switches, and switches per domain, Maximum number of switches per domain
H, h_i^j, M, M_j'	Set of servers, server i in domain j , total number of servers, and servers per domain
D	Set of domains
Server Load and Prediction	
$L_{h_i^j}$	Load of server h_i^j in domain j
$x_i, Ld_i^{CPU}, \vartheta_i^x$	CPU utilization ratio (normalized), current CPU usage in abnormal condition, and total CPU capacity of server i
$f_{h_i^j}$	Resource utilization matrix of server h_i^j in domain j
$\hat{x}_{t+1}^i$	Predicted CPU utilization at time $t + 1$ for server i
$L_{h_j}^{CPU}$	Average CPU usage of servers in domain j
Controller and Switch Load	
$L_{s_n^j}^r$	Load on controller c_j caused by switch s_n^j for resource r
$Load_n^r$	Utilization level of switch s_n^j for resource r
θ_j^r	Capacity of controller c_j for resource r
$L_{s_n^j}^j$	Aggregated resource load of switch s_n^j
$L_{c_j}, \bar{L}_{c_j}$	Total and average load of controller c_j
LC_j^r	Resource r utilization of controller c_j
$x_{qw}, \hat{x}_{t+1}^n$	PACKET_IN messages from switch q at time w and predicted for s_n^j at $t + 1$
$a_1, \dots, a_4$	Weights for CPU, memory, HDD, and bandwidth in load aggregation
Link Load and Routing	
$B_{nm}^j, U_{nm}^j, D_{nm}^j$	Bandwidth, utilization, and delay of intra-domain link between s_n^j and s_m^j
$B_{nm}^{jk}, U_{nm}^{jk}, D_{nm}^{jk}$	Bandwidth, utilization, and delay of inter-domain link between domains j and k
b_{nm}^j, b_{nm}^{jk}	Effective bandwidth of intra-/inter-domain link
b_{p_j}, d_{p_j}	Minimum effective bandwidth and total delay on intra-domain path
$b_{p_{j,k}}, d_{p_{j,k}}$	Same for inter-domain path between j and k
ϕ, ϕ_k	Candidate servers in source domain and domain D_k
ϑ	Set of candidate domains for routing
$L_{h_i^j}, L_{h_i^k}$	Server load before and after migration
p_j, p_{jk}	Selected intra- and inter-domain paths

sliding window size) to predict future load values, specifically f_{CPU} , f_{Mem} , f_{Hdd} , and f_{BW} , for each server.

3.2.2 Server Load Prediction and Classification Using LSTM and Fuzzy Logic

In the SD-IoMT environment, we utilize a LSTM network—a specialized type of RNN to forecast server resource consumption. The architecture consists of one LSTM layer comprising n neurons, followed by a dense output layer. The data is normalized and partitioned into 80% training and 20% testing sets. Using a sliding lookback window, the model predicts future values for CPU, memory, hard disk, and bandwidth consumption, denoted as $\hat{x}_{t+1}^i$, $\hat{y}_{t+1}^i$, $\hat{z}_{t+1}^i$, and $\hat{w}_{t+1}^i$, respectively. By capturing temporal dependencies, this forecasting mechanism significantly enhances the precision of subsequent resource optimization.

Following the prediction phase, a fuzzy logic system is employed to classify server load levels under uncertainty. Fuzzy logic is particularly effective for modeling nonlinear relationships and supporting decision-making processes when dealing with imprecise data. The classification process utilizes

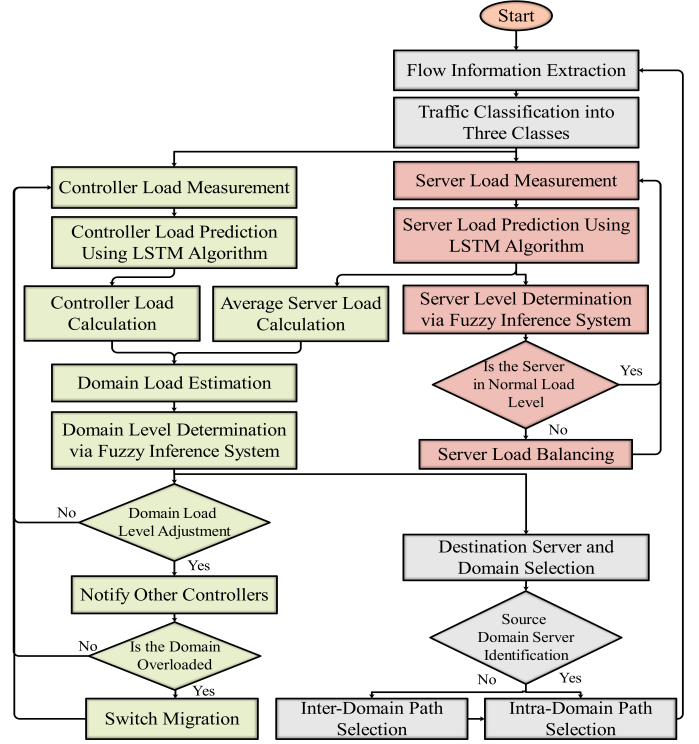


Figure 2. Overall architecture of the proposed multi-domain SD-IoMT load balancing framework

