ESMLB: Efficient Switch Migration-Based Load Balancing for Multicontroller SDN in IoT

Kshira Sagar Sahoo, Deepak Puthal, *Senior Member, IEEE*, Mayank Tiwary, Muhammad Usman, Bibhudatta Sahoo, Zhenyu Wen, Biswa P. S. Sahoo, and Rajiv Ranjan, *Senior Member, IEEE*

***Abstract*—In software-defined networks (SDNs), the deployment of multiple controllers improves the reliability and scalability of the distributed control plane. Recently, edge computing (EC) has become a backbone to networks where computational infrastructures and services are getting closer to the end user. The unique characteristics of SDN can serve as a key enabler to lower the complexity barriers involved in EC, and provide better quality-of-services (QoS) to users. As the demand for IoT keeps growing, gradually a huge number of smart devices will be connected to EC and generate tremendous IoT traffic. Due to a huge volume of control messages, the controller may not have sufficient capacity to respond to them. To handle such a scenario and to achieve better load balancing, dynamic switch migrating is one effective approach. However, a deliberate mechanism is required to accomplish such a task on the control plane, and the migration process results in high network delay. Taking it into consideration, this article has introduced an efficient switch migration-based load balancing (ESMLB) framework, which aims to assign switches to an underutilized controller effectively. Among many alternatives for selecting a target controller, a multicriteria decision-making method, i.e., the technique for order preference by similarity to an ideal solution (TOPSIS), has been used in our framework. This framework enables flexible decision-making processes for selecting controllers having different resource attributes. The emulation results indicate the efficacy of the ESMLB.**

***Index Terms*—Controller, software-defined network (SDN), switch migration, technique for order preference by similarity to an ideal solution (TOPSIS).**

I. Introduction

THE POPULARITY of software-defined networks (SDNs) has been growing due to their unique characteristics, i.e., the separation of the control plane from the forwarding plane [1], [2]. The control plane can have a global view of the whole network, which supports efficient control of the underlying forwarding planes. Nowadays, the edge computing (EC) paradigm has become a trend that brings computational infrastructures and services closer to end devices, instead of relying on cloud datacenters (CDC) [6], [7]. The decoupled characteristics of SDN can bridge the gap between EC and CDC. The combination of EC and CDC can lower the complexity of the EC architectures and utilize the available resources more efficiently. For instance, the control plane can act as a decision maker on whether the incoming task should execute in edge or forward to cloud [5]. Additionally, the programmability of the network and multitenant capability of SDN improves the resource utilization and network performance by resolving various application requirements. Fig. 1 gives an illustration of multicontroller SDN architecture for IoT-based EC systems. Despite all these impressive innovations, the key problem faced by the controllers is to manage a large number of control messages during the peak hour of the day. In a mature software-defined Internet of Things (IoT) system, huge data transmission occurs which causes a bottleneck at the switch. In turn, a massive volume of control requests makes the controller unstable.

Load balancing techniques could be static or dynamic. The static mapping between switches and controller creates reliability issues and consequently degrade the performance of the network due to the uneven load distribution among controllers [9]. Dynamic approaches are always more effective than static methods due to the load assignment process-based on the traffic pattern of the network. In SDN, a controller is overloaded primarily due to the *packet_in* control messages sent by the switches. Different parameters are considered for load balancing in SDN, such as resource utilization, throughput, execution time, energy consumption, and peak load ratio, etc. From the latest survey, load balancing approaches can be categorized into two subcategories: 1) deterministic and 2) nondeterministic approaches [3]. The load migration, i.e., rerouting of the flows are some of the techniques of the deterministic approach. The output is always the same for the deterministic approach, whereas outputs always differs for the nondeterministic approach. Response time optimization, increased controller throughput are the primary advantages of the deterministic approach.

From the literature, authors have used both approaches for solving load balancing issues. But, in recent years, research on switch migration-based load balancing is predominant because it supports dynamic load management and flexible

Manuscript received September 1, 2019; revised October 13, 2019; accepted November 1, 2019. Date of publication November 8, 2019; date of current version July 10, 2020. *(Corresponding authors: Kshira Sagar Sahoo; Zhenyu Wen.)*

K. S. Sahoo is with the Department of IT, VNRVJIET, Hyderabad 50090, India (e-mail: kshirasagar12@gmail.com).

D. Puthal, Z. Wen, and R. Ranjan are with the School of Computing, Newcastle University, Newcastle upon Tyne NE1 7RU, U.K. (e-mail: zhenyu.wen@newcastle.ac.uk).

M. Tiwary is with Core Cloud Platform, SAP Lab, Bengaluru 560066, India.

M. Usman is with the School of SEIT, Federation University, Ballarat, VIC 3350, Australia.

B. Sahoo is with the Department of Computer Science and Engineering, National Institute of Technology Rourkela, Rourkela 769008, India.

B. P. S. Sahoo is with the Department of Electrical Engineering, National Taiwan University, Taipei 10617, Taiwan.

Digital Object Identifier 10.1109/JIOT.2019.2952527

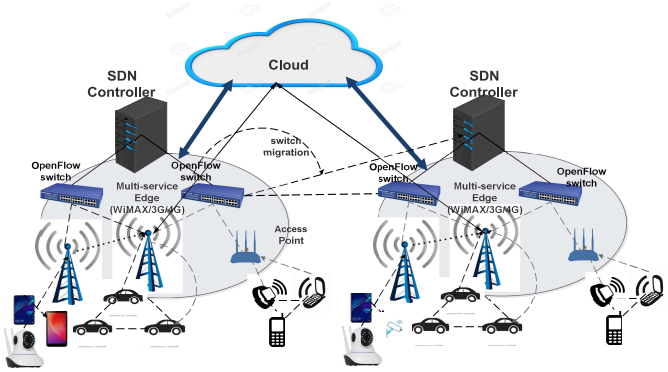


Fig. 1. Multicontroller SDN architecture in IoT-based EC system.

deployment. In this article, a novel switch migration strategy with a detailed process for the control plane load balancing is proposed and implemented.

