Load-balanced and QoS-aware Software-defined Internet of Things

Ahmadreza Montazerolghaem, Mohammad Hossein Yaghmaee

Load-balanced and QoS-aware Software-defined Internet of Things

Ahmadreza Montazerolghaem, *Student Member, IEEE,* Mohammad Hossein Yaghmaee, *Member, IEEE*

Abstract—Internet of Things (IoT) offers a variety of solutions to control industrial environments. The new generation of IoT consists of millions of machines generating huge traffic volumes; this challenges the network in achieving the Quality of Service (QoS) and avoiding overload. Diverse classes of applications in IoT are subject to specific QoS treatments. In addition, traffic should be distributed among IoT servers based on their available capacity. In this paper, we propose a novel framework based on Software-defined Networking (SDN) to fulfill the QoS requirements of various IoT services and to balance traffic between IoT servers simultaneously. At first, the problem is formulated as an Integer Linear Programming (ILP) model that is NP-hard. Then, a predictive and proactive heuristic mechanism based on time-series analysis and fuzzy logic is proposed. Afterwards, the proposed framework is implemented in a real testbed, which consists of the Open vSwitch, Floodlight controller, and Kaa servers. To evaluate the performance, various experiments are conducted under different scenarios. The results indicate the improved IoT QoS parameters, including throughput and delay, and illustrate the non-occurrence of overload on IoT servers in heavy traffic. Furthermore, the results show improved performance compared to similar methods.

Index Terms—IoT softwarization, Software-defined networking, Large-scale IoT network, Load balancing, Fuzzy logic.

1 Introduction

Internet of things has become one of the major and fascinating topics in the networking field over the last few years [1]–[5]. IoT gives access to information at any time and place. However, this causes the network to deal with a huge amount of traffic data. In the coming years, traffic volume as well as the number of IoT devices will grow exponentially. This issue is more critical in large-scale industrial IoT as the information and data management are more sensitive and vital in industries [6]. In IoT, a large number of heterogeneous devices are constantly exchanging heavy traffic from the network to servers [7]. In this case, an improper traffic distribution between the servers causes some of them to face resource scarcity and thus become overload. Moreover, since there are different traffic classes in IoT, existing mechanisms may not satisfy the necessary QoS requirements (e.g., bandwidth and delay) for this huge traffic volume [8], [9].

Fig. 1 illustrates an IoT abstract model. This model shows that the sensed or measured traffic in the infrastructure (the sensing layer) is conveyed to IoT servers via different paths in a communication network (the network layer). IoT servers offer different IoT applications and services as well (the application layer) [10]. IoT traffic can be grouped into three classes:

1. *Delay-centric* which is related to mission-critical or event-driven applications.
2. *Bandwidth-centric* which is associated with continuous traffic, for instance query-driven and real-time monitoring.
3. *Best-effort* which consists of general applications such as non-real time monitoring.

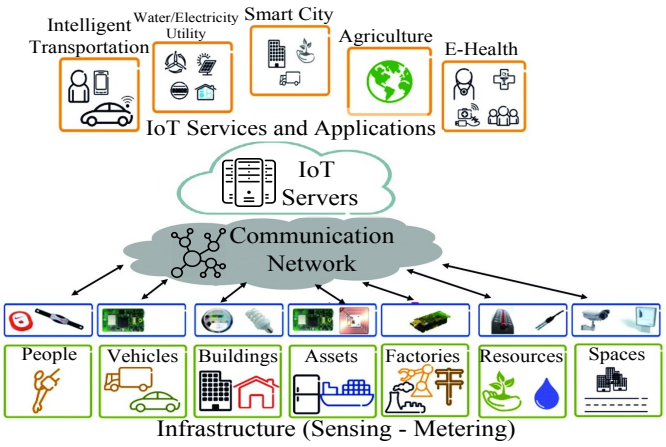


Figure 1. An IoT abstract model

Over the next few years, with an increase in the traffic volume collected and sent from the infrastructure to IoT servers, not only will IoT servers face overload, but also the communication network will fail to satisfy the QoS of traffic. In this paper, we have considered this issue as a best path selection problem, so that the selected path establishes the QoS of the IoT traffic classes, and balances the load of IoT servers (Fig. 2). The knowledge regarding available resources of servers, network topology, active paths, and QoS parameters of the network, such as link's bandwidth and delay are among the challenges of the problem.

With the emergence of Software-defined networking (SDN) technology, the entire network and its elements are viewed as a virtual network (global view), which is controlled using applications and APIs as well as a logically centralized controller. This causes the network resources (in-

• *A. Montazerolghaem is with the Department of Computer Engineering, Quchan University of Technology, Quchan, Khorasan Razavi, Iran. MH. Yaghmaee is with the Department of Computer Engineering, Ferdowsi University of Mashhad, Mashhad, Iran. E-mail: Ar.montazer@qiet.ac.ir, hyaghmae@um.ac.ir*

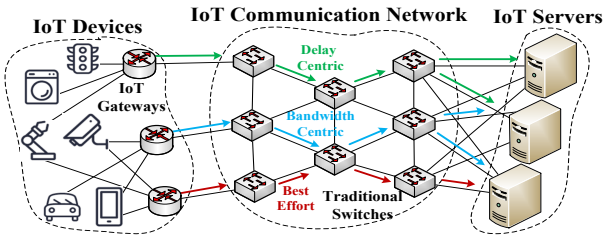


Figure 2. The best path selection problem to achieve the load balancing of IoT servers and to ensure QoS of IoT traffic

cluding servers) to be managed efficiently and on-demand, ensuring a more effective QoS. In fact, SDN is a dynamic, flexible, and manageable architecture that separates the control plane from the data plane. The data plane includes simple forwarding devices such as OpenFlow switches whereas the control plane consists of controller and various network applications such as routing, firewall, load balancing, monitoring [11]–[14]. The OpenFlow protocol [15] is also responsible for establishing communication between the two planes using a set of messages. In other words, collecting statistics or installing rules is performed using messages, such as `Features-request/reply`, `Packet-In` or `Flow-mod`. The statistics collected can be related to network topology, bandwidth and link delay, type of flows, etc.

1.1 Motivations

Why do we need a scalable IoT infrastructure? Scalability will be key to handling the explosive growth in the IoT. This means that IoT applications must have the ability to support an increasing number of connected devices, users, application features, and analytics capabilities, without any degradation in the QoS. Scalable IoT applications are also essential to monitoring, securing, and managing an increasing number of devices through a proportionate increase in the resources. The traffic and number of IoT devices will grow exponentially in the coming years. Knowing that the IoT infrastructure is a key service in next-generation network, it is vital to design and develop high performance and scalable communication infrastructure with QoS support. Moreover, for future large-scale IoT, it will be necessary to collect and transfer information from each sensor. With the overall modern IoT road map, both the number of sensors and the sampling frequency on a sensor will increase dramatically. Therefore, this will create a huge traffic load which should be efficiently routed and balanced across the network and IoT servers.

1.2 Contributions

Our study mainly aims at scalable managing the IoT traffic using SDN. In this regard, by applying SDN in IoT, we provide a resource and QoS conscious framework. In this framework, we seek to choose the best path among the existing paths for each of the IoT traffic classes in a way that the load balance of IoT servers is established, and the QoS of the traffic class is satisfied. Consequently, we propose a modular controller that uses fuzzy logic and the Normalised Least Mean Squares (NLMS) prediction system. We focus on proactive resource management that is based on predictions of future workload based on the past workload. The most important method in this area is time-series analysis. Time-series analysis could be used to detect repeating patterns in the workload or to estimate future values for resource allocation. In order to apply time-series analysis in resource management or load balancing, a certain metric will be periodically sampled at fixed intervals. The result will be a time-series containing a sequence of last observations. Time-series methods extrapolate this sequence to predict future values. Some of the methods used for this purpose are Moving Average, Autoregression, Autoregressive-Moving-Average (ARMA), exponential smoothing, and machine learning techniques [16]. We utilize the work of [17] that proposes a comparative analysis among prediction techniques, used for developing a dynamic prediction based the resource allocation strategy. Among these techniques, the NLMS predictor is the one providing the best trade-off between complexity, accuracy, and responsiveness.

To summarize, the main contributions of this article are:

- We design a novel SDN-based control and management framework for IoT,
- We define the problem of preventing overload occurrence on IoT servers along with increasing the QoS as an optimization problem,
- We prove NP-hardness of the defined optimization problem with limited resources,
- We provide a predictive and proactive heuristic method based on time-series analysis and fuzzy logic,
- We design modules as well as a flowchart of the proposed framework,
- We implement the proposed framework in a real testbed,
- We evaluate the performance of the implemented framework under different experiments and scenarios.

1.3 Organization

Our paper is organized as follows: Section 2 refers to the related work. The proposed model and framework are discussed in Section 3. Section 4 includes the implementation and performance evaluation of the framework. Finally, the conclusions and future work are provided in Section 5.

2 Related Work

Research communities investigate different aspects of IoT, and they provide surveys regarding IoT issues and challenges [18]–[20] that should be addressed such as security and privacy [21]–[23], architectural issue [19], big data [24], [25] as well as energy efficiency [26] and QoS [27]–[29]. This section discusses state-of-the-art research that uses SDN technology in IoT. Application of SDN technology in IoT has received considerable attention over the past few years.

In [30] a SDN-based data transfer security model for IoT is designed that attempts to improve the availability of secure applications, and to actively respond to network threats. The authors of [31] propose a centralized control system for IoT security (SDSec) and storage (SDStore) services. The authors discuss that SDSec can provide visibility for all the traffic flows in the network. As a result, it makes it possible to figure out suspicious traffic. The authors of [32] offer a software-defined wireless sensor network

framework (Soft-WSN) to support IoT service provisioning. Similarly, [33] introduces SDN-WISE that provides a flexible vendor-independent policy implementation in WSN for IoT application. SDN-WISE propounds APIs that allow software developers to implement the SDN controller. Wu et al. [34] offer UbiFlow, the software-defined IoT model for flow control and management of mobility. UbiFlow partitions the IoT network into the small network. Each network is administered by a physically distributed SDN controller. The IoT devices in each network could connect to different access points depending on their locations. In [35], kakiz et al. propose an IoT architecture based on SDN to overcome big data issues. Also, [36] focuses on analyzing big data in smart cities using SDN. An energy efficient routing in IoT through the use of a SDN controller is described in [37]. In [38], the authors propose a QoS routing algorithm for SDN-based IoT systems based on the simulated annealing algorithm. In addition, saha et al. [39] use the K-shortest paths algorithm in SDN controller to calculate a feasible routing path by taking into account the QoS requirements of each IoT packet.