Algorithm 1: Server Load Measurement with LSTM Prediction

Input: $C_j, Ld_i^{CPU}, Ld_i^{mem}, Ld_i^{hdd}, Ld_i^{bw}, h_i^j$

Output: $f_{CPU}, f_{Mem}, f_{Hdd}, f_{BW}$

- 1 **for** $h_i^j \in C_j$ **do**
- 2 Compute normalized utilization ratios (0 to 1):
- 3 $x_i = \frac{Ld_i^{CPU}}{\vartheta_i^x}$ (CPU), $y_i = \frac{Ld_i^{mem}}{\vartheta_i^y}$ (memory),
- 4 $z_i = \frac{Ld_i^{hdd}}{\vartheta_i^z}$ (hard disk), $w_i = \frac{Ld_i^{bw}}{\vartheta_i^w}$ (bandwidth)
- 5 Construct historical matrix $f_{h_i^j}$ with dimensions $P \times W$, where $P = 4$ (four resource metrics: CPU, memory, hard disk, bandwidth), and W = window size (number of time steps). Each row corresponds to one resource metric across W time steps:
- 6 Construct $f_{h_i^j} = \begin{bmatrix} x_{11} & x_{12} & \dots \\ \vdots & \ddots & \vdots \\ x_{P1} & \dots & x_{PW} \end{bmatrix}$;
- 7 **if** $M_j' < P$ (fewer servers than metrics) **then**
- 8 Set $x_{PW} = 0, p = 4, w = 1, 2, \dots, W$;
- 9 Apply LSTM on $f_{h_i^j}$ to predict $f_{CPU}, f_{Mem}, f_{Hdd}, f_{BW}$

the four predicted utilization metrics from the LSTM model: $\hat{x}_{t+1}^i$ (CPU), $\hat{y}_{t+1}^i$ (Memory), $\hat{z}_{t+1}^i$ (hard disk), and $\hat{w}_{t+1}^i$ (bandwidth). To reduce computational complexity and ensure real-time scalability within the SDN controller, these inputs are grouped into two complementary pairs: (x, y) representing (CPU, memory) and (z, w) representing (hard disk, bandwidth). Each pair is fuzzified into three linguistic terms—Low, Medium, and High—resulting in nine combinations per pair (ranging from LL to HH).

The fuzzy logic system implements 81 Mamdani-type rules

(9×9) that encompass all possible four-input combinations. The output categorizes the server status into one of four load levels: *Under-load*, *Normal-load*, *Highly-load*, or *Over-load*. For instance, the rule: **If** $(x, y = LL)$ and $(z, w = LL)$, **then** *Under-load* serves as an illustrative example. The complete rule base, established through expert thresholds and empirical observations, is presented in Table 2 as a 9×9 matrix. This compact representation ensures both interpretability and space efficiency, effectively avoiding the necessity of listing each rule individually.

3.2.3 Server Load Distribution

When a flow enters the domain, servers at the *under-load* level have the highest priority to receive traffic, followed by servers at the *normal-load* level, and finally servers at the *highly-load* level.

When a server h_i^j in domain C_j is at *over-load* level, the algorithm first identifies candidate servers τ within C_j whose levels are either *under-load* or *normal-load*. The final server selection is performed based on a weighted evaluation of both the current and predicted fuzzy outputs. For each server k , the fuzzy outputs F_{real}^k and F_{pred}^k are first normalized within the system output range $[v_{\min}, v_{\max}]$ as $N_{\text{real}}^k = \frac{F_{\text{real}}^k - v_{\min}}{v_{\max} - v_{\min}}$ and $N_{\text{pred}}^k = \frac{F_{\text{pred}}^k - v_{\min}}{v_{\max} - v_{\min}}$. Then, a combined weight $W_{\tau+1}^k = \omega_{\text{real}} N_{\text{real}}^k + \omega_{\text{pred}} N_{\text{pred}}^k$ is computed, where ω_{real} and ω_{pred} represent the contribution ratios of real and predicted fuzzy values, respectively. Finally, the server with the minimum $W_{\tau+1}^k$ value is selected as the optimal server for new flow allocation.

3.3 Controller Load Balancing

The second component of the SD-IoMT load-balancing architecture addresses controller load balancing. This component is structured into several functional submodules, detailed as follows.

3.3.1 Controller Load Measurement and Calculation

In the initial stage of controller load balancing, the load for each controller is estimated by analyzing PACKET-IN message rates and the throughput of active flows originating from switches within its domain. These metrics are monitored at the switch level and subsequently processed via an LSTM model—mirroring the server load prediction approach—to forecast future controller load. Historical PACKET-IN counts are maintained in a $Q \times W$ matrix, where Q represents the maximum number of switches in any domain (i.e., the capacity of the domain with the highest switch density) and W denotes the sliding window size. Each element x_{qw} signifies the packet count transmitted by switch q to its domain controller j during window w . For domains with fewer than Q switches ($N_j' < Q$), we set $x_{qw} = 0$ for indices $q = N_j' + 1, \dots, Q$. Flow throughput is organized using an identical matrix structure to maintain consistency.

Packet predictions are generated using the same LSTM-based methodology employed for server load estimation. The model accepts the historical packet count matrix as input and yields the predicted packet count, $\hat{x}_{t+1}^n$. Finally, for each switch s_n^j and resource $r \in \{\text{CPU, Memory, Hard Disk, Bandwidth}\}$, the resource-specific load is derived by aggregating the predicted PACKET-IN messages, $\hat{p}kt_n$, and the predicted flow throughput, $\hat{throughput}_n$, as follows:

$$Load_n^r = \alpha_1^r \cdot \hat{p}kt_n + \alpha_2^r \cdot \hat{throughput}_n \quad (22)$$

where α_1^r and α_2^r represent the impact of signaling and data traffic on resource r . For example, $Load_n^{CPU}$ depends on both terms, while $Load_n^{bw}$ uses only $\alpha_1^{bw} \cdot \hat{p}kt_n$.

the normalized load of each switch with respect to the controller's capacity is obtained as:

$$0 \leq L_{s_n^j}^r = \frac{Load_n^r}{\theta_j^r} \leq 1, \quad r \in \{\text{CPU, mem, hdd, bw}\} \quad (23)$$

where θ_j^r is controller j 's capacity for resource r . The total switch load (as a percentage) is:

$$L_{s_n^j} = [a_1 L_{s_n^j}^{CPU} + a_2 L_{s_n^j}^{\text{mem}} + a_3 L_{s_n^j}^{\text{hdd}} + a_4 L_{s_n^j}^{\text{bw}}] \times 100 \quad (24)$$

with weights a_1 to a_4 indicating relative importance. Finally, the total and average controller load are:

$$L_{c_j} = \sum_{s_n^j \in c_j}^{N_j'} L_{s_n^j}, \quad (25)$$

$$\bar{L}_{c_j} = \frac{L_{c_j}}{N_j'} \quad (26)$$

where N_j' is the number of switches in domain j .