The major contributions of this article compared to other related works are abstracted as follow.

1) With the thought of controller load balancing, the efficient switch migration technique for load balancing (ESMLB) framework has been explored. Through investigating the device migration process, the proposed framework recognizes the under-utilized controller by averaging the control plane load. In the meantime, the candidate switch is elected based on a selection probability.
2) For choosing the target controller, TOPSIS—a decision analysis method—is used to rank the controllers based on certain criteria, such as current memory usage, CPU load, bandwidth status of the controller, and hop count.
3) The proposed framework has been tested over the Mininet platform with Floodlight as the experimental controller and later the result outcomes are compared with previous relevant works.

The remaining part of this article is structured as follows. Section II gives a summary of related work and motivation to the research problem. In Section III, the proposed ESMLB framework for SDN load balancing is outlined. The emulation results of the ESMLB are presented in Section IV. Finally, the conclusion and future directions are summarized in Section V. In the rest of this article, the terms switch migration and device migration are used interchangeably.

II. Related Work and Problem Motivation

This section outlines the up-to-date research progress of load-balancing techniques in distributed controller environments that support the description of our research background and its theoretical justification.

A. Related Work

Traditional SDN architecture relies on a centralized control plane. Potential issues, such as scalability, performance, and reliability of the control plane are faced by the centralized architecture. Hence, researchers proposed distributed control plane architecture, such as Onix, Kandoo, ONOS, etc. Although the distributed architecture solves the above mentioned issues, it encounters several other limitations. In a distributed multicontroller environment, the mapping between the switches and controllers is static in nature. The unexpected network demand and dynamic changing of the topology create an uneven load distribution among controllers as well as synchronization of the control events issues [11], [23]. For instance, in the deployment scenario of cloud and software-defined IoT (SD-IoT), these issues may degrade the overall resource utilization of controllers leading to suboptimal performance. A few load balancing works are tabulated in Table I.

According to OpenFlow v1.3, a device can be attached to multiple controllers; furthermore, it allows the load to be distributed among other controllers [4]. In order to address the load imbalance problem in SDN, few authors have embraced the device migration approach, and a few important works are reviewed below. First, Advait *et al.* proposed device migration technique for load-balancing in SDN [12]. For switch migration, the authors suggested the nearby controller for transferring the load, with an expectation to reduce the migration time. The proposed technique was suitable for LAN controllers but did not address the issues of the wide area network. In the distributed load balancing approach (DALB), for each controller, a threshold is defined. When the load of a controller increases this limit, the load collection process starts, and it goes to the load balancing phase [15]. In order to achieve the optimal performance, deciding the threshold value was the key issue of this approach. Liang *et al.* [14] suggested a device migration-based load balancing approach for a group of controllers. Although it supports controller failover, at the same time the response time increases significantly. In another work, Cheng and Chen [13] proposed a migration algorithm that randomly selects a switch for migration, which could quickly lead to new load imbalance. Bari *et al.* introduced a dynamic controller-provisioning framework that minimizes the flow setup time. However, massive state synchronization is required during the reassignment of the entire control [10]. Yu *et al.* [24] proposed a mechanism based on load information sharing strategy. In their work, each controller reports its load status to other neighboring controllers periodically. Moreover, it reduces the decision time during an overloaded situation. Further, Cui *et al.* suggested SMCLBRT, a load-balancing strategy based on the changing features of real-time response times [25]. This scheme consumes high bandwidth and produces congestion due to simultaneous switch migration by multiple controllers.

B. Motivation

In order to achieve optimal performance of the overall system, certain network parameters and heterogeneous controller resources, such as CPU load, memory, bandwidth, delay, and hop count, need to be optimized. During optimization, specific criteria need to be maximized (such as memory usage, CPU utilization), and other criteria need to be minimized (such as latency, hop-count). In the above discussed state-of-the-art approaches of controller selection, all the criteria are assumed to have equal weight. Such assumptions are unrealistic for real-time applications. Moreover,

TABLE I
SELECTED LOAD BALANCING TECHNIQUES USED IN SDN

Authors/Year	Techniques	Short Analysis
Dixit *et al.* [12], 2014	Elastic distributed controller architecture through switch migration (Elasticon)	Better load balancing but response time increases after a certain threshold value, and designed for small-scale networks.
Wang *et al.* [21], 2016	Switch migration based decision making (SMDM) scheme	Improves the migration efficiency and enabling elasticity of controller via switch migration. But, latency and throughput have not been measured.
Cell *et al.* [22], 2017	BalCon (Balanced Controller) via switch migration	Increase the load balancing rate through migration decision but suitable for small-scale network.
Kang and Choo [20], 2018	SDN enhanced inter-cloud manager (S-ICM)	Reduce average response time in the congested network. The disadvantage is generating additional control messages and bottleneck.
Cui et al. [25], 2018	Multiple controller load-balancing strategy based on response time (SMCLBRT)	This scheme increases the load balancing among the controllers. However, it consumes high network bandwidth and selecting appropriate response time is difficult.
Hu et al. [19], 2019	Efficiency-Aware Switch Migration (EASM) for balancing controller load using Simulated Annealing technique.	Improves the load balancing rate, response time, throughput of the control plane. Security and WAN issues are silent.

individual criterion is tested separately, and mutual effects have not been discussed.

From the above challenges, the proposed ESMLB adopts a multicriteria decision analysis method called the technique for order preference by similarity to an ideal solution (TOPSIS). It is a decision-making technique and most suitable where there are limited numbers of choices, but each has a considerable number of criteria (properties). In this context, it enables a flexible selection of the suitable target controllers with heterogeneous criteria related to the controller resources.