While much prior research has suggested the potential benefits of applying SDN in computer networks in order to facilitate network management, there has only been few studies about the practical approaches of applying SDN in IoT [40]. Additionally, the whole concept of SD-IoT is in its infancy and standardization efforts in terms of framework, protocols, applications, and assessment tools are still underway. Also, as discussed earlier, the proposed ideas are mostly preliminary proposals about softwarization of WSNs; or they focus on security and big data challenges of IoT. Here, we pay particular attention to management of resources as well as QoS. To the best of our knowledge, there are no studies concerning a comprehensive approach for combining IoT server load balancing with IoT QoS mechanism. Therefore, the exploration of such an approach is timely and crucial, especially considering the rapid development of IoT applications and the emergence of SDN. In this paper, we propose a SDN-based architecture for IoT applications, so that both the path and server selection can be managed together to improve QoS for the IoT users, and to balance traffic between IoT servers simultaneously.

3 The Proposed Model and Framework

In this section, we introduce the mathematical system model and the novel SDN-based framework in a respective order.

3.1 The System Model

The notations used in this subsection are described in Table 1. Assume that an IoT communication network can be modeled as graph $G = (V, E)$, where V represents the set of switches, and E indicates the set of links. Let $|V| = v$ and $|E| = e$ be the number of switches and number of links, respectively. This graph connects l IoT gateways to z IoT servers (Fig. 2). IoT gateway is a bridge which connects between communication network (servers/switches) and sensor network (IoT devices). It is responsible for protocol conversion and data fusion of different sensor data. Each link $(i, j) \in E$ has a delay $d_{i,j}$ and bandwidth $b_{i,j}$. The resource consumption of each IoT server is also denoted by r. Here, ρ represents the remaining resources of each IoT server. Generally h is considered as a hop count. The hop count and resources are assumed to be non-negative. There are three types of IoT traffic on this network (delay-centric, bandwidth-centric, and best-effort). The goal is to find the best path from the source node s (the IoT gateway) to the target node t (the most appropriate IoT server) which satisfies the resource limits of the servers as well as QoS of traffic. In other words, the objective of the problem is balancing the load of IoT servers, and satisfying the QoS in order to efficiently use resources and consequently achieve low delay and high throughput. In this regard, to guarantee QoS requirements, we are seeking the shortest path for best-effort traffic, minimum-delay path for delay-centric traffic, and maximum-bandwidth path for bandwidth-centric traffic.

Table 1
Key Nomenclatures

V	Set of switches
E	Set of links
P	Set of all the routes from s to t
v	The number of switches
e	The number of links
l	The number of IoT gateways
z	The number of IoT servers
$d_{i,j}$	Delay of link $(i, j) \in E$
$b_{i,j}$	Bandwidth of link $(i, j) \in E$
r	The resource consumption of each IoT server
ρ	The remaining resources of each IoT server
h	Hop count
s	Source node (the IoT gateway)
t	Target node (the most appropriate IoT server)
q_p	QoS of path $p \in P$
r_p	Resource consumption of server on path $p \in P$
b_p	Bandwidth of path $p \in P$
d_p	Delay of path $p \in P$
h_p	Length of path $p \in P$
B_p	The maximum network bandwidth
D_p	The maximum acceptable delay
H_p	The maximum number of permissible hops
R_p	The maximum server resources

Prior to proposing the SDN-based heuristic approach, we prove that the problem of load-balancing and QoS-aware routing in the IoT network with resource limitation is an ILP problem and is, therefore, NP-hard.

Proposition 1. *The joint load-balance and QoS-aware routing problem in an IoT communication network with limited server resources is an ILP problem.*

Proof: Let P be the set of all the routes from s to t. For any path $p \in P$, a binary variable u_p is introduced. Also, q_p, r_p, and ρ_p are used to denote QoS, resource consumption, and remaining resources of p, respectively. The optimization problem is:

$$\textit{maximize} \sum\nolimits_p f_p u_p \tag{1}$$

subject to:

$$f_p = \sum\nolimits_p (\alpha q_p - \beta r_p), \forall p \in P \tag{2}$$

$$q_p = \gamma \frac{b_p}{B_p} - \delta \frac{d_p}{D_p} - \vartheta \frac{h_p}{H_p}, \ \forall p \in P \tag{3}$$

$$b_p \leq b_{i,j} u_p, \ (i,j) \in p, \forall p \in P \tag{4}$$

$$d_p = \sum\nolimits_{(i,j)\in p} d_{i,j} u_p, \ \forall p \in P \tag{5}$$

$$h_p = \sum\nolimits_{(i,j)\in p} |\{i,j\}| u_p, \ \forall p \in P \tag{6}$$

$$r_p \leq \frac{\rho_p}{R_p} u_p, \ \forall p \in P \tag{7}$$

$$b_p \leq B_p, \ \ d_p \leq D_p, \ \ h_p \leq H_p, \ \ \rho_p \leq R_p, \tag{8}$$

$$0 \leq q_p \leq 1, \ \ 0 \leq r_p \leq 1, \tag{9}$$

$$0 \leq \alpha \leq 1, \ \ 0 \leq \beta \leq 1, \tag{10}$$

$$\sum\nolimits_p u_p = 1, \tag{11}$$

$$\sum\nolimits_p (\gamma + \delta + \vartheta) u_p = 1, \tag{12}$$

Variables: $u_p \in \{0,1\}, \gamma, \delta, \vartheta \in \{0,1\}, q_p, r_p, b_p, d_p, h_p \geq 0$

This model seeks to find the path p, so that f_p is maximized (Eq. (1)). Here, f_p is a function of both the QoS and server consumable resources of the path p (Eq. (2)). The α and β coefficients provide a trade-off between QoS and resources. The objective is to achieve the highest weighted sum of QoS and resources. With f_p being maximized, a path can be explored that has the highest weighted sum of QoS and server resources. The q_p depends on the traffic class, while γ, δ and ϑ show which traffic is related to which of the three traffic classes (bandwidth-centric, delay-centric, and best-effort traffic, respectively). This model seeks to find the maximum-bandwidth, minimum-delay or shortest path by Eq. (3) and with respect to the traffic classes. Here, b_p , d_p and h_p represent the bandwidth, delay, and length of p, respectively (Eq. (4)-(6)). Eq. (7) ensures that the server consumable resources associated with the path p are less than its remaining resources. Moreover, B_p, D_p, H_p and R_p are the maximum network bandwidth, maximum acceptable delay, maximum number of permissible hops, and maximum server resources, respectively (Eq. (8)). The existence of these parameters causes the q_p and r_p values fall to between 0 and 1 (Eq. (9)), and therefore Eq. (2) normalizes. For this reason, α and β coefficients are between 0 and 1 as well (Eq. (10)). Eq. (11) guarantees that exactly one path is selected. Furthermore, each time the model is solved, the best path is obtained for one of the traffic classes (Eq. (12)). Although the objective function and constraints are linear, the binary variable u_p renders the model an ILP which is generally NP-hard, and cannot be solved in polynomial time [41]. To overcome this drawback, a SDN-based framework is proposed. ■

3.2 The SDN-based Heuristic Approach

To reduce time complexity, the problem is decomposed into two subproblems (server selection subproblem and path selection subproblem), which are solved in two phases.

In this regard, an SDN-based framework is proposed for the IoT communication network which considers the subproblems as SDN modules (select least-load iot server module and select QoS-aware route module) in the application plane. This framework benefits from SDN's ability to provide a global view of the entire network and thereby provides the route QoS and server load balancing. In this regard, the appropriate server is first selected, and then an appropriate path with respect to the selected server is found.

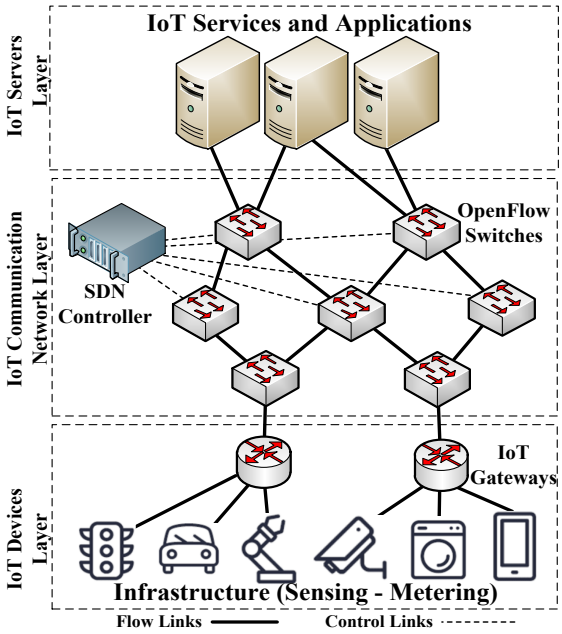


Figure 3. The proposed SDN-based IoT framework

3.2.1 The Proposed Framework

Our goal is to move towards the Software-defined IoT so that conscious traffic management is feasible for the IoT. To this end, we propose the framework, as presented in Fig. 3. The architecture of this framework is compatible with the ONF (Open Networking Foundation) SDN [42] and the IoT reference model (Fig. 1) consisting of three layers that interact with each other through a series of interfaces. The IoT device layer includes sensing and metering devices as well as IoT gateways. Traffic is aggregated by these IoT gateways. Accordingly, traffic is achieved through OpenFlow switches and flows into the layer of IoT servers by managing the SDN controller. The controller has a global view of resources, traffic, and the IoT network. Thereby, an appropriate path and server for each traffic class are considered with the help of OpenFlow protocol. Therefore, it provides control, management, scalability, and flexibility for large-scale IoTs.

Fig. 4 shows a general example of how the IoT traffic management is governed by the controller. Upon receiving the first message from any traffic class, the OpenFlow switch encapsulates it within a `Packet-In` message, and then sends it to the controller to determine the appropriate path and server. The controller installs the response with the `Flow-mod` message on track switches. Then, the traffic is exchanged between the IoT gateway and IoT server. Thus, effective traffic routing is realized. Now, the main remaining issue is the architecture as well as the performance of the controller, which are discussed in the following subsections.

3.2.2 The Proposed Controller

Fig. 5 shows the architecture of the proposed controller. It builds a clear distinction between data logic and management based on the modular structure. The IoT traffic and two types of management datagrams flow into the data and control planes (sFlow and OpenFlow), respectively. The *Network Statistics Measurement* module is capable of collecting detailed data from the topology state, traffic flows, switches, and links using OpenFlow messages. The *IoT Traffic Classification* module categorizes the IoT traffic into three classes. The sFlow protocol [43] provides the consumable resources statistics of IoT servers for the controller using the *sFlow Agent* (embedded within the servers) and the *sFlow Collector*. The CPU and memory are the most important of these resources. In other words, the servers load status is sent to the *Collector* module by the *sFlow Agent* via the sFlow protocol. The *sFlow Agent* uses a sampling method to capture CPU and memory statistics from the server. *sFlow Datagrams* are used to forward the sampled statistics to a *sFlow Collector* in order to store in the database for analysis purpose.