The total resource consumption of each controller (CPU, Memory, Hard Disk, Bandwidth) is computed as follows:

$$L_{c_j}^r = \sum_{s_n^j \in c_j} L_{s_n^j}^r \quad (27)$$

3.3.2 Server and Domain Load Calculation

After computing controller load, the average predicted resource utilization across all M_j' servers in domain j is:

$$Lh_j^r = \frac{1}{M_j'} \sum_{h_i^j \in c_j} \hat{r}_{t+1}^i \quad (28)$$

The domain load for each resource type r combines average server load and controller load:

$$LD_j^r = \lambda_1 Lh_j^r + \lambda_2 Lc_j^r \quad (29)$$

3.3.3 Domain Multi-Threshold Classification

Consistent with the server load classification approach, a Mamdani-type fuzzy inference system utilizing 81 rules is employed to categorize domains into three operational states: *Under-load*, *Normal-load*, and *Over-load*. Notably, the *Highly-load* category is excluded at this level, as domain-level aggregation effectively dampens extreme resource fluctuations. The system inputs comprise the four predicted aggregate resource metrics—CPU, memory, hard disk, and bandwidth—denoted as $\hat{x}_{t+1}^i, \hat{y}_{t+1}^i, \hat{z}_{t+1}^i$, and $\hat{w}_{t+1}^i$, respectively. Each input is mapped to three linguistic fuzzy sets: *Low*, *Medium*, and *High*. An illustrative rule is presented as follows:

If $(x \text{ is Medium})$ and $(y \text{ is Medium})$ and $(z \text{ is High})$ and $(w \text{ is High})$, **then** domain class is *Over-load*.

These rules are formulated based on expert knowledge and system behavior patterns to accurately capture domain-level performance under varying workloads.

3.4 Switch Migration Operation

To balance controller load in an *over-load* domain, switches are migrated to *under-load* or *normal-load* target domains. The average switch load in overloaded domain C_j is:

Table 2
Fuzzy Rules Based on Input Pairs (x, y) and (z, w)

$(x, y) \backslash (z, w)$	LL	LM	ML	MM	LH	HL	MH	HM	HH
LL	Under	Under	Under	Under	Normal	Normal	Normal	Normal	Highly
LM	Under	Under	Under	Normal	Normal	Normal	Normal	Normal	Highly
ML	Under	Under	Under	Normal	Normal	Normal	Normal	Normal	Highly
MM	Under	Normal	Normal	Normal	Normal	Normal	Highly	Highly	Highly
LH	Normal	Normal	Normal	Normal	Highly	Highly	Highly	Highly	Over
HL	Normal	Normal	Normal	Normal	Highly	Highly	Highly	Highly	Over
MH	Normal	Normal	Normal	Highly	Highly	Highly	Highly	Highly	Over
HM	Normal	Normal	Normal	Highly	Highly	Highly	Highly	Highly	Over
HH	Highly	Highly	Highly	Highly	Over	Over	Over	Over	Over

Abbreviations: L = Low, M = Medium, H = High. For example, LL = Low-Low, HM = High-Medium.

Table 3
Fuzzy Rules for Load Classification Based on (x, y) and (z, w) (Second Scenario)

$(x, y) \backslash (z, w)$	LL	LM	ML	MM	LH	HL	MH	HM	HH
LL	Under	Under	Under	Under	Normal	Normal	Normal	Normal	Over
LM	Under	Under	Under	Under	Normal	Normal	Normal	Normal	Over
ML	Under	Under	Under	Under	Normal	Normal	Normal	Normal	Over
MM	Normal	Normal	Normal	Normal	Normal	Normal	Over	Over	Over
LH	Normal	Normal	Normal	Normal	Normal	Normal	Over	Over	Over
HL	Normal	Normal	Normal	Normal	Normal	Normal	Over	Over	Over
MH	Normal	Normal	Normal	Over	Over	Over	Over	Over	Over
HM	Normal	Normal	Normal	Over	Over	Over	Over	Over	Over
HH	Over	Over	Over	Over	Over	Over	Over	Over	Over

Abbreviations: L = Low, M = Medium, H = High. For example, LL = Low-Low, HM = High-Medium.

$$\bar{L}_{c_j} = \frac{L_{c_j}}{N_j^r} \quad (30)$$

Switches with load greater than $\bar{L}_{c_j}$ become migration candidates. For each candidate switch s_n^k migrating to target controller C_k , the normalized resource impact is:

$$0 \leq L_{s_n^k}^r = \frac{\text{Load}_r}{\theta_k^r} \leq 1 \quad (31)$$

where Load_r denotes resource consumption from PACKET-IN messages, and θ_k^r is controller k 's capacity. Server load in the target domain is then evaluated using the fuzzy inference system. A target server is eligible only if its post-migration load remains *under-load* or *normal-load*; among eligible servers, the one with the lowest load is selected. The migration cost depends on the target domain's state. For *under-load* targets:

$$M_{jk} = b_1 \cdot d_{jk}^n \quad \text{if } D_k \in \text{Under-load} \quad (32)$$

If the target domain is in a *normal-load* state, the cost is computed similarly, but with a different weight factor b_2 , where $b_2 > b_1$ (typically 1.5 to 2 times greater):

$$M_{jk} = b_2 \cdot d_{jk}^n \quad \text{if } D_k \in \text{Normal-load} \quad (33)$$

The switch-target pair with minimum M_{jk} is selected for migration. Algorithm 2 outlines the procedure.