III. ESMLB FRAMEWORK FOR CONTROLLER LOAD BALANCING

This section describes the proposed ESMLB approach for scalable load balancing of controllers. The network topology is presented by an un-directed graph, i.e., $G = (V, E)$, where $V = \{V_1, V_2, \ldots, V_n\}$ is a set of controllers and switches. Let, $C = \{C_1, C_2, \ldots, C_k\}$ be the set of controllers and $S = \{s_1, s_2, \ldots, s_m\}$ be the set of switches, in general $V = C \bigcup S$. The load on the controller is dynamic in nature. Let, $\mathcal{G}_{C_i} \in S$, represent the switch set managed by the controller C_i. When the controller gets overloaded, a set of switches $\mathcal{G}'_{C_i}$, i.e., $\mathcal{G}'_{C_i} \in \mathcal{G}_{C_i}$ migrate to target controller C_T. The switches to controllers interconnection edges are assumed to be full-duplex, represented by $E = \{e_1, e_2, \ldots, e_r\}$. In the switch migration mechanism, after load shifting, the current control domain becomes the new master domain whereas the previous one automatically goes to the slave mode.

The load balancing scheme using the ESMLB framework and the workflow of the proposed ESMLB are depicted in Figs. 2 and 3, respectively. The step by step process, starting from load measurement to load distributions are outlined in the following sections. Each controller contains a copy of the ESMLB framework, and it consists of four modules. For instance, *Load_Measurement* module is used for the load measurement of a controller. The target controller selection (TCS) module outlined in Algorithm 4, is designated for selecting a target controller, and candidate switch selection (CSS) module selects a candidate switch for migration. Finally, the *Switch_Migration* module is responsible for migrating the selected device to the target controller. The

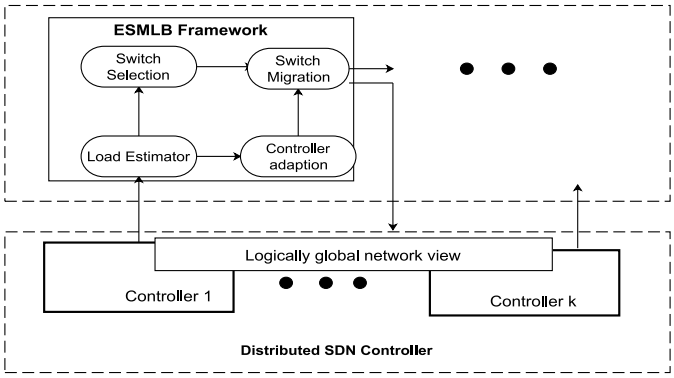


Fig. 2. Load balancing scheme using ESMLB framework.

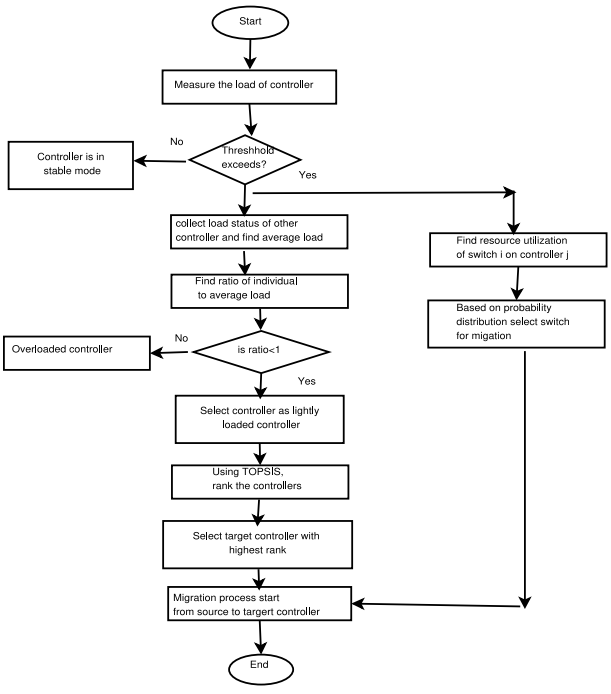


Fig. 3. Flowchart of ESMLB framework.

notations used in ESMLB are listed in Table II. There are some realistic assumptions made as follows while presenting ESMLB.

TABLE II
NOTATIONS USED IN ESMLB

Notations	Description
R	Static relation between switch and controller.
τ_i	Resource utilization ratio.
θ_j^x,θ_j^y,θ_j^z	The weighted value of CPU, memory, bandwidth on C_j.
UR_{ij}	Estimated resource utilization produced by s_i on C_j.
MC_{mx}	Message exchange cost.
MC_{il}	Increased load on controller.
$L(C_i)$	Current load status of controller C_i.
φ	Degree of current load to average load.
δ'	Controller's threshold.
C_U	Under-loaded controller list.

Algorithm 1 ESMLB Framework

Input: δ'
Output: control plane load balancing successful
1: **if** $L(C_{cl} > \delta')$ **then**
2: Construct R and call *Load_Measurement*()
3: Construct $\mathcal{D}$ and call *TCS*()
4: For candidate switch selection call *CSS*()
5: *Switch_Migration* starts
6: Load status updation of both controllers
7: **end if**
8: **return** control plane load balancing successful

Assumption 1: In the network, switches can access and migrate from one controller domain to another in order to balance the control plane load.

Assumption 2: If a switch has elected for migration, it is not allowed to return back to the former controller domain until all control requests have been accomplished by the new controller domain.

Assumption 3: Only one controller is allocated to each switch as a master controller and the rest run in equal or slave mode.

Assumption 4: All the controllers cannot be overloaded for the time frame T.

The description of different algorithms used by ESMLB is abstracted below. Inside the main ESMLB module, the remaining three algorithms have been called. If the current controller load exceeds the δ' value, it calls the load measurement module. Then both TCS and CSS modules call one after another. The switch migration process conducted by *Switch_Migration* module. The entire process is illustrated in Algorithm 1.