In order to react faster to CPU and memory variations, the controller has to be predictive-driven, and for this reason we have used time-series analysis. The NLMS algorithm [44] is one of the best options in this regard, since it can create a compromise between complexity, accuracy, and responsiveness [17]. The *NLMS Prediction System* module is then applied to analyze and predict future CPU and memory values of the servers. This module uses historical data collected in the database (the details of this module are given in the subsection 3.2.2.1). Due to the equitable and proactive distribution of load between IoT servers, each IoT server is provided with a load window. With respect to the *NLMS* module output, the *Fuzzy System* module set the size of these windows. The greater the size of a server load window, the lesser the load (the details of this module are described in the subsection 3.2.2.2). Finally, the *Select Least-load IoT Server* module chooses a server with the maximum available resources. This server has the largest load window size. Then, the *Select QoS-aware Route* module finds the path with the desired QoS of the classified traffic for achieving the selected server.

The flowchart of the controller is shown in Fig. 6. Time is divided into τ intervals. For the time-series analysis,

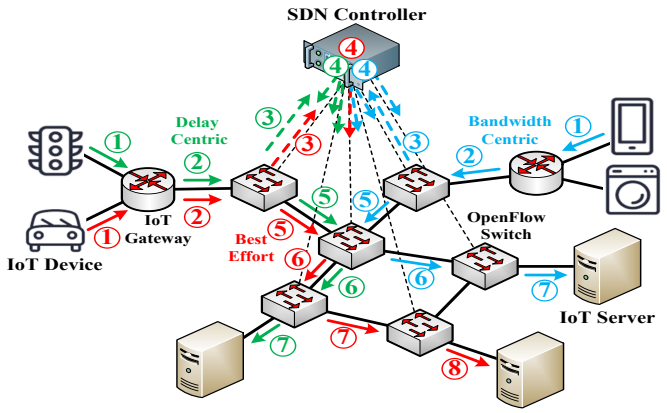


Figure 4. An example of software-defined IoT traffic management

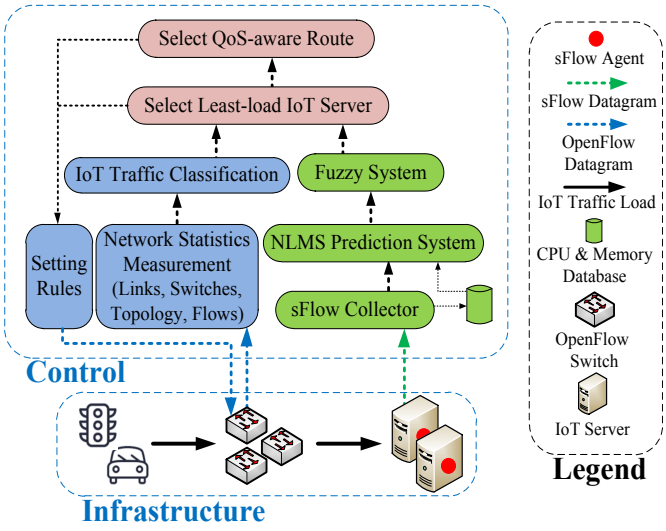


Figure 5. The architecture of the proposed controller

the server resources should be sampled at specific time periods. The result is a time series containing a series of last observations. Here, c_τ^k and m_τ^k represent the values sampled by the *sFlow Agent* from the CPU and memory of the server k at the current τ, respectively (box 1). These values are stored as historical data in the database (box 2). So, there are two vectors $X_\tau^k(CPU) = [c_\tau^k, c_{\tau-1}^k ..., c_{\tau-\phi+1}^k]$ and $X_\tau^k(Memory) = [m_\tau^k, m_{\tau-1}^k, ..., m_{\tau-\phi+1}^k]$ containing ϕ of the previous observations for each server k available. As a result, the estimated value of c^k and m^k for the next τ (i.e., $\widehat{c}_{\tau+1}^k$ and $\widehat{m}_{\tau+1}^k$) is obtained by the *NLMS Prediction System* (box 3). $W_{\tau+1}^k$ is the window size of the server k in the next τ, and $\omega_{\tau+1}^k$ is its rate of variations. The *Fuzzy System* is responsible for determining $\omega_{\tau+1}^k$ with respect to $\widehat{c}_{\tau+1}^k$, and $\widehat{m}_{\tau+1}^k$ (box 4). $W_{\tau+1}^k$ is then obtained from the recurrence relation $W_\tau^k + (\omega_{\tau+1}^k \times W_\tau^k)$, and therefore the vector $[W_{\tau+1}^1, W_{\tau+1}^2, ..., W_{\tau+1}^z]$ for the whole server z is available (box 5). Among these servers, a server with a higher $W_{\tau+1}$ is selected as the least-loaded server (first box 6). At the same time, the topology, traffic type, and link statistics are identified (second box 6). All possible paths to the selected server are obtained as the set P (box 7). Now, the bandwidth and delay of candidate path are calculated, and then maintained in the two arrays $[b_1, b_2, ..., b_p]$ and $[d_1, d_2, ..., d_p]$ (first box 8). The equation $b_p = min\{b_{i,j} | (i,j) \in p, p \in P\}$ explains the bandwidth measurement in the path, where b_p is the available bandwidth for the path p, while $b_{i,j}$ is the available bandwidth for any link (i,j) in the path. Available link bandwidth is calculated based on the link utilization rate $u_{i,j}$ and maximum possible link capacity $B_{i,j}$ through the equation $b_{i,j} = B_{i,j} \times (1 - u_{i,j})$. The equation $d_p = \sum_{(i,j)\in p} d_{i,j}$ formulates the delay in the path p where d_p refers to the path delay and $d_{i,j}$ indicates the delay of each link (i,j), of which the path p is composed. The IoT traffic class is then specified (second box 8), and accordingly the appropriate path is selected (box 9). Finally, suitable rules are developed (box 10).

3.2.2.1 The proposed NLMS system for the load prediction of IoT servers: This system can detect patterns of

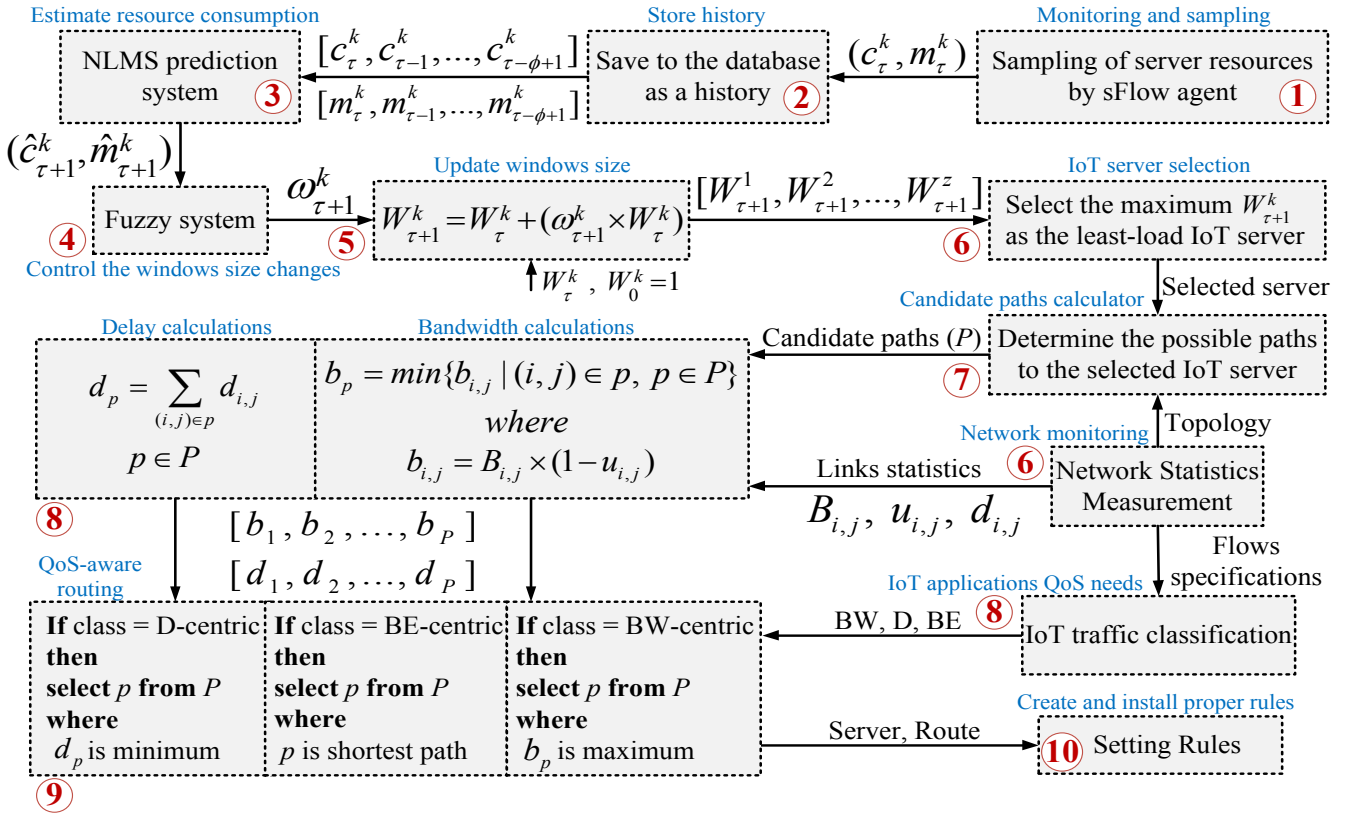


Figure 6. The flowchart of the proposed controller

CPU and memory consumption, and is also able to estimate their future values. The details are illustrated in Fig. 7. The input of the system consists of the two vectors $X_\tau^k(CPU)$ and $X_\tau^k(memory)$, seeking to minimize the mean-square error. The output of this system includes $\hat{c}_{\tau+1}^k$ and $\hat{m}_{\tau+1}^k$, which are obtained based on the equations (13) and (14) as well as the two filters g_τ and y_τ:

$$\hat{c}_{\tau+1}^k = g_\tau X_\tau^{kT}(CPU) \tag{13}$$

$$\hat{m}_{\tau+1}^k = y_\tau X_\tau^{kT}(memory) \tag{14}$$

For each new data, filters are updated based on the following recurrence equations:

$$g_\tau = g_{\tau-1} + \mu \frac{\varepsilon_{\tau-1}^k(c) X_{\tau-1}^k(CPU)}{\| X_{\tau-1}^k(CPU) \|^2} \tag{15}$$

$$y_\tau = y_{\tau-1} + \mu \frac{\varepsilon_{\tau-1}^k(m) X_{\tau-1}^k(memory)}{\| X_{\tau-1}^k(memory) \|^2} \tag{16}$$

Moreover, errors are obtained from the following equations:

$$\varepsilon_\tau^k(c) = c_{\tau+1}^k - \hat{c}_{\tau+1}^k \tag{17}$$

$$\varepsilon_\tau^k(m) = m_{\tau+1}^k - \hat{m}_{\tau+1}^k \tag{18}$$

Note that y_0 and g_0 are initialized (usually with the value of 0). Moreover, μ is a constant parameter called the step size ($0 < \mu < 2$).