3.5 Link Load Balancing

The third component of the proposed SD-IoMT load-balancing framework addresses link load balancing. This component comprises several functional submodules, as described below.

3.5.1 Link Load Measurement

Each controller monitors both intra-domain and inter-domain links. For an intra-domain link $e(s_n^j, s_m^j) \in I_j$, where $s_n^j, s_m^j \in D_j$, the controller measures the available bandwidth B_{nm}^j , link utilization U_{nm}^j , and delay D_{nm}^j . For an inter-domain link $e(s_n^j, s_m^k) \in O$, where $j \neq k$, the corresponding metrics are denoted by B_{nm}^{jk} , U_{nm}^{jk} , and D_{nm}^{jk} , respectively.

Based on its QoS requirements, each incoming traffic flow is classified into one of three classes: *Bandwidth-centric*, *Delay-centric*, or *Best-effort-centric*. This classification subsequently guides destination selection and path computation.

3.5.2 Destination Server and Domain Selection

After measuring the link states and classifying the incoming traffic flow, the controller selects an appropriate destination server to process the traffic. The selection procedure is presented in Algorithm 3. The controller first identifies eligible servers within the source domain. A local server is considered eligible if its current state is *Under-load*, *Normal-load*, or *Highly-load*, provided that its estimated post-migration state does not become *Over-load*.

3.5.3 Path Routing Based on Traffic Class

Once the destination server and domain are selected, the appropriate path is chosen based on whether the destination resides in another domain (inter-domain) or the same domain (intra-domain). Algorithm 4 performs this selection for both cases, taking as input either the inter-domain path set O or the intra-domain path set I_j .

For intra-domain routing, $\mathcal{P} = I_j$, and metrics $B_{nm}^j, U_{nm}^j, D_{nm}^j$ are used. For inter-domain routing, $\mathcal{P} = O$, with metrics $B_{nm}^{jk}, U_{nm}^{jk}, D_{nm}^{jk}$.

Algorithm 2: Switch Migration

Input: C_j, s_n^j , Domain load, $L_{s_n^j}, h_i^k$, Load of other domains
Output: Migrating switch and target domain

```

1 if  $D_j$  in under-load class then
2    $D_U \leftarrow D_j$ ;
3 else if  $D_j$  in normal-load class then
4    $D_N \leftarrow D_j$ ;
5 else if  $D_j$  in over-load class then
6    $D_O \leftarrow D_j$ ;
7 for  $D_j \in D_O$  do
8   for  $s_n^j \in C_j$  do
9     if  $L_{s_n^j} \geq \bar{L}_{c_j}$  then
10        $\rho \leftarrow s_n^j$ ;
11 for  $D_k \in D_U \cup D_N$  do
12   for  $s_n^j \in \rho$  do
13     Calculate new controller load using
      Equations (26)–(29);
14   for  $h_i^k$  in under-load or normal-load class in  $D_k$  do
15     Calculate server load after migration using fuzzy
      inference;
16     if  $h_{i*}^k \in \text{under-load or normal-load}$  then
17        $\phi \leftarrow h_i^k$ ;
18   for  $h_i^k \in \phi$  do
19     Select the server with the lowest load level;
20   Recalculate domain load level after migration;
21   if  $D_k^* \in D_U$  then
22      $M_{jk} = b_1 \cdot d_{jk}^n$ ;
23      $\gamma \leftarrow (s_n^j, D_k, M_{jk})$ ;
24   else if  $D_k^* \in D_N$  then
25      $M_{jk} = b_2 \cdot d_{jk}^n$ ;
26      $\gamma \leftarrow (s_n^j, D_k, M_{jk})$ ;
27 return Switch-domain pair with minimum  $M_{jk}$ 

```

4 IMPLEMENTATION AND PERFORMANCE EVALUATION

4.1 Simulation Environment

A simulation testbed was built using Mininet and the Floodlight SDN controller (OpenFlow 1.3).

Figure 3 illustrates the multi-domain testbed topology with three interconnected domains (Domain 1–3), each managed by one controller and equipped with homogeneous servers of identical processing and bandwidth capacities to ensure fair load redistribution. Inter-domain coordination is activated under overload or imbalance.

Mininet provides lightweight topology creation and real-time traffic control via Open vSwitch, while Floodlight handles flow installation and path computation. Monitoring is performed using sFlow-RT for CPU, bandwidth, and link utilization, with Prometheus/Grafana for visualization. Traffic is generated using iPerf (100–900 pps, 10 Mbps link), repeated three times and averaged.

To ensure practical applicability, the LSTM retraining interval and fuzzy membership boundaries were calibrated using empirical observations from the reference QoS-aware implementation and validated through controlled sensitivity checks in Mininet. Small variations in retraining frequency and membership breakpoints did not significantly affect migration decisions or QoS metrics, indicating that the frame-

Algorithm 3: Select the Destination Server and Domain

Input: C_j, h_i^j
Output: Destination server and domain

```

1 for  $h_i^j \in D_j$  and  $L_{h_i^j} \in \{\text{under-load, normal-load, highly-load}\}$  do
2   if  $L_{h_i^j} == \text{under-load or normal-load or highly-load}$  then
3      $\phi \leftarrow h_i^j$ ;
4 if  $\phi == \emptyset$  then
5   for  $D_k \in D_U \cup D_N, j \neq k$  do
6     for  $h_i^k \in D_k$  and  $h_i^k$  in under-load or normal-load do
7       Simulate load after migration:  $L_{h_i^k}$ ;
8       if  $L_{h_i^k} \in \{\text{under-load, normal-load}\}$  then
9          $\phi_k \leftarrow h_i^k$ ;
10      if  $\phi_k \neq \emptyset$  then
11         $\vartheta \leftarrow D_k$ ;
12 for  $D_k \in \vartheta$  do
13   Select the domain with the shortest distance;
14 for  $h_i^k \in \phi_k$  and  $D_k$  is selected do
15   Choose the server with minimum load; if equal,
    choose the one with least CPU usage;