A. Load Measurement

Three parameters are considered for measuring the load on the controller. The resource utilization becomes imbalanced in terms of CPU, memory, bandwidth due to the heavy demand from the forwarding plane. The static connection within switch s_i and controllers C_j can be designated by a matrix R, where each entry, i.e., $r_{ij} \in R$ is a binary value and it can be described as

$$r_{ij} = \begin{cases} 1, & \text{if } s_i \in \mathcal{G}_{\mathcal{C}_j} \\ 0, & \text{otherwise.} \end{cases}$$

Let, n_i be the *packet_in* counts generated by s_i to controller C_j at time T. Due to *packet_in* requests, the total load of C_j can be defined as

$$L(C_j) = \sum_{s_i \in C_j} n_i(T) r_{ij} \tag{1}$$

Let τ_j be the estimated resource utilization of C_j. The resource utilization of C_j expresses the consumed fraction of resources on three different factors: CPU, memory, and bandwidth. It can be expressed by using

$$\tau_j = \sum_{s_i \in C_j} \tau_{ij}. \tag{2}$$

τ_{ij} is the aggregate sum of calculated resource utilization component created by s_i on C_j. It can be further expressed as

$$\tau_{ij} = \log\left(\theta_j^x \frac{n_i * x_i}{\mu_j} + \theta_j^y \frac{n_i * y_i}{\nu_j} + \theta_j^z \frac{n_i * z_i}{\psi_j}\right) \tag{3}$$

where the terms $n_i * x_i$, $n_i * y_i$, and $n_i * z_i$ denote the CPU, memory, bandwidth of controller utilized by s_i, respectively. The value θ_j^x, θ_j^y, θ_j^z denotes three different weights that satisfy $\theta_j^x + \theta_j^y + \theta_j^z = 1$. Additionally, μ_j, ν_j, and ψ_j represent three resource capabilities of C_j.

To realize this process, *Load_Measurement*() measures the current mean load of the network based on the above discussed process

$$\bar{L} = \frac{1}{k} \sum_{i=1}^{k} L(C_i). \tag{4}$$

Then, *Load_Measurement*() calculates φ, which is the factor of current load to the average load of the network, i.e.,

$$\varphi = \frac{L(C_i)}{\bar{L}}. \tag{5}$$

This ratio generates three cases.

1) $\varphi < 1$: It shows the controller is in under utilized state.
2) $\varphi = 1$: It shows the controller is in stable state.
3) $\varphi > 1$: It indicates the controller is in overloaded state.

Considering these cases, all the controllers are listed in two sets: 1) over provisioned controller set (C_O) and 2) under provisioned controller set (C_U). It can be written as follows:

$$C_O = \begin{cases} \varphi > 1, & \text{over provisioned} \\ \varphi = 1, & \text{stable} \end{cases}$$
$$C_U = \{ \varphi < 1, \quad \text{under provisioned.}$$

The overall working principle of the *Load_Measurement* module is discussed in Algorithm 2.

Algorithm 2 Load_Measurement

Input: C
Output: C_U
1: $k = |C|$
2: **for** *controller* $i = 1$ to k **do**
3: Fetch $L(C_i)$
4: $Sum = Sum + L(C_i)$
5: **end for**
6: calculate $\bar{L}$
7: **for** *controller* $i = 1$ to k **do**
8: $\varphi = \frac{L(C_i)}{\bar{L}}$
9: **if** $\varphi < 1$ **then**
10: $C_U \leftarrow C_i$
11: **else**
12: $C_O \leftarrow C_i$
13: **end if**
14: **end for**
15: **return** C_U

Algorithm 3 CSS

Input: Current Controller C_j
Output: Candidate Switch set Q
1: $Q \leftarrow NULL$
2: **for** $\forall s_i \in C_j$ **do**
3: **for** $\forall Flows \in S_i$ **do**
4: $Receivedpackets \leftarrow s_i.flowstats()$
5: Estimate τ using Equation (3)
6: **end for**
7: **end for**
8: $Q \leftarrow$ select switch based on Equation (6)
9: **return** Q

B. Switch Selection

For selecting a switch, the CSS module follows

$$\mathrm{UR}_{ij} = \frac{e^{\frac{-\tau_{ij}}{1-\tau_{ij}}}}{\sum_{s_i \in C_j} e^{\frac{-\tau_{ij}}{1-\tau_{ij}}}}. \tag{6}$$

where τ_{ij} represents the utilization component which was discussed in (3). The switch which generates higher control requests will bear additional overhead for state synchronization. Hence, the selection probability of a switch is higher if its τ_{ij} value is smaller. In other words, a larger UR_{ij} increases the likelihood of s_i for migration.

The entire process of switch selection is outlined in Algorithm 3. At first, the algorithm obtains the packet counts passing across each switch at time T. Two control messages, such as *STATS_REQUEST* and *STATS_REPLY*, are used for packet count, designated by *s.flowstats*() (step 5). The module runs across each attached device of an over-provisioned SDN controller and meanwhile, it determines the estimated resource utilization factor produced by the devices. Later, based on (6), the module returns the candidate switch set Q. In step 1, the switch selection array Q is set to NULL. The two loops from step 2 to 7 collect the flow statistics and evaluate the resource utilization ratio. After evaluation, the set Q keeps the switch information according to the probability function UR_{ij}.

C. Target Controller Selection

The prime objective of the ESMLB is to select a least loaded controller in terms of CPU, memory, and bandwidth resources. More precisely, our controller selection is based on the technique for order of preference by similarity to ideal solution (TOPSIS) which is an efficient multi criteria decision making (MCDM) method [8].