3.2.2.2 The proposed fuzzy system for controlling the load window size of IoT servers: As mentioned before,

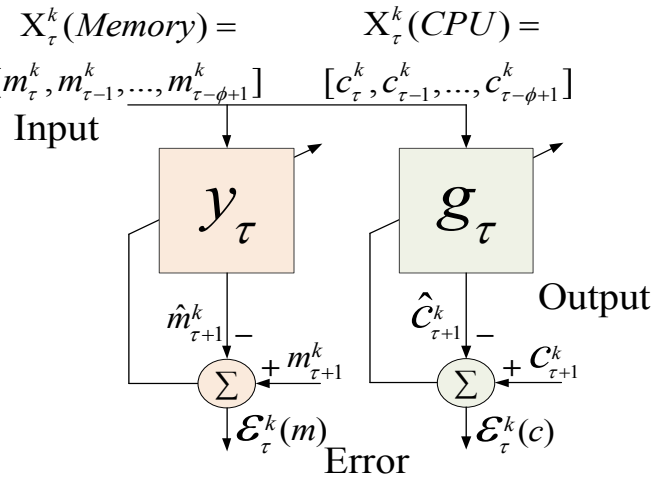


Figure 7. The proposed NLMS system for resource estimation

in this paper, a window-based load-balancing method is presented. In this method, a window is used to limit traffic sent to the servers. We use fuzzy logic to pinpoint the window size. The proposed *Fuzzy System* is shown in Fig. 8. As can be observed in Fig. 8, in this method, a fuzzy controller is embedded in order to dynamically change the window size. The fuzzy controller uses $\hat{c}_{\tau+1}^k$ and $\hat{m}_{\tau+1}^k$ as the input, and $\omega_{\tau+1}^k$ as the output.

Their membership functions are fine-tuned using expert knowledge as well as repeated experiments (Fig. 9). The system performance depends strongly on the membership

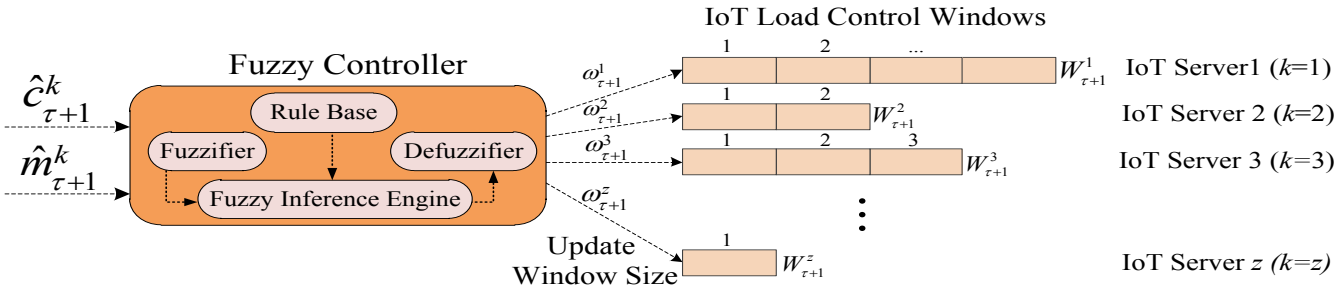
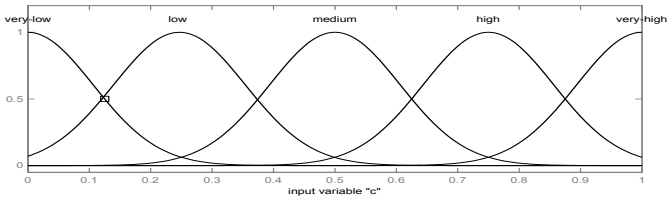
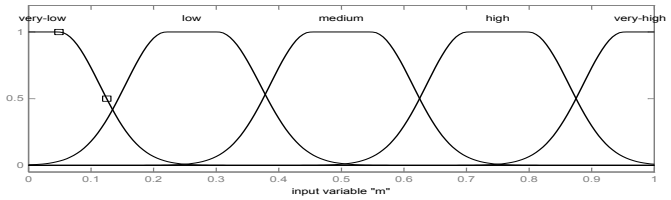


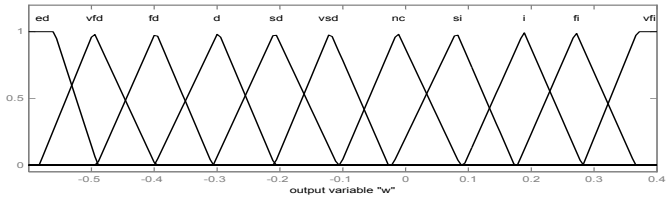
Figure 8. The proposed fuzzy system for size control of load windows



(a) The input membership function for $\widehat{c}^k_{\tau+1}$



(b) The input membership function for $\widehat{m}^k_{\tau+1}$



(c) The output membership function for $\omega^k_{\tau+1}$

Figure 9. Membership functions

functions. The membership functions in Fig. 9 are curves that define how each point in the input space is mapped to a membership value (degree of membership). Also, this figure shows that gaussian membership function is used for $\widehat{c}^k_{\tau+1}$ and $\widehat{m}^k_{\tau+1}$, and triangular membership function is used for $w^k_{\tau+1}$. In this regard, this figure illustrates the range of changes for $\omega^k_{\tau+1}$ is $[-0.6, 0.4]$ and for $\widehat{c}^k_{\tau+1}$ and $\widehat{m}^k_{\tau+1}$ are [0,1].

In addition, the fuzzy inference method is Mamdan, while the Center of Gravity (CoG) is used for defuzzification.

In the next step of the fuzzy system design process, the fuzzy rules base is defined (Table 2). The rules are constructed to indicate that greater consumable resources cause higher load; therefore, the size of the window should be smaller. The opposite of this is true as well.

4 Implementation and Performance Evaluation

4.1 The Implementation Testbed

In this study, a testbed is provided based on the *Kaa IoT platform* [45] in order to evaluate the efficiency of the proposed framework (Fig. 10). Kaa is a flexible open source platform for building and managing applications in IoT. Kaa IoT platform consists of *Kaa servers* and *endpoint SDKs*. Kaa server implements the back-end part of the platform, and uses Apache ZooKeeper for services coordination. It is comprised of the control, operations, and bootstrap services. An endpoint SDK is a library, which can be used to create clients [45].

Fig. 11 illustrates the testbed, which its topology includes three Kaa servers, two Kaa endpoint SDKs, seven OpenFlow switches, and one SDN controller. Further, Kaa servers are *Kaa Sandbox* [45]. Kaa Sandbox is a Kaa instance, which is presented as a stand-alone virtual machine. Kaa uses a random basic mechanism to distribute loads between servers. We use *Open vSwitch v2.4.1* [46] and *Floodlight v1.2* [47] to implement OpenFlow switches and controller, respectively. The Floodlight SDN controller is a multi-threaded Java-based controller that uses Netty framework [48]. Open vSwitch is a software implementation of a virtual network switch, which supports OpenFlow protocol.

The requirements for the implementation of the proposed method is given in Table 3. The Floodlight controller is run on an *HP G9 DL580* hardware. In addition, Kaa servers are run on three *Quantic servers (QSR model)*. The background traffic is generated by using the *iPerf*

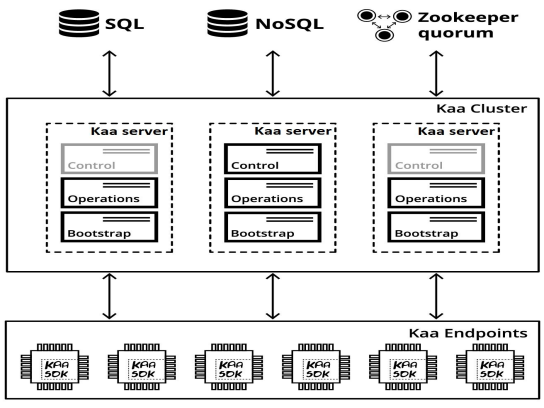


Figure 10. Basic Kaa IoT platform [45]

Table 2
Fuzzy Rules Base ($\omega^i_{\tau+1}$ changes)

$\widehat{m}^i_{\tau+1}$ \ $\widehat{c}^i_{\tau+1}$	very low	low	medium	high	very high
very low	very fast increase (vfi)	fast increase (fi)	increase (i)	very slow decrease (vsd)	slow decrease (sd)
low	fast increase (fi)	increase (i)	slow increase (si)	slow decrease (sd)	decrease (d)
medium	increase (i)	slow increase (si)	no change (nc)	decrease (d)	fast decrease (fd)
high	very slow decrease (vsd)	slow decrease (sd)	decrease (d)	fast decrease (fd)	very fast decrease (vfd)
very high	slow decrease (sd)	decrease (d)	fast decrease (fd)	very fast decrease (vfd)	extremely decrease (ed)

Table 3
Testbed Characteristics for the Experiments

	IoT Servers	SDN Controller	OpenFlow Switches	IoT Gateways	Background Traffic Generator
Software	Kaa Sandbox	Floodlight v1.2	Open vSwitch v2.4.1	Endpoint SDKs	iPerf
Quantity	3	1	7	2	1
CPU	Dual Core 1.8GHz	Xeon E5645 2.4GHz*	Dual Core 1.8GHz	Xeon E5645 2.4GHz	Dual Core 1.8GHz
RAM	2 GB	4 GB	1 GB	4 GB	2 GB
Operating System	CentOS v7.2	Linux kernel v3.10	Linux kernel v3.10	CentOS v7.2	Red Hat v6
	* 6 cores - 12 threads				