```

Algorithm 4: Routing Based on Traffic Class (Inter or Intra-Domain)

Input: Traffic class, path set $\mathcal{P}$ (either O for inter-domain or I_j for intra-domain)
Output: Selected path

```

1 for each path  $p \in \mathcal{P}$  do
2    $b_{nm} = B_{nm} \times (1 - U_{nm})$  (available bandwidth for
    each link);
3    $b_p = \min\{b_{nm} \mid (n, m) \in p\}$  (bottleneck bandwidth of
    path);
4    $d_p = \sum_{(n, m) \in p} D_{nm}$  (total path delay);
5 if class == delay-centric then
6   Select  $p \in \mathcal{P}$  where  $d_p$  is minimum;
7 else
8   if class == best-effort-centric then
9     Select the shortest path (minimum hop count);
10  else
11    Select  $p \in \mathcal{P}$  where  $b_p$  is minimum (most available
    bandwidth);

```

work is not overly sensitive to parameter tuning and can be deployed in heterogeneous SDN environments without strict reconfiguration.

4.2 Experimental Results

4.2.1 Comparative Analysis of LSTM and NLMS Predictors

The proposed LSTM network was benchmarked against the conventional NLMS algorithm for predicting CPU and memory utilization. Performance was evaluated using RMSE, Mean Absolute Error (MAE), Mean Absolute Percentage Error (MAPE), Correlation Coefficient (Corr), and Nash–Sutcliffe Efficiency (NSE).

RMSE, MSE, and MAE quantify prediction errors, with lower values indicating higher accuracy, while MAPE expresses the error relative to actual values. Corr measures the linear association between predicted and actual values,

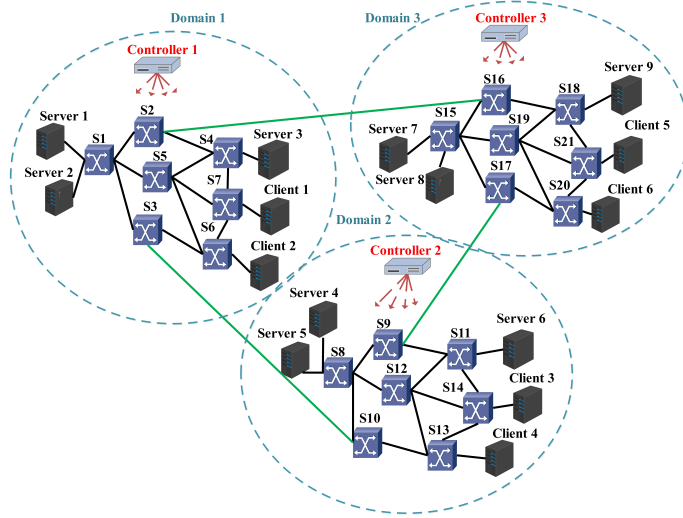


Figure 3. The testbed of the proposed framework

Table 4

Comparison of Prediction Performance between LSTM and NLMS

Model	Target	RMSE	MAE	MAPE	Corr	NSE
LSTM	CPU	0.7430	0.6800	3.21%	-0.3386	—
	Memory	0.3934	0.1798	1.37%	0.9614	—
NLMS	CPU	2.000	—	2.87%	—	-1.338
	Memory	1.296	—	2.36%	—	-1.833

ranging from -1 to $+1$. NSE compares prediction errors with the variance of observed data.

As shown in Table 4, LSTM consistently outperformed NLMS. Its low RMSE values (0.7430 for CPU and 0.3934 for memory) and high Corr (up to 0.9614) demonstrate effective temporal learning and nonlinear modeling. Although Figure 4 shows three legend entries, the actual and test prediction curves frequently overlap, confirming high prediction accuracy. Conversely, NLMS produced substantially higher errors, indicating limited ability to capture complex temporal correlations (Fig. 5).

4.2.2 Load Balancing Across Servers

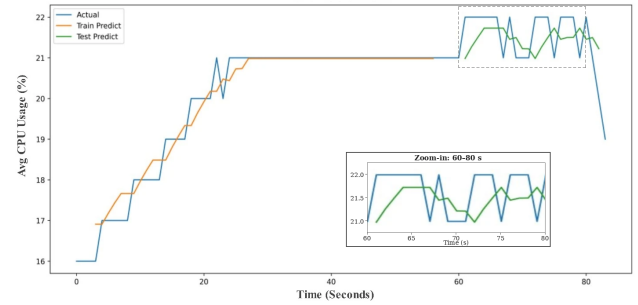
Figure 6 shows CPU utilization across traffic classes. In the BE-centric scenario, the proposed method achieves lower and more balanced CPU utilization than Random, which lacks resource awareness. Unlike the QoS-aware method in [7], our method incorporates bandwidth and hard disk load indicators and uses LSTM to improve prediction accuracy, although [7] provides slightly more precise fuzzy classification. Both perform similarly under moderate loads, whereas our framework performs better under high-load conditions, such as congestion and hard disk saturation.

4.2.3 Controller Load Estimation Methods

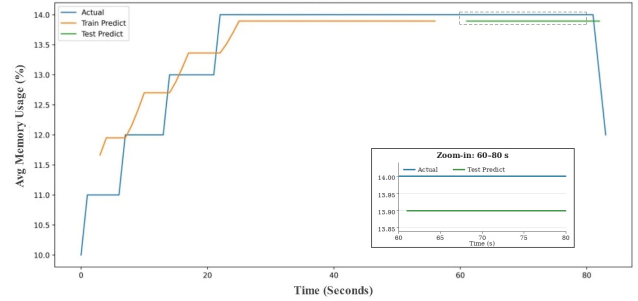
Three methods for estimating controller load in SDN-based SD-IoMT are compared.