Selecting the optimal target controller for switch migration with different parameters is an optimization problem for a large scale network. There are several dimensions, such as CPU, memory, and bandwidth of the controller consider for the optimal controller resource selection. Since a controller can be overloaded, a metric has been defined consisting of three different resources, that captures the combined load status. Additionally, hop-count has been considered for controller selection, which has a greater impact on large networks. This method chooses an alternative which is as close as possible to the ideal solution. The proximity of each performance parameter is measured in Euclidean distance with optional weight value. For a single decision maker, the TOPSIS technique involves the following steps.

Step 1) For solving a multicriteria decision-making problem, TOPSIS starts with the construction of a decision making matrix $\mathcal{D}$. The decision matrix can be expressed in the following way:

$$\mathcal{D} = \begin{bmatrix} & c_1 & c_2 & \dots & c_n \\ a_1 & x_{11} & x_{12} & \dots & x_{1n} \\ a_2 & x_{21} & x_{22} & \dots & x_{2n} \\ \dots & \dots & \dots & \dots & \dots \\ a_m & x_{m1} & x_{m2} & \dots & x_{mn} \end{bmatrix}$$

where $\{a_1, a_2, \dots, a_m\}$ are the possible alternatives from which the decision maker has to choose the suitable one based on the criteria set $\{c_1, c_2, \dots, c_n\}$. Moreover, x_{ij} is the rating of alternate a_i with respect to criterion c_j. The values of the criteria for alternatives is used for order preference.

Step 2) In this step, different attribute dimensions are transferred to nondimensional attributes. Usually, a different criterion has different measuring units. Hence, the value of decision matrix $\mathcal{D}$ has transformed into a normalized decision matrix $\mathcal{R}$, which is calculated as follows:

$$\mathcal{R} = \begin{bmatrix} r_{11} & r_{12} & \dots & r_{1n} \\ r_{21} & r_{22} & \dots & r_{2n} \\ \dots & \dots & \dots & \dots \\ r_{m1} & r_{m2} & \dots & r_{mn} \end{bmatrix} \tag{7}$$

where

$$r_{ij} = \frac{x_{ij}}{\sqrt{\sum_{i=1}^{m} x_{ij}^2}}. \tag{8}$$

Step 3) All the selection criteria may not have equal importance. Thus, adding weighted value quantify the relative significance of the different selection criteria. In this step, the weighting decision matrix (W) is constructed by multiplying each element of $\mathcal{R}$

with the value of the random weights. Each element of the weighted normalized decision matrix (v_{ij}) is computed as follows:

$$v_{ij} = W_i \times r_{ij} \tag{9}$$

where W_i is the weight of the ith criterion and $\sum_{i=1}^{n} W_i = 1$.

Step 4) Calculate the positive ideal solution (A^+) and negative ideal solution (A^-). The A^+ solution is designated for maximizing the benefit criterion (I) and minimizing the cost criterion (J). On the other hand, A^- solution is designated for maximizing the cost and minimizing the benefit criteria. The form of both the solutions can be defined as follows:

$$\begin{aligned} A^+ &= \{v_1^+, \ldots, v_n^+\} \\ &= \left\{\left(\max\ v_{ij} | j \in I\right), \left(\min\ v_{ij} | j \in J\right)\right\} \quad (10) \\ A^- &= \left\{v_1^-, \ldots, v_n^-\right\} \\ &= \left\{\left(\min\ v_{ij} | j \in I\right), \left(\max\ v_{ij} | j \in J\right)\right\}. \quad (11) \end{aligned}$$

Step 5) This step determines the separation measure of each alternative from both positive and negative ideal solutions using Euclidean distance

$$d_i^+ = \sqrt{\sum_{j=1}^{n}\left(v_{ij} - v_j^+\right)^2}, \quad i = 1, 2, \ldots, m \tag{12}$$

$$d_i^- = \sqrt{\sum_{j=1}^{n}\left(v_{ij} - v_j^-\right)^2}, \quad i = 1, 2, \ldots, m \tag{13}$$

Step 6) Finally, determine the relative closeness (RC) to the A^+ ideal solution using

$$\mathrm{RC}_i = \frac{d_i^-}{d_i^+ + d_i^-}. \tag{14}$$

Since d_i^+ and $d_i^+ \geq 0$, it is obvious that $\mathrm{RC}_i \in [0, 1]$.

Step 7) Sort the alternatives according to RC_i. The higher RC_i, is the better alternative.

Let us discuss some definitions and examples that support the background and theoretical foundation of this article.

Definition 1 [Migration Cost (MC)]: Primarily two components have been considered for MC. One component is the total number of packet exchange during migration. The second component is the additional load on the controller. When s_i migrates from C_j to C_T, the MC MC_{s_i,C_T} can be expressed as

$$\mathrm{MC}_{s_i,C_T} = \mathrm{MC}_{mx} + \mathrm{MC}_{il}. \tag{15}$$

Furthermore, MC_{il} can be formulated as follows:

$$\mathrm{MC}_{il} = n_i.\left(h_j - h_m\right) \tag{16}$$

where, n_i is the *packet_in* messages generated from s_i, h_m denotes the minimum hop count between s_i and target controller C_T.