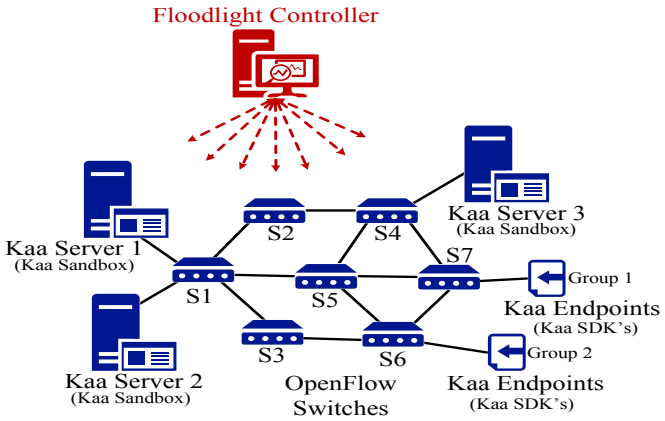


Figure 11. The testbed of the proposed framework

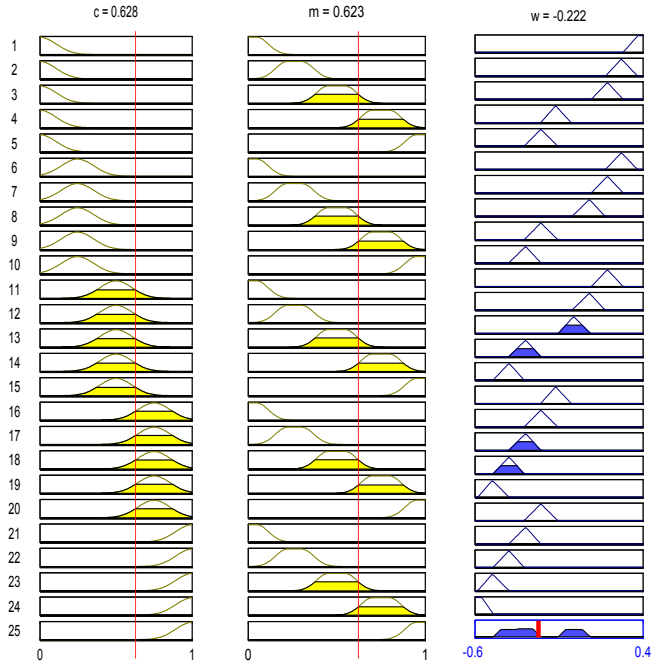
Table 4
Key Setup for Experiments (Properties and Parameters)

Parameter	Properties
Number of iterations	3 times for each test
ϕ (filter order)	30
μ (step size)	0.8
Bandwidth of each link	10 Mbps
$\omega^k_{\tau+1}$ (window variations)	$[-0.6, 0.4]$
$c^k_{\tau+1}$ (sample of CPU usage)	$[0, 1]$
$m^k_{\tau+1}$ (sample of memory usage)	$[0, 1]$
W^k_0 (window size in $\tau = 0$)	1
Traffic load (total)	100-900 mps
Background load	100-400 mps
y_0 (memory filter with $\tau = 0$)	0
g_0 (CPU filter with $\tau = 0$)	0

[49], which sends packets at a constant rate. The designed systems and modules are implemented on the controller. Each test is run for three times, and their mean value is considered as the result. Further, ϕ and μ are considered to have the values of 30 and 0.8, respectively, as based on the performed results analysis, these values provide satisfactory efficiency. Several experiments are conducted to tune these parameters. Also, the bandwidth of each link is 10 Mbps. Table 4 illustrates the details of the network configuration used to perform the experiments.

The Load Balancing by Server Response Time (LBBSRT) [50], History Window Weighted Average Response Time (HWAR) [51], and Transaction Least-Work-Left (TLWL) [52] methods are the most important comparison algorithms in this regard. LBBSRT uses the controller to obtain the response time of each server in order to select a server with minimum the response time. In this regard, the controller sends multiple `Packet-out` messages to the switches with time interval t and records the transmission time. HWAR is based on the history of the server's response time, which is used to estimate the load being currently processed on servers. HWAR selects a server with the smallest amount of this history. It has an infinite memory capable of recording an immense number of response time values (keeps track of the whole history). TLWL selects a server that has the least work, where work (i.e., load) is based on the relative estimates of transaction costs. In other words, TLWL estimates the server load based on the weighted average of transactions that are currently handled by a server. In this regard, this method maintains a set of counters includes the weighted number of transactions assigned to each server. A new request is allocated to the server with the lowest counter.

Figs. 12 and 13 illustrate the relationship between the input and the output values of the proposed fuzzy system. By following these figures, the relations between two inputs ($\widehat{c}^k_{\tau+1}$, $\widehat{m}^k_{\tau+1}$) and output ($\omega^k_{\tau+1}$) can be obtained. Based on the performed evaluations, the best range of changes for $\omega^k_{\tau+1}$ is the interval of $[-0.6, 0.4]$. The proposed fuzzy rule base has 25 rules (Table 2). Fig. 12 shows the *MATLAB Rule Viewer*. On the left side, the rules are numbered 1 through 25. These rows of rules apply across each column representing the variables. The leftmost 2 columns are the inputs and below them, the corresponding values chosen for each. The



(a) $\widehat{c}^k_{\tau+1}$ (b) $\widehat{m}^k_{\tau+1}$ (c) $\omega^k_{\tau+1}$

Figure 12. The fuzzy system output for $[0.628, 0.623]$

rightmost column is the output and at the bottom right is the end result. For example, for the input values $\widehat{c}^k_{\tau+1} = 0.628$ and $\widehat{m}^k_{\tau+1} = 0.623$, the system considers the value -0.222 for $\omega^k_{\tau+1}$, indicating that the window size of the server k has to be reduced by 0.222 ($0 < \widehat{c}^k_{\tau+1}, \widehat{m}^k_{\tau+1} < 1$).

Fig. 13 shows how $\omega^k_{\tau+1}$ is modified by $\widehat{c}^k_{\tau+1}$ and $\widehat{m}^k_{\tau+1}$. With an increase in $\widehat{c}^k_{\tau+1}$ and $\widehat{m}^k_{\tau+1}$, the window size of the server k is reduced. The interpretation of this figure is that with increasing consumption of CPU and memory of server k ($\widehat{c}^k_{\tau+1}$, $\widehat{m}^k_{\tau+1}$), the size of the corresponding window decreases ($\omega^k_{\tau+1}$). This will send less load to the server k and prevent overload. The reverse is also true. If the resource consumption of the server k decreases, then the size of the window will increase, thus sending more load to the server k.

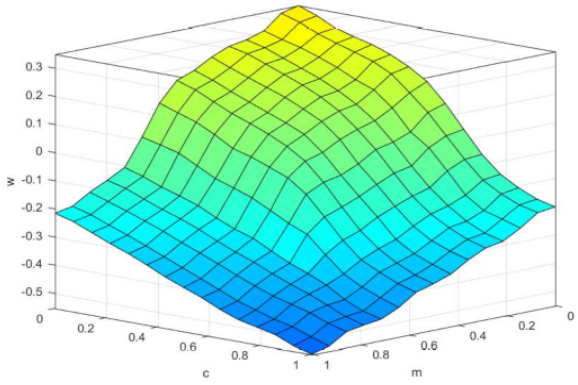


Figure 13. $\omega^k_{\tau+1}$ versus $\widehat{c}^k_{\tau+1}$ and $\widehat{m}^k_{\tau+1}$

4.2 The Experimental Results

In this section, the results of seven experiments are presented. The purpose of the first experiment is to compare the proposed method with other methods in terms of throughput and delay. The second experiment examines the proposed method in different traffic class scenarios. In this experiment, four scenarios are designed according to the traffic classes. In the third experiment, we examine more closely the traffic classes. In this regard, we carefully examine three flows of each traffic class in terms of path bandwidth and delay. Experiment 4 examines how load distribution is performed in the proposed method despite background traffic? The fifth experiment discusses the fairness of the proposed method in distributing the load between servers. Experiment 6 shows how the failure of the switches affects the quality of service. Finally, the seventh experiment analyses the designed controller.

4.2.1 First Experiment: comparison with other mechanisms

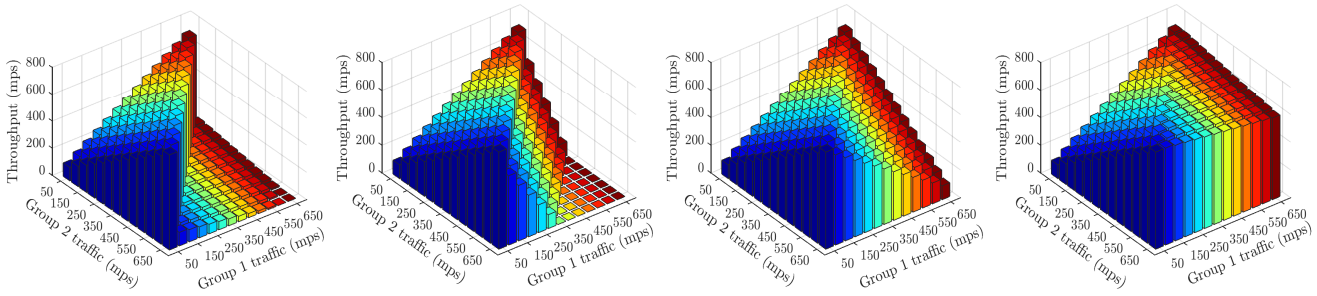
Regardless of the traffic class, two groups of Kaa endpoints are used to inject the load on the network in the first experiment (the Kaa endpoints send traffic to the Kaa cluster). Each of these groups generates traffic from 50 to 650 messages per second (mps). Figs. 14 and 15 demonstrate the throughput and response time of servers. Based on the observations, when there is no control for distributing traffic between the Kaa servers, the throughput decreases after the total traffic reaches 700 mps (Fig. 14a), indicating that the capacity of the whole servers is approximately 700 mps. After this threshold, due to the lack of an overload control mechanism, the servers are overloaded, and as a result, the throughput is dropped sharply. This trend continues by increasing the traffic volume until the throughput becomes zero. As shown in Fig. 15a, the servers' response time is almost zero before being overloaded; then, it heavily increases and reaches about 1000 ms.

Figs. 14b and 15b represent the basic function of Kaa. As mentioned above, the distribution of the load between the Kaa servers is random in this method, resulting in a relative improvement in performance compared to the method with no control. Accordingly, the drop in throughput and the increase in response time occurs in this method with less intensity. The LBBSRT mechanism uses the servers' response time to estimate their load. Although this criterion is not as accurate as the estimation of resources available to servers, it increases the throughput and reduces the response time compared to the two previous methods (Figs. 14c and 15c).

Finally, Figs. 14d and 15d suggest a good performance of the framework proposed in this paper to achieve high throughput and low response time. This framework has been able to provide a precise estimate of server load by proactive resource management, and thus, prevented the servers from overloading and its negative consequences by appropriate load distribution. In this method, the minimum throughput is about 500 mps, while the maximum response time is less than 200 ms.