A. PACKET-IN Based Load Estimation (CLE-PacketIn)

This method estimates load solely from PACKET-IN messages. Experiments with six UDP flows (100–1100 pps) show PACKET-IN counts remain nearly constant for persistent flows, underestimating CPU/memory usage (Figure 7). It is computationally simple but lacks precision under high throughput.

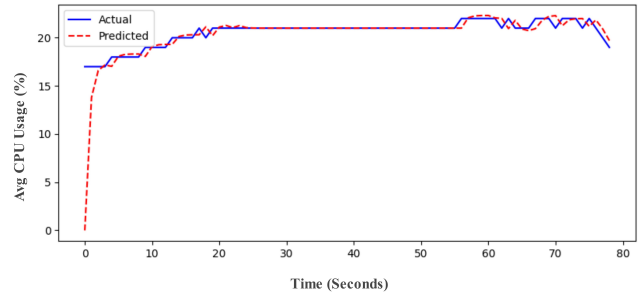


(a) CPU Utilization

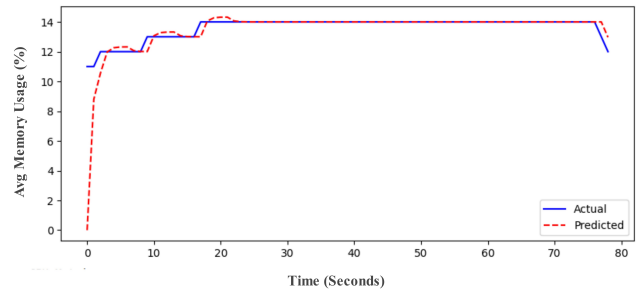


(b) Memory Utilization

Figure 4. LSTM Prediction



(a) CPU Utilization



(b) Memory Utilization

Figure 5. NLMS Prediction

B. Direct Resource Monitoring (Reference Method [7])

This approach directly measures CPU/memory utilization, which correlates clearly with traffic rate (Figure 8). It is reliable for benchmarking but does not capture per-switch variations or real-time dynamics.

C. Hybrid Load Estimation Model

Figure 10 compares the three methods over the range of 100–1100 pps. Direct monitoring accurately reflects aggregate controller resource utilization but provides no per-switch visibility, whereas PACKET-IN-based estimation captures switch activity but may underestimate the load generated by persistent flows. The hybrid model combines both information sources and therefore provides a more responsive and representative estimate of controller load, as further illustrated in Figure 9.

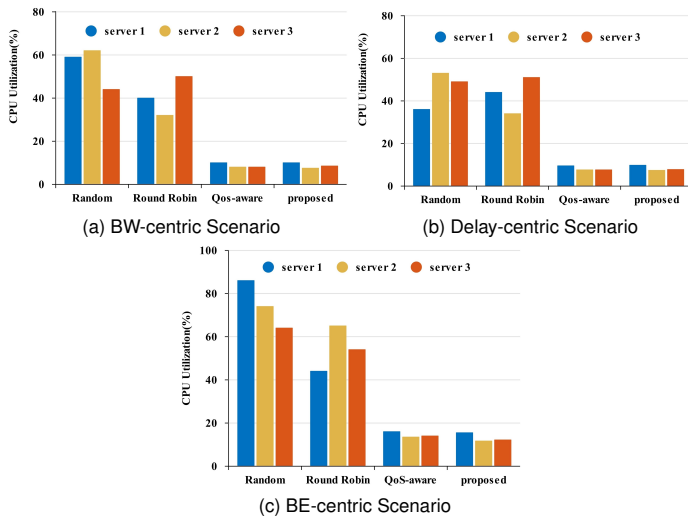


Figure 6. The CPU utilization of servers with different traffic classes in various methods (traffic of 500 pps)

Table 5
QoS Metrics under Different Traffic Loads

Rate	No BG		BG 300 pps		
	Loss (%)	Delay (ms)	Loss (%)	Delay (ms)	Jitt. (ms)
100	0.38	100	0.48	125.4	0.034
300	0.38	100.2	0.64	112.8	0.020
500	0.29	100	0.51	112.8	0.028
700	0.34	100.2	0.60	113.0	0.046
900	0.38	112.7	0.49	189.6	0.385

4.2.4 Voice QoE Evaluation under Network Traffic Load

Voice quality was evaluated under varying congestion using a primary flow of 100–900 pps, with and without 300 pps background traffic. Loss, jitter, and delay were measured using iPerf/ping, while the E-model calculated R-factor and MOS values for G.711, G.729, and Opus-WB (Tables 5 and 6).

Without background traffic, the network remained stable up to 700 pps (delay $\sim$ 100 ms, jitter $\sim$ 0.03 ms, and loss $\sim$ 0.3%), with delay reaching 112.7 ms at 900 pps. Voice quality remained high for G.711 (MOS $\sim$ 4.37, R-factor $\sim$ 91.4), Opus-WB (MOS $\sim$ 4.24), and G.729 (MOS $\sim$ 4.04).

With 300 pps background traffic, QoE remained acceptable up to 700 pps. At 900 pps, delay and jitter increased to 189.6 ms and 0.385 ms, reducing the R-factor of G.711 to $\sim$ 87.8 (Good) and G.729 to $\sim$ 76.9 (Fair). Overall, QoE remained stable at moderate loads ($\leq$ 700 pps) but declined under heavy loads. G.711 provided the highest quality, followed by Opus-WB, while G.729 remained intelligible until saturation.

4.2.5 Integrated Multi-Layer Load Balancing Performance

The proposed mechanism was evaluated on a three-domain topology (Fig. 3) under worst-case traffic imbalance: Domain 1 (500 pps, underloaded), Domain 2 (900 pps, normal), and Domain 3 ($2 \times 900 = 1800$ pps, overloaded).

Table 7 reports JFI values of at least 0.994 for all resources, indicating near-uniform intra-domain resource distribution. Despite carrying the highest traffic, Domain 3 reached equilibrium without inter-domain offloading, demonstrating the scheduler's effectiveness under heavy loads.