Example 1: The importance of the TOPSIS technique in the context of switch migration is outlined here. Let

Algorithm 4 TCS

Input: Decision matrix $\mathcal{D} = (x_{ij})_{mn}$
Output: Target Controller C_T
1: $C_T \leftarrow NULL$
2: Calculate r_{ij} using Equation (8).
3: $\mathcal{R} = r_{ij}$
4: Calculate W and compute v_{ij} using Equation (9).
5: Calculate A^+ and A^- using Equation (10), (11)
6: Calculate Euclidean distance d_i^+ and d_i^-
7: $RC_i = \frac{d_i^-}{d_i^+ + d_i^-}$
8: $sort(RC)$
9: $C_T \leftarrow RC_1$
10: **return** C_T

TABLE III
DECISION MATRIX

alternative	criteria			
	CPU_{load}	RAM (GB)	Bandwidth (Gbits/sec)	Distance (Hop count)
C_1	1000	2	0.30	2
C_2	1200	1	0.20	3
C_3	900	2	0.40	3
C_4	1100	3	0.10	4

TABLE IV
NORMALIZED DECISION MATRIX

alternative	criteria			
	CPU_{load}	RAM (GB)	Bandwidth (Gbits/sec)	Distance (Hop count)
C_1	0.238	0.357	0.300	0.166
C_2	0.285	0.285	0.200	0.250
C_3	0.214	0.142	0.400	0.250
C_4	0.261	0.214	0.100	0.330

TABLE V
WEIGHTED NORMALIZED DECISION MATRIX

alternative	criteria			
	CPU_{load}	RAM (GB)	Bandwidth (Gbits/sec)	Distance (Hop count)
C_1	0.0595	0.0892	0.3000	0.0750
C_2	0.0712	0.0712	0.0500	0.0625
C_3	0.0535	0.0355	0.1000	0.0625
C_4	0.0652	0.0535	0.0250	0.0825

$\mathrm{CPU}_{\mathrm{load}}$, RAM, *Hop_counts*, Bandwidth (BW) be the criteria for the controller set $\{C_1, C_2, \ldots, C_5\}$. Let us assume C_5 is the overloaded controller. For switch migration, now C_5 has four alternatives with four different criteria. Each alternative C_i is evaluated with respect to m criterion by C_1. Let Table III represent the current status of the controllers in the form of decision matrix. In Table III, $\mathrm{CPU}_{\mathrm{load}}$, RAM, and BW denote the current consumed resources. The CPU is typically the throughput bottleneck of a controller, and the $\mathrm{CPU}_{\mathrm{load}}$ is approximately proportioned to the arrival rate of *packet_in* message. Hence, we calculate the total *packet_in* messages reaching the controller at time T as the load. Then, the normalized decision matrix can be in the form as given in Table IV. The weighted normalized decision matrix can be calculated as $W_i \times r_{ij}$, and it can be in the form as given in Table V. For instance, for $\mathrm{CPU}_{\mathrm{load}}$ the v_j^+ and v_j^- are 0.0712 and 0.0535, respectively. Similarly, for bandwidth, 0.3 is the positive ideal value and 0.025 is the negative ideal value. Then, the separation measure of each alternative using

Euclidean distance is estimated. Finally, RC to ideal solutions using (14) is calculated. The final RC values of the alternatives are $\{0.9649, 0.1576, 0.2640, 0.5136\}$. So, the best alternative controller is C_1 which has the highest RC toward the positive ideal solution.

Lemma 1: During device migration, selecting a minimal *packet_in* generated switch can realize good load balancing performance.

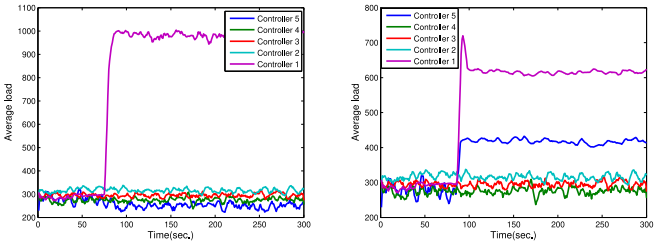
Proof: The load balancing rate expresses the level of closeness to the ideal load distribution. Let a static relation between controller and switches be represented as $<C_i : switch\ set>$. For instance, the current network is: $<C_1 : s_1, s_2, s_3>$, $<C_2 : s_4, s_5, s_6>$, and $<C_3 : s_7, s_8, s_9>$. Let the incoming packet rate of switches to different controllers be: $<C_1 : 30, 30, 30>$, $<C_2 : 30, 30, 40, 50>$, and $<C_3 : 30, 40>$. Assume the controller threshold is 120. At time t, for the above scenario, the overloaded controller C_2 (i.e., load is 150) selects C_3 as the immigrant controller. For evaluation of LBR, the following equation has been utilized:

$$\text{LBR} = 1 - \frac{\sum_{i=1}^{k} |L(C_i) - \bar{L}|}{k * \bar{L}}. \tag{17}$$

From (17), it can be noted that LBR was 0.67 prior to the migration. If controller C_2 selects s_7, the balancing performance will be 0.850 and if s_5 is chosen, it can be enhanced to 0.921. This evaluation proves that choosing a switch with fewer control events can improve the performance of the network. ■

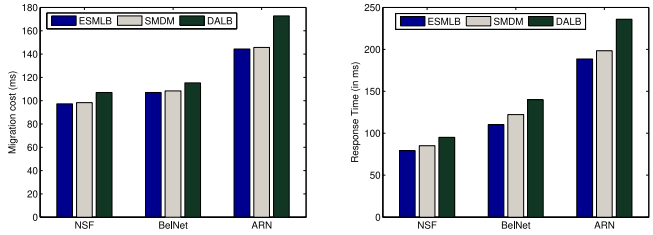
IV. Performance Evaluation

The experiment was conducted over the Mininet platform [16] with Java-based Floodlight controller which supports OpenFlow 1.3v. In order to conduct the experiment more representatively, real-time network topologies have been considered from Topology-zoo website [18]. The emulation was conducted on three separate topologies, such as NSF (nodes: 13, edges:15), BelNet (nodes: 22, edges: 43), and ARN (nodes: 30, edges: 29). For the testbed setup, five Floodlight controllers installed in different computers and Zodiac FX OpenFlow enabled switch have been used to establish the connection between end hosts and controllers. Few nodes of the topology are considered as switches and assigned to any of the five controllers for their flow management. As far as the machine configuration is concerned, Intel Core i7 processors with 32-GB RAM are used for the experiment. The network behavior is being monitored with the help of *Iperf* tool and *Cbench* tool is used in throughput mode for measuring the average capacity of the controller. Due to the loop-back problem of Mininet, it is hard to overload an SDN controller. For this reason, the controller capacity has been maintained as low as possible, i.e., 3 kilosrequest per second, and the threshold value set to 1 Kilorequest. However, in this experiment specifically the average resource demand of *packet_in* events has been estimated. For experimenting, many new flows are generated by *Iperf* tool, and for each new flow, switches send *packet_in* event to the controller. In the meantime the resource consumption (e.g., x, y, z) caused due to *packet_in* is recorded. The



(a) (b)

Fig. 4. Load distribution without and with ESMLB strategy. (a) Load distribution among controllers using SMCS strategy. (b) Load distribution with ESMLB approach.