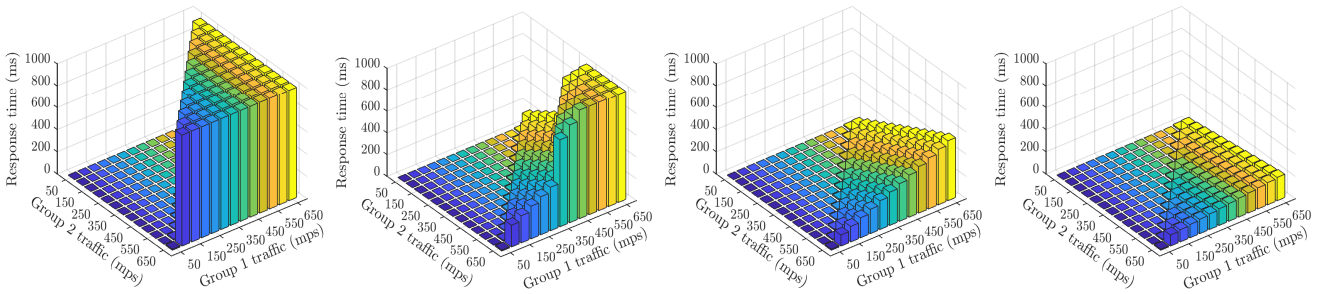
4.2.2 Second Experiment: QoS and load balancing in different scenarios

In the second experiment, regardless of groups generating traffic, the traffic is injected into the network with different



(a) No control. (b) Basic Kaa mechanism. (c) LBBSRT mechanism. (d) Proposed mechanism.

Figure 14. Servers' throughput by increasing traffic



(a) No control. (b) Basic Kaa mechanism. (c) LBBSRT mechanism. (d) Proposed mechanism.

Figure 15. Servers' response time by increasing traffic

classes (four scenarios). Figs. 16 and 17 show the throughput and average delay as QoS evaluation criteria. Furthermore, Figs. 18 and 19 illustrate the CPU utilization as an evaluation criterion for loads balance between servers.

The throughput of all methods in the BW and D-centric scenarios is almost higher than that of the BE-centric scenario, as the shortest path is not usually the highest quality path (Fig. 16). For this reason, the average delay in the BE-centric scenario is higher than that of the rest of the scenarios (Fig. 17). The results of the combined scenario are between the results of three other scenarios, while the results of the BW and D-centric scenarios are almost identical. However, the throughput of methods in the BW-centric scenario is slightly more, and the methods' delay is slightly lower in the D-centric scenario.

By increasing the traffic to up to 700 mps, the throughput increases in all methods and then, decreases in traffic of 900 mps, except for the method suggested in this paper. The proposed method has the maximum throughput in all scenarios even in heavy traffic at 900 mps. Then, HWAR, TLWL, and LBBSRT methods have the most throughput, respectively. The reason for higher throughput of HWAR algorithm compared to other two algorithms is that this algorithm considers the history response times rather than the last response time. The lowest throughput belongs to Round-Robin and Random methods (Fig. 16). In the Round-Robin (RR) algorithm, if the previous packet was assigned to server M, the next message is allocated to server $(M+1) mod N$, where N is again the number of servers in the cluster.

The methods can be categorized in terms of delay as follows (Fig. 17):

- First category: the least delay including the proposed method;
- Second category: the moderate delay including methods of HWAR, TLWL, and LBBSRT;
- Third category: the highest delay including Round-Robin and Random methods.

Note that the delay of the methods in the second and third categories has a cubic growth at 900 mps.

Figs. 18 and 19 illustrate the load balance between the three Kaa servers by different methods before (traffic of 500 mps) and after overloading (traffic of 900 mps). As observed, the CPU utilization of all scenarios is higher in traffic 900 mps. The resources of some Kaa servers are saturated even in this traffic and in the BE-centric scenario. For example, the Kaa server 3 and Kaa server 1 are saturated in the RR and Random methods, respectively (Fig. 19c). In the BE-centric scenario, the CPU utilization is more than that of other scenarios, and even the load balance between servers fails to operate well because these types of traffic are inherently looking for the nearest server without considering the resources. In the proposed method, the CPU utilization of the three servers is very small and close to each other. The Random method has the worst load distribution due to its lack of knowledge about server resources. The load distribution in TLWL, HWAR, and LBBSRT methods becomes worse in traffic of 900 mps compared to the traffic of 500 mps. However, the proposed method performs better in the load balance independent of traffic and scenario.

4.2.3 *Third experiment: QoS of the IoT traffic classes*

In the third experiment, we investigated the flows in the proposed method in detail. In this regard, three sample

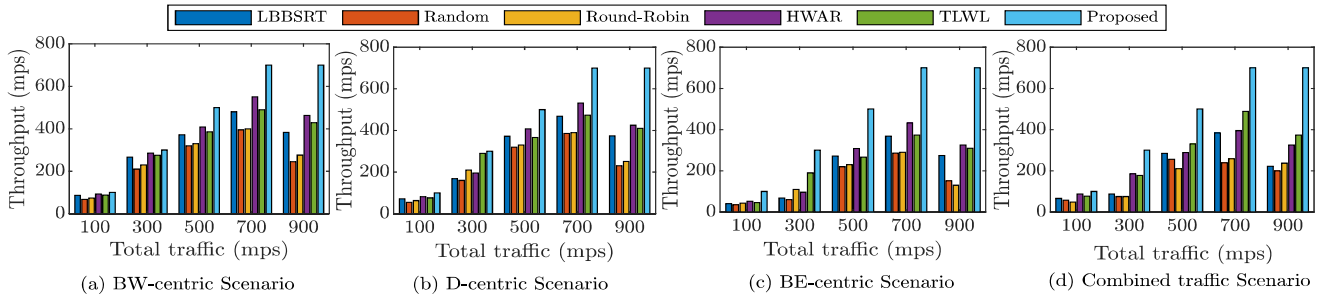


Figure 16. The throughput of different traffic classes in different methods

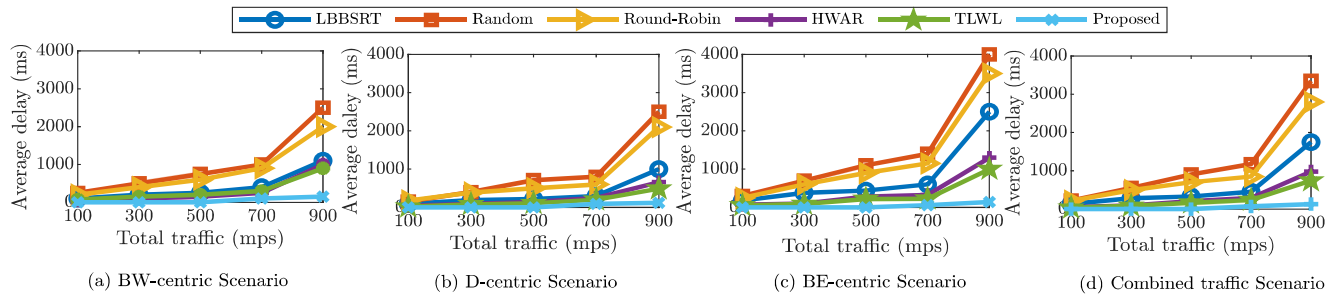


Figure 17. The average delay of different traffic classes in different methods

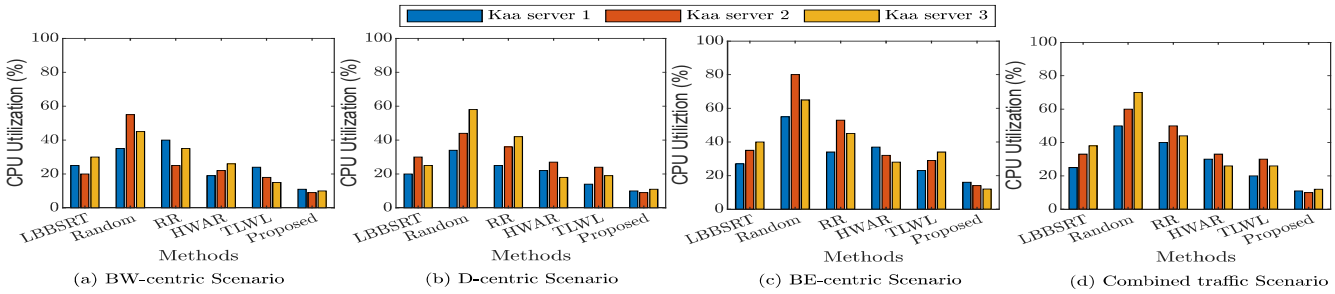


Figure 18. The CPU utilization of servers with different traffic classes in various methods (traffic of 500 mps)

flows (F1, F2, and F3) are considered from three traffic classes in 100 seconds. Fig. 20 depicts path bandwidth and delay of these flows. As illustrated in Fig. 20, the proposed method has considered paths for F1-D to F3-D with a small delay (Fig. 20a). In addition, the paths considered for F1-BW to F3-BW have the highest bandwidth (Fig. 20b). F1-BE to F3-BE paths have a moderate delay and bandwidth. This suggests that the proposed framework has been able to provide QoS for different flows of IoT, as a result of using the SDN technology.

Fig. 21 shows the consumption resources of servers and the total throughput of flows in each traffic class. The consumed CPU and memory in all three classes are very low and almost equal ($\sim 21\%$). The throughput of all three classes is also equal and very close to the offered load, as the controller decides for each traffic class according to its nature (Fig. 20).

4.2.4 *Fourth Experiment: load balancing in unequal conditions*

The fourth experiment involves two scenarios with different background traffic. In Scenario 1, the background traffic of all three Kaa servers is 100 mps while it is 100 mps, 200 mps, and 400 mps for servers 1 to 3 in scenario 2, respectively. The constant traffic of 900 mps is injected into the network for 100 seconds.

Fig. 22 represents the service rate for the Kaa servers. In Scenario 1, since the background traffic of all three servers is equal, then, the service rate of all three servers is equal to approximately 300 mps (Fig. 22a). In other words, the traffic of 900 mps is equally distributed between the three servers. In Scenario 2, the service rate of server 1 is roughly 512 mps, which is the twice of the service rate of server 2 (256 mps) (Fig. 22b), because the background traffic and the consumption resources of server 2 are twice of the server 1. Therefore, in the proposed framework, each server serves according to its background traffic. This experiment indicates that the controller is completely aware of the resources available to the servers.