Table 8 presents the pre- and post-migration results. Migrating one heavily loaded switch from Domain 3 to Domain 1 increased the fairness indices by 0.135–0.163 (16–21%). Controller 3 CPU utilization decreased from 87% to 46% (–41 per-

Table 6
Voice QoE Metrics (MOS / R) under Different Traffic Loads

Rate (pps)	G.711		Opus-WB		G.729		Scenario
	MOS	R	MOS	R	MOS	R	
Without Background Traffic							
100	4.37	91.44	4.24	86.45	4.04	80.48	0 pps
300	4.37	91.43	4.24	86.46	4.04	80.48	0 pps
500	4.37	91.52	4.24	86.54	4.04	80.56	0 pps
700	4.37	91.47	4.24	86.49	4.04	80.51	0 pps
900	4.36	91.04	4.23	86.07	4.03	80.09	0 pps
With Background Traffic (300 pps)							
100	4.36	90.74	4.22	85.76	4.02	79.79	300 pps
300	4.36	90.89	4.23	85.92	4.02	79.96	300 pps
500	4.36	91.00	4.23	86.03	4.03	80.06	300 pps
700	4.36	90.93	4.23	85.96	4.02	79.99	300 pps
900	4.28	87.84	4.13	82.86	3.90	76.89	300 pps

Table 7
Server-level Load Balancing Metrics across Three Domains

Performance Metric	Domain 1	Domain 2	Domain 3
Fairness (Jain's index) – CPU	0.994	0.994	0.997
Fairness (Jain's index) – Memory	0.994	0.994	0.997
Fairness (Jain's index) – HDD	1.000	1.000	1.000
Fairness (Jain's index) – Bandwidth	0.994	0.994	0.997
Avg. CPU Utilization (%)	20.05	36.84	35.56
Avg. Memory Utilization (%)	18.78	34.64	33.41
Avg. Bandwidth Utilization (%)	28.24	51.80	49.34
Avg. HDD Utilization (%)	20.04	20.07	20.14

Table 8
Controller-level Summary: Before vs. After Switch Migration

Key Metric	Before LB	After LB	Change
CPU Fairness (Jain's index)	0.824	0.959	+0.135 ($\uparrow$ 16.4%)
Memory Fairness (Jain's index)	0.819	0.959	+0.140 ($\uparrow$ 17.1%)
Bandwidth Fairness (Jain's index)	0.792	0.955	+0.163 ($\uparrow$ 20.6%)
Controller 3 CPU Utilization (%)	87.0	46.0	–41.0 pp
Controller 1 CPU Utilization (%)	27.0	72.0	+45.0 pp

centage points), while Controller 1 increased from 27% to an operationally balanced 72% (+45 percentage points), retaining sufficient headroom. Controller 2 remained unchanged.

5 CONCLUSION

This study proposed a predictive and holistic framework for jointly balancing controller, server, and link loads in multi-domain SD-IoMT networks, addressing the limitations of decoupled, component-specific methods. By combining LSTM-based resource-load prediction with fuzzy logic for dynamic classification, the framework outperforms conventional approaches in operational efficiency. This coordinated strategy mitigates multi-component load fluctuations, achieves near-optimal resource fairness (Jain's index $\approx$ 1), and maintains high perceptual QoE under heavy traffic. Overall, the framework provides a scalable, intelligent, and adaptive solution for proactive SD-IoMT resource management. Future work will validate the framework using large-scale, real-world multimedia datasets and explore deep reinforcement learning (DRL) for self-optimizing, long-term control policies.

REFERENCES

- [1] S. A. Alvi, B. Afzal, G. A. Shah, L. Atzori, and W. Mahmood, "Internet of multimedia things: Vision and challenges," *Ad Hoc Networks*, vol. 33, pp. 87–111, 2015.
- [2] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of things: A survey on enabling technologies,

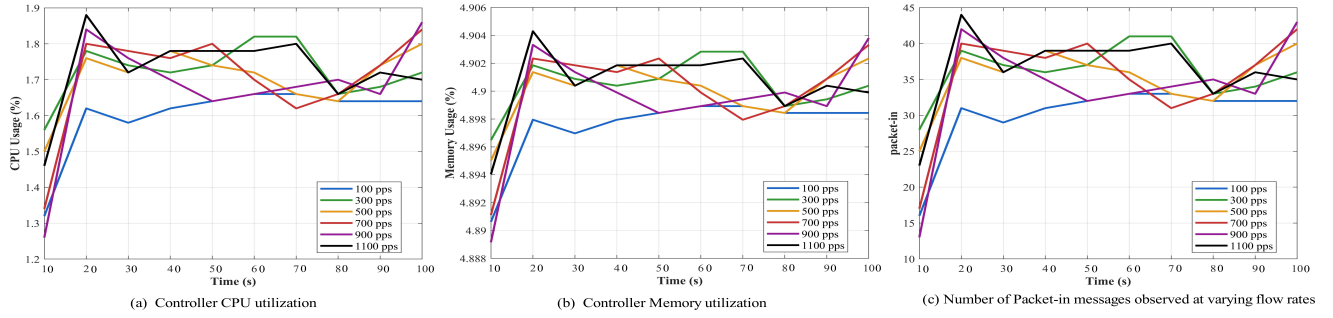


Figure 7. Controller resource behavior under varying PACKET-IN rates: (a) CPU utilization, (b) Memory utilization, and (c) frequency of PACKET-IN events.