(a) (b)

Fig. 5. Average (a) MC and (b) response time using three strategies.

computational resources utilized by an event can be obtained by $(x/new\ flows)$ and $(y/new\ flows)$. The average resource utilization is estimated after 20-time emulation of the same experiment.

A comparative study has been performed with other related works. For evaluation purposes, a static mapping between controller and switch (SMCS) scheme has been taken into consideration. Additionally, two migration schemes—switch-migration-based decision making (SMDM) and DALB—are considered. Wang *et al.* [9] proposed the SMDM scheme, which makes a migration efficiency model based on load balancing rate and MC. Zhou *et al.* [15] proposed DALB, which uses device migration for load balancing of the controller.

A. Result Discussion

To verify the effectiveness of ESMLB, the emulation was conducted for 15 min on BelNet topology. In the first experiment, the proposed work was compared with SMCS, i.e., without adopting any switch migration technique. For this experiment, a large number *packet_in*s are created by introducing new flows to the "Controller-1". Fig. 4 demonstrates the emulation result of the first 300 s, where Controller-1 is under overloaded and rest of the controllers are under loaded conditions. Fig. 4(a) illustrates the overloaded scenario without using any load balancing strategy. Fig. 4(b) shows that when the load-balancing operation starts, the proposed framework effectively rebalanced the load of Controller-1. From the final RC values [discussed in (14)], ESMLB determines "Controller-5" bears the highest RC value. Hence, after making the decision, $\mathcal{G}'_{C_1}$ migrated to Controller-5 at 90 s. The increasing trend of Controller-1's load has been well distributed before 110 s. From Fig. 5(a), it is observed that MC

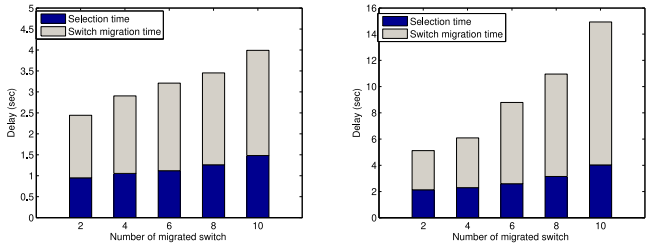


Fig. 6. Average execution time with varying number of switches. (a) NSF. (b) ARN.

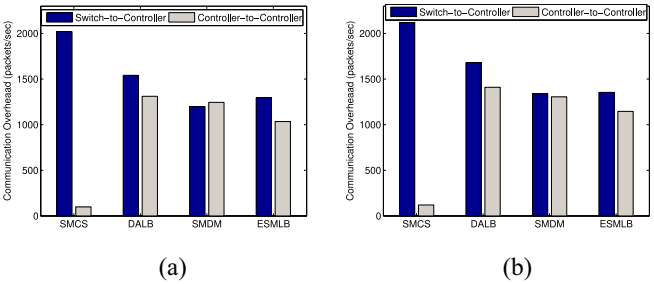


Fig. 7. Communication overhead. (a) NSF. (b) ARN.

of both ESMLB and SMDM shows a better result than the DALB approach. The reason is obvious; the DALB framework opts for a device that generates higher control requests, whereas both ESMLB and SMDM select the switch that is based on a probability function as defined in (6). As far as response time is concerned, ESMLB outperforms the other two approaches. It is also inferred that selecting the target controller using the TOPSIS technique improves the migration ability. Additionally, it avoids new possible unevenness among controllers, as illustrated in Fig. 5(b). In the next experiment, the average execution time of the proposed scheme is determined. During migration, notably, two factors need to be considered: 1) decision-making time and 2) migration time. The average execution time of the ESMLB framework with varying number of migrated devices on NSF topology is illustrated in Fig. 6(a). The selection time is observed to be linear in two topologies. However, in ARN-like larger networks the switch selection time is higher as depicted in Fig. 6(b). As the number of migrated switches increases, as a result, the migration time also rises. Here, the state synchronization of the controllers incurs an additional delay. The selection of the target controller of ESMLB is based on the TOPSIS algorithm which takes less time than the migration efficiency model adopted by the SMDM scheme. The average load balancing time is approximately 4 s [in Fig 6(a)] which is reasonably good for running different network applications with the varying requirement. Moreover, the selection process of the target controller has better performance in terms of resource utilization than the other two approaches. In another analysis, the overhead due to the controller-to-switch communication and the controller-to-controller communication of different approaches is explored. From the perspective of the switch-to-controller communication, SMCS bears the highest overhead.

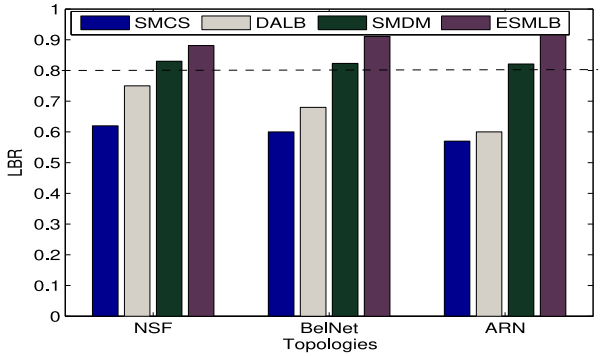


Fig. 8. Load balancing rate in various topologies.