4.2.5 *Fifth Experiment: fairness analysis*

The fifth experiment is conducted to verify the fairness of the proposed method. Therefore, two groups of Kaa endpoints inject traffic in four 250-second intervals. Fig. 23 indicates the server's throughput in 1000 seconds with total traffic of 700 mps. In the interval of [0, 250] sec, only group

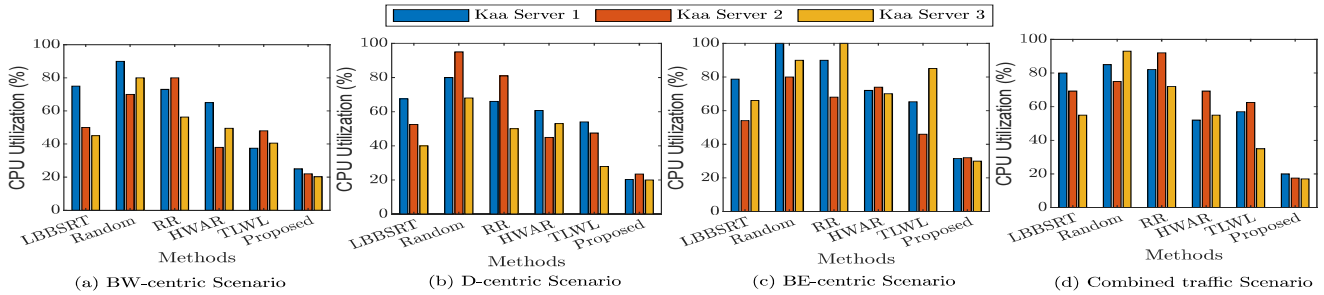


Figure 19. The CPU utilization of servers with different traffic classes in various methods (traffic of 900 mps)

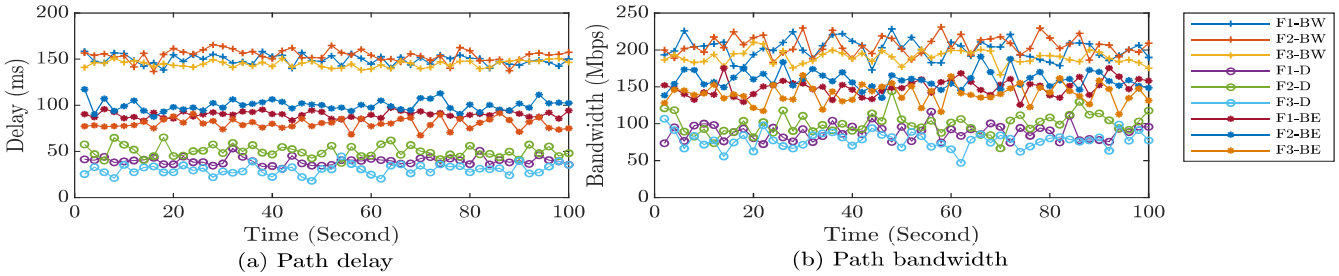


Figure 20. Bandwidth and delay of traffic flows in the proposed method (total traffic=700 mps)

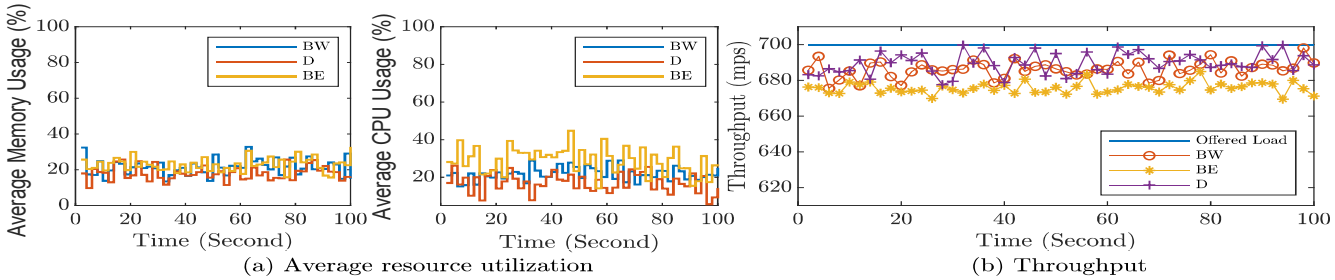


Figure 21. The consumption of resources and the throughput of traffic flows in the proposed method (total traffic=700 mps)

1 injects BW traffic. As a result, this class owns the total capacity of the system, which is 700 mps. In the interval [250, 500] sec, the group 1 injects BW and D traffic. In this case, the capacity of the system is equally divided between the two classes of traffic (approximately 358 mps). In the interval [500, 750] sec, group 2 injects BE traffic, in addition to group 1. In this case, the capacity is equally divided between the three classes (approximately 232 mps). Eventually, the combined traffic is also injected into the network in the interval [750, 1000] sec, and the throughput of each of the four classes is equal to approximately 174 mps. Generally, the system's capacity is divided between traffic fairly and independently from the class.

4.2.6 Sixth Experiment: failure assessment in QoS

The sixth experiment examined the impact of the failure in QoS. In this experiment, the traffic of 900 mps is injected into the network for 1000 seconds, and three switches are damaged in three stages. The results are shown in Fig. 24. Before the failure of S5, all network paths are available, and the delay is negligible. With the S5 failure, paths containing the S5 are out of reach, resulting in an increase in the delay. However, the controller directs traffic flows from alternative paths, and therefore, delays are acceptable. Then, S3 is failed and thus, the delay increases again. However, the traffic

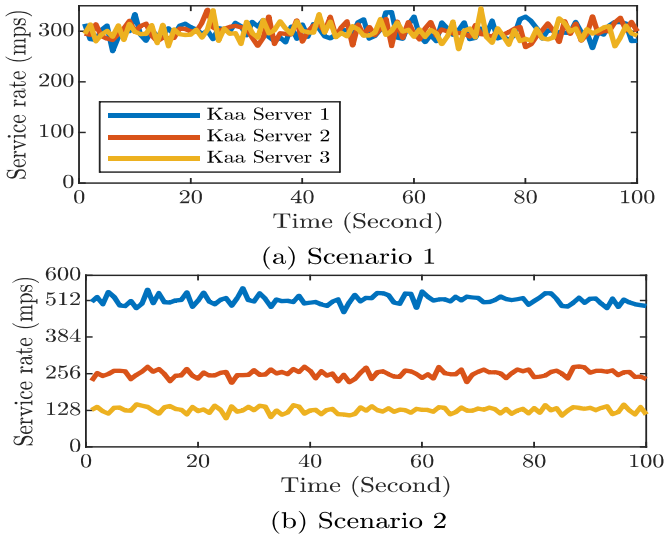


Figure 22. The service rate for Kaa servers in the proposed method with different background traffic (total traffic=900 mps)

passes through the S6-S7-S4-S2-S1 path with the controller management, and the delay returns to the normal limit below 200 ms. Finally, S2 fails, and only the S6-S7-S4 path

Table 5
Proposed Controller Performance

Total traffic (mps)	100	300	500	700	900	1100
Throughput (fps) ∼	98	297	496	695	894	1093
Average response time (ms) ∼	14.55	23.75	32.97	44.86	59.68	68.97
CPU consumption (%) ∼	4.47	8.65	12.75	16.53	25.76	38.97
Memory consumption (%) ∼	4.01	7.97	10.64	15.75	22.64	33.89

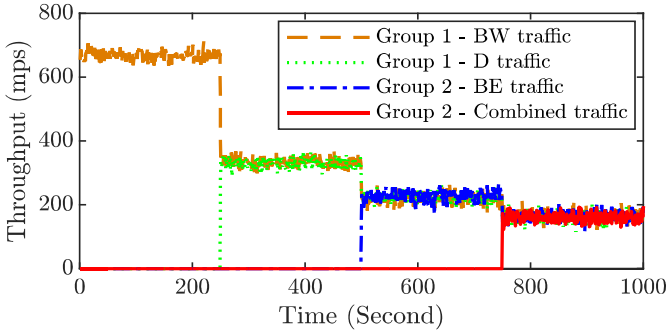


Figure 23. Fairness analysis of the proposed method (total traffic=700 mps)

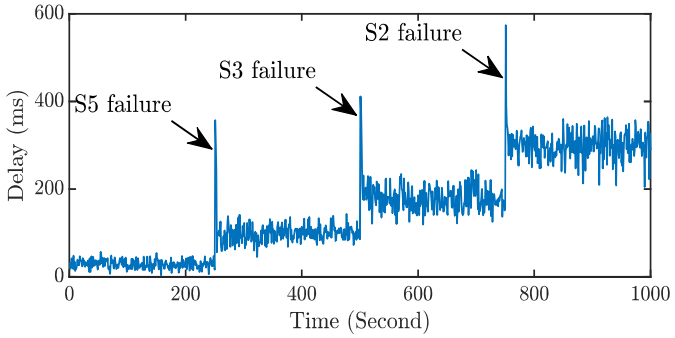


Figure 24. OpenFlow switch failure analysis (total traffic=900 mps)

and server 3 are available. In this case, the delay increases to more than 200 ms. Therefore, there is no exponential growth in the delay in any of these failures, due to having a global view by the controller, which can select the best alternative paths and servers with a significant impact on QoS.

4.2.7 *Seventh Experiment: controller performance*

The final experiment evaluates the performance of the proposed SDN-based controller. Table 5 represents the efficiency of the proposed controller in different traffic. As presented in Table, the controller has been able to handle almost all requests with low response time. Further, its modules do not have overhead because of low CPU utilization and memory. Therefore, the probability of an overload in the controller is very low, and it is not a bottleneck.

For a closer look at the two approaches, the ILP Model (optimal solution) for limited cases (limited sources and destinations) of the problem is compared with the SDN-based controller (near-optimal solution). Note that given that the ILP problem is NP-hard, the optimal solution cannot be achieved in polynomial time and increasing the size of the problem exponentially increases the time of ILP Model. We use CPLEX solver to solve ILP Model. In Table 6, the solution value and execution time is provided for various inputs for the two methods. This table shows the proposed SDN-based controller is able to closely resembles the result of ILP. As is evident in this table, execution time significantly increases for ILP method, whereas SDN-based controller has a slow growth. In fact, the proposed framework is a *heuristic* method to solve global load-balanced and QoS-aware routing problem in a IoT communication network.

5 Conclusion and Future Work

The traffic management of millions of heterogeneous devices emerging in IoT field is one of the most important aspects in IoT. The rapid evolution of IoT and the different QoS requirements of its traffic classes demand an integrated IoT resource management framework. In addition, IoT suffers from the lack of informed mechanisms for balancing the load of its servers.

The present study proved that balancing the load of IoT servers and simultaneously, satisfying the QoS of IoT traffic is an NP-hard problem considering the resource constraints and massive traffic volume. For this reason, this research provided a novel SDN-based control and management framework for IoT, which supported load balance and QoS simultaneously. In this regard, we designed a predictive and proactive controller equipped with modules based on OpenFlow, sFlow, time-series analysis, and fuzzy logic. The proposed framework was implemented in a real testbed, including Open vSwitch, Floodlight controller, and Kaa IoT servers. To the best of our knowledge, this study practically conducted wide experiments under various scenarios on the Kaa IoT platform, and presented its results for the first time. While utilizing server resources optimally, achieving high throughput and low response time indicate the proper performance of the proposed framework. Achieving fairness, reducing the impact of failure in QoS, and better performance are among other achievements of the proposed framework compared to similar methods.

As our future work, we plan to study the IoT QoS management in the distributed SDN control plane and multi-domain network. Finally, "designing more advanced policies for estimating the IoT servers load," "providing IoT services as softwarization," and "developing IoT framework based on Network Functions Virtualization (NFV) for energy and QoS management" are among the most important future works of the paper [53].