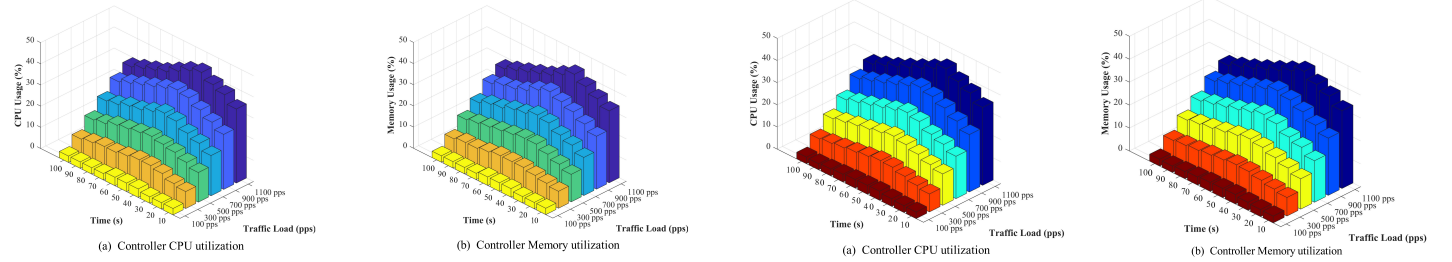


Figure 8. Controller resource utilization under varying flow rates: (a) CPU utilization, (b) Memory utilization

Figure 9. Controller resource utilization using the hybrid method across different flow rates: (a) CPU utilization, (b) Memory utilization

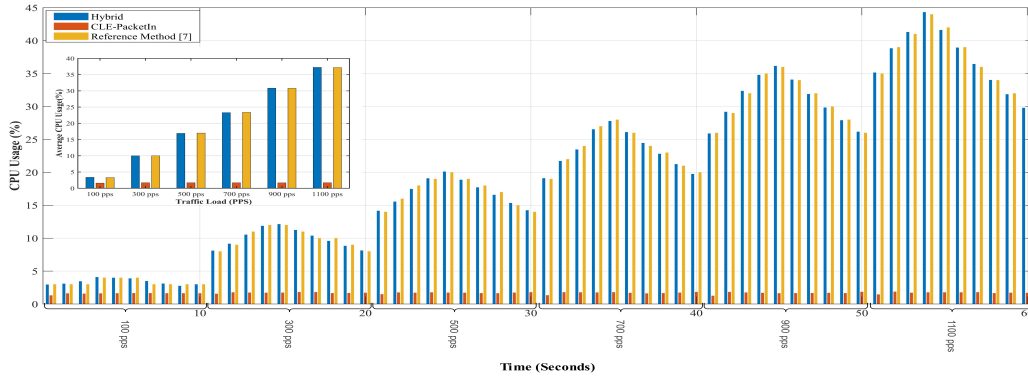


Figure 10. Controller load for six consecutive traffic rates (100–1100 pps). Each 10-second interval represents a rate, and the three load calculation methods (PACKET-IN, direct, hybrid) are plotted for comparison.

- protocols, and applications,” *IEEE communications surveys & tutorials*, vol. 17, no. 4, pp. 2347–2376, 2015.
- [3] A. Nauman, Y. A. Qadri, M. Amjad, Y. B. Zikria, M. K. Afzal, and S. W. Kim, “Multimedia internet of things: A comprehensive survey,” *Ieee Access*, vol. 8, pp. 8202–8250, 2020.
 - [4] M. Priyadarsini and P. Bera, “Software defined networking architecture, traffic management, security, and placement: A survey,” *Computer Networks*, vol. 192, p. 108047, 2021.
 - [5] S. K. Keshari, V. Kansal, and S. Kumar, “A systematic review of quality of services (qos) in software defined networking (sdn),” *Wireless Personal Communications*, vol. 116, no. 3, pp. 2593–2614, 2021.
 - [6] M. Hamdan, E. Hassan, A. Abdelaziz, A. Elhigazi, B. Mohammed, S. Khan, A. V. Vasilakos, and M. N. Marsono, “A comprehensive survey of load balancing techniques in software-defined network,” *Journal of Network and Computer Applications*, vol. 174, p. 102856, 2021.
 - [7] A. Montazerolghaem and M. H. Yaghmaee, “Load-balanced and qos-aware software-defined internet of things,” *IEEE Internet of Things Journal*, vol. 7, no. 4, pp. 3323–3337, 2020.
 - [8] A. Montazerolghaem, “Software-defined internet of multimedia things: Energy-efficient and load-balanced resource management,” *IEEE Internet of Things Journal*, vol. 9, no. 3, pp. 2432–2442, 2021.
 - [9] P. Abhishek, A. Naik, P. Doddannavar, R. Patil, M. M. Raikar, and S. Meena, “Load balancing for network resource management in software-defined networks,” in *Advances in Distributed Computing and Machine Learning: Proceedings of ICADCML 2022*. Springer, 2022, pp. 193–203.
 - [10] H. Zheng, J. Guo, Q. Zhou, Y. Peng, and Y. Chen, “Application of improved ant colony algorithm in load balancing of software-defined networks,” *The Journal of Supercomputing*, vol. 79, no. 7, pp. 7438–7460, 2023.
 - [11] Z. Guo, M. Su, Y. Xu, Z. Duan, L. Wang, S. Hui, and H. J. Chao, “Improving the performance of load balancing in software-defined networks through load variance-based synchronization,” *Computer Networks*, vol. 68, pp. 95–109, 2014.
 - [12] C. Li, K. Jiang, and Y. Luo, “Dynamic placement of multiple controllers based on sdn and allocation of computational resources based on heuristic ant colony algorithm,” *Knowledge-Based Systems*, vol. 241, p. 108330, 2022.
 - [13] H. Babbar, S. Rani, A. K. Bashir, and R. Nawaz, “Lbsmt: Load balancing switch migration algorithm for cooperative communication intelligent transportation systems,” *IEEE Transactions on Green Communications and Networking*, vol. 6, no. 3, pp. 1386–1395, 2022.
 - [14] H. Wang, H. Xu, L. Huang, J. Wang, and X. Yang, “Load-balancing routing in software defined networks with multiple controllers,” *Computer Networks*, vol. 141, pp. 82–91, 2018.