References

[1] C. Chang, S. N. Srirama, and R. Buyya, "Internet of things (iot) and new computing paradigms," *Fog and Edge Computing: Principles and Paradigms*, pp. 1–23, 2019.
[2] M. Weyrich and C. Ebert, "Reference architectures for the internet of things," *IEEE Software*, vol. 33, no. 1, pp. 112–116, Jan 2016.

Table 6
Comparison between SDN-based framework and ILP model

		$l = 50, z = 10$ $v = 8, e = 11$	$l = 60, z = 15$ $v = 10, e = 15$	$l = 70, z = 20$ $v = 12, e = 19$	$l = 80, z = 25$ $v = 14, e = 23$
Execution Time (ms)	ILP	83	699	4543	13548
	SDN-based	143	172	232	299
Objective Function ($\sum(f_p u_p)$)	ILP	97	223	345	498
	SDN-based	96	220	340	491

[3] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of things: A survey on enabling technologies, protocols, and applications," *IEEE Communications Surveys Tutorials*, vol. 17, no. 4, pp. 2347–2376, Fourthquarter 2015.

[4] L. Da Xu, W. He, and S. Li, "Internet of things in industries: A survey," *IEEE Transactions on industrial informatics*, vol. 10, no. 4, pp. 2233–2243, 2014.

[5] G. Liu, W. Quan, N. Cheng, H. Zhang, and S. Yu, "Efficient ddos attacks mitigation for stateful forwarding in internet of things," *Journal of Network and Computer Applications*, vol. 130, pp. 1–13, 2019.

[6] M. Vögler, J. M. Schleicher, C. Inzinger, and S. Dustdar, "A scalable framework for provisioning large-scale iot deployments," *ACM Transactions on Internet Technology (TOIT)*, vol. 16, no. 2, p. 11, 2016.

[7] S. Verma, Y. Kawamoto, Z. M. Fadlullah, H. Nishiyama, and N. Kato, "A survey on network methodologies for real-time analytics of massive iot data and open research issues," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1457–1477, 2017.

[8] A. Brogi and S. Forti, "Qos-aware deployment of iot applications through the fog," *IEEE Internet of Things Journal*, vol. 4, no. 5, pp. 1185–1192, 2017.

[9] I. Awan, M. Younas, and W. Naveed, "Modelling qos in iot applications," in *Network-Based Information Systems (NBiS), 2014 17th International Conference on*. IEEE, 2014, pp. 99–105.

[10] A. Haidine, S. El Hassani, A. Aqqal, and A. El Hannani, "The role of communication technologies in building future smart cities," in *Smart Cities Technologies*. InTech, 2016.

[11] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2015.

[12] W. Xia, Y. Wen, C. H. Foh, D. Niyato, and H. Xie, "A survey on software-defined networking," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 1, pp. 27–51, 2015.

[13] H. Farhady, H. Lee, and A. Nakao, "Software-defined networking: A survey," *Computer Networks*, vol. 81, pp. 79–95, 2015.

[14] W. Quan, N. Cheng, M. Qin, H. Zhang, H. A. Chan, and X. Shen, "Adaptive transmission control for software defined vehicular networks," *IEEE Wireless Communications Letters*, vol. 8, no. 3, pp. 653–656, June 2019.

[15] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[16] A. Y. Nikravesh, S. A. Ajila, and C.-H. Lung, "Measuring prediction sensitivity of a cloud auto-scaling system," in *2014 IEEE 38th International Computer Software and Applications Conference Workshops*. IEEE, 2014, pp. 690–695.

[17] R. G. Garroppo, S. Giordano, M. Pagano, and G. Procissi, "On traffic prediction for resource allocation: A chebyshev bound based allocation scheme," *Computer Communications*, vol. 31, no. 16, pp. 3741–3751, 2008.

[18] G. Gardašević, M. Veletić, N. Maletić, D. Vasiljević, I. Radusinović, S. Tomović, and M. Radonjić, "The iot architectural framework, design issues and application domains," *Wireless personal communications*, vol. 92, no. 1, pp. 127–148, 2017.

[19] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of things (iot): A vision, architectural elements, and future directions," *Future generation computer systems*, vol. 29, no. 7, pp. 1645–1660, 2013.

[20] A. H. Ngu, M. Gutierrez, V. Metsis, S. Nepal, and Q. Z. Sheng, "Iot middleware: A survey on issues and enabling technologies," *IEEE Internet of Things Journal*, vol. 4, no. 1, pp. 1–20, 2017.

[21] C. M. Medaglia and A. Serbanati, "An overview of privacy and security issues in the internet of things," in *The Internet of Things*. Springer, 2010, pp. 389–395.

[22] Z.-K. Zhang, M. C. Y. Cho, C.-W. Wang, C.-W. Hsu, C.-K. Chen, and S. Shieh, "Iot security: ongoing challenges and research opportunities," in *Service-Oriented Computing and Applications (SOCA), 2014 IEEE 7th International Conference on*. IEEE, 2014, pp. 230–234.

[23] I. Farris, T. Taleb, Y. Khettab, and J. S. Song, "A survey on emerging sdn and nfv security mechanisms for iot systems," *IEEE Communications Surveys & Tutorials*, 2018.

[24] M. Chen, S. Mao, and Y. Liu, "Big data: A survey," *Mobile networks and applications*, vol. 19, no. 2, pp. 171–209, 2014.

[25] M. Strohbach, H. Ziekow, V. Gazis, and N. Akiva, "Towards a big data analytics framework for iot and smart city applications," in *Modeling and processing for next-generation big-data technologies*. Springer, 2015, pp. 257–282.

[26] N. Kaur and S. K. Sood, "An energy-efficient architecture for the internet of things (iot)," *IEEE Systems Journal*, vol. 11, no. 2, pp. 796–805, 2017.

[27] A. Brogi and S. Forti, "Qos-aware deployment of iot applications through the fog," *IEEE Internet of Things Journal*, vol. 4, no. 5, pp. 1185–1192, Oct 2017.

[28] R. Duan, X. Chen, and T. Xing, "A qos architecture for iot," in *Internet of Things (iThings/CPSCom), 2011 International Conference on and 4th International Conference on Cyber, Physical and Social Computing*. IEEE, 2011, pp. 717–720.

[29] G. White, V. Nallur, and S. Clarke, "Quality of service approaches in iot: A systematic mapping," *Journal of Systems and Software*, vol. 132, pp. 186–203, 2017.

[30] Y. Liu, Y. Kuang, Y. Xiao, and G. Xu, "Sdn-based data transfer security for internet of things," *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 257–268, 2018.

[31] Y. Jararweh, M. Al-Ayyoub, E. Benkhelifa, M. Vouk, A. Rindos *et al.*, "Sdiot: a software defined based internet of things framework," *Journal of Ambient Intelligence and Humanized Computing*, vol. 6, no. 4, pp. 453–461, 2015.

[32] S. Bera, S. Misra, S. K. Roy, and M. S. Obaidat, "Soft-wsn: Software-defined wsn management system for iot applications," *IEEE Systems Journal*, vol. 12, no. 3, pp. 2074–2081, 2018.

[33] L. Galluccio, S. Milardo, G. Morabito, and S. Palazzo, "Sdn-wise: Design, prototyping and experimentation of a stateful sdn solution for wireless sensor networks," in *Computer Communications (INFOCOM), 2015 IEEE Conference on*. IEEE, 2015, pp. 513–521.

[34] D. Wu, D. I. Arkhipov, E. Asmare, Z. Qin, and J. A. McCann, "Ubiflow: Mobility management in urban-scale software defined iot," in *2015 IEEE Conference on Computer Communications (INFOCOM)*, April 2015, pp. 208–216.

[35] M. T. Kakiz, E. Öztürk, and T. Çavdar, "A novel sdn-based iot architecture for big data," in *Artificial Intelligence and Data Processing Symposium (IDAP), 2017 International*. IEEE, 2017, pp. 1–5.

[36] S. Din, M. M. Rathore, A. Ahmad, A. Paul, and M. Khan, "Sdiot: Software defined internet of thing to analyze big data in smart cities," in *2017 IEEE 42nd Conference on Local Computer Networks Workshops (LCN Workshops)*, Oct 2017, pp. 175–182.

[37] C. Kharkongor, T. Chithralekha, and R. Varghese, "A sdn controller with energy efficient routing in the internet of things (iot)," *Procedia Computer Science*, vol. 89, pp. 218–227, 2016.

[38] G.-C. Deng and K. Wang, "An application-aware qos routing algorithm for sdn-based iot networking," in *2018 IEEE Symposium on Computers and Communications (ISCC)*. IEEE, 2018, pp. 00 186–00 191.

[39] N. Saha, S. Bera, and S. Misra, "Sway: Traffic-aware qos routing in software-defined iot," *IEEE Transactions on Emerging Topics in Computing*, pp. 1–1, 2018.

[40] N. Bizanis and F. A. Kuipers, "Sdn and virtualization solutions for the internet of things: A survey," *IEEE Access*, vol. 4, pp. 5591–5606, 2016.
[41] C. H. Papadimitriou, "On the complexity of integer programming," *Journal of the ACM (JACM)*, vol. 28, no. 4, pp. 765–768, 1981.
[42] "Onf sdn model," www.opennetworking.org, accessed: 2019-01-30.
[43] "sflow protocol," www.sflow.org, accessed: 2019-01-30.
[44] S. S. Haykin, *Adaptive filter theory*. Pearson Education India, 2008.
[45] "Kaa project wiki," www.kaaproject.org, accessed: 2019-01-30.
[46] "Open vswitch," www.openvswitch.org, accessed: 2019-01-30.
[47] "Floodlight controller," www.projectfloodlight.org, accessed: 2019-01-30.
[48] "Netty framework," www.netty.io, accessed: 2019-01-30.
[49] "Network monitoring tool," www.iperf.fr, accessed: 2019-01-30.
[50] H. Zhong, Y. Fang, and J. Cui, "Lbbsrt: An efficient sdn load balancing scheme based on server response time," *Future Generation Computer Systems*, vol. 80, pp. 409–416, 2018.
[51] A. Montazerolghaem, S.-K. Shekofteh, M. Yaghmaee, and M. Naghibzadeh, "A load scheduler for sip proxy servers: design, implementation and evaluation of a history weighted window approach," *International Journal of Communication Systems*, vol. 30, no. 3, p. e2980, 2017.
[52] H. Jiang, A. Iyengar, E. Nahum, W. Segmuller, A. N. Tantawi, and C. P. Wright, "Design, implementation, and performance of a load balancer for sip server clusters," *IEEE/ACM transactions on networking*, vol. 20, no. 4, pp. 1190–1202, 2012.
[53] I. Sarrigiannis, K. Ramantas, E. Kartsakli, P.-V. Mekikis, A. Antonopoulos, and C. Verikoukis, "Online vnf lifecycle management in a mec-enabled 5g iot architecture," *IEEE Internet of Things Journal*, 2019.