From the perspective of the controller-to-controller communication, DALB endures the highest overheads as depicted in Fig. 7. Since SMCS does not follow any load balancing scheme, the communication between controllers is minimal, whereas the communication overheads between the switch and the controller in SMCS is maximum. In terms of controller communication overheads, ESMLB produces similar results to SMDM. In this approach, the overhead reduced by approximately 5% over SMDM and 14% over DALB, respectively, on NSF topology as shown in Fig. 7(a). This framework reduces the interaction among irrelevant controllers, which could lower the communication overheads among controllers. From this experiment, it is concluded that during controller load balancing, ESMLB bears least communication overhead as compared to the other three approaches.

Essentially, the duration of device migration influences the end-user QoS. The longer is the migration period, the more degradation of QoS and *vice versa*. From the experiment, it can be noted that ESMLB's migration time is faster, hence, this framework introduces less extra loads for the system. Finally, using (17), a comparison of the load balancing rate with three different topologies has been made. From Fig. 8, it can be noted that as the network size grows, ESMLB and SMDM schemes's load balancing efficiency is in an agreeable limit. With the increased size of the network, the load balancing rate is increasing by applying ESMLB and SMDM. In case of SMCS, the LBR is at the lowest level because it does not follow any load balancing strategy. The above analysis summarizes that the ESMLB is an efficient load balancing framework that can imperially function in a scalable SDN system.

V. Conclusion

This article focused on solving the load imbalance problem of the distributed control plane in SD-IoT. A novel load balancing scheme called ESMLB was proposed to overcome the load imbalance in the control plane. For achieving better load balancing, the proposed ESMLB monitors the real-time load information of the control plane and decides whether to implement switch migration. A multicriteria decision technique TOPSIS was introduced for choosing the target controller. Following the selection process, the selected switch shifts to the target controller domain. Finally, the proposed framework has been validated on a Mininet—a virtual network

environment, with real-time topologies. It has been found that the proposed approach outperforms SMDM and DALB approaches in relation to lower communication overhead and reduced response time. Moreover, ESMLB achieves load balancing on a distributed control plane efficiently and quickly. For the future scope of the work, this framework could be implemented for large-scale IoT environments and switch migration could happen based on the traffic demands.

REFERENCES

[1] F. Hu, Q. Hao, and K. Bao, "A survey on software-defined network and openflow: From concept to implementation," *IEEE Commun. Surveys Tuts.*, vol. 16, no. 4, pp. 2181–2206, 4th Quart, 2014.

[2] B. Qian *et al.*, "Orchestrating development lifecycle of machine learning based IoT applications: A survey," *arXiv: 1910.05433*, 2019.

[3] A. A. Neghabi, N. J. Navimipour, M. Hosseinzadeh, and A. Rezaee, "Load balancing mechanisms in the software defined networks: A systematic and comprehensive review of the literature," *IEEE Access*, vol. 6, pp. 14159–14178, 2018.

[4] Y. Xu *et al.*, "Dynamic switch migration in distributed software-defined networks to achieve controller load balance," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 3, pp. 515–529, Mar. 2019.

[5] S. Garg, K. Kaur, G. Kaddoum, S. H. Ahmed, and D. N. K. Jayakody, "SDN-based secure and privacy-preserving scheme for vehicular networks: A 5G perspective," *IEEE Trans. Veh. Technol.*, vol. 68, no. 9, pp. 8421–8434, Sep. 2019.

[6] S. Bera, S. Misra, and A. V. Vasilakos, "Software-defined networking for Internet of Things: A survey," *IEEE Internet Things J.*, vol. 4, no. 6, pp. 1994–2008, Dec. 2017.

[7] S. Garg *et al.*, "Edge computing-based security framework for big data analytics in VANETs," *IEEE Netw.*, vol. 33 no. 2, pp. 72–81, Mar./Apr. 2019.

[8] C.-T. Chen, "Extensions of the TOPSIS for group decision-making under fuzzy environment," *Fuzzy Sets Syst.*, vol. 114, no. 1, pp. 1–9, 2000.

[9] C. Wang, B. Hu, S. Chen, D. Li, and B. Liu, "A switch migration-based decision-making scheme for balancing load in SDN," *IEEE Access*, vol. 5, pp. 4537–4544, 2017.

[10] M. F. Bari *et al.*, "Dynamic controller provisioning in software defined networks," in *Proc. 9th Int. Conf. Netw. Service Manag. (CNSM)*, 2013, pp. 18–25.

[11] S. Garg, K. Kaur, S. H. Ahmed, A. Bradai, G. Kaddoum, and M. Atiquzzaman, "MobQoS: Mobility-aware and QoS-driven SDN framework for autonomous vehicles," *IEEE Wireless Commun.*, vol. 26, no. 4, pp. 12–20, Aug. 2019.

[12] D. Advait, F. Hao, S. Mukherjee, T. V. Lakshman, and R. Kompella, "Towards an elastic distributed SDN controller," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 7–12, 2013.

[13] G. Cheng and H. Chen, "Game model for switch migrations in software-defined network," *Electron. Lett.*, vol. 50, no. 23, pp. 1699–1700, Jun. 2014.

[14] C. Liang, R. Kawashima, and H. Matsuo, "Scalable and crash-tolerant load balancing based on switch migration for multiple open flow controllers," in *Proc. 2nd Int. Symp. Comput. Netw.*, 2014, pp. 171–177.

[15] Y. Zhou *et al.*, "A load balancing strategy of SDN controller based on distributed decision," in *Proc. IEEE 13th Int. Conf. Trust Security Privacy Comput. Commun.*, 2014, pp. 851–856.

[16] *Mininet Emulator*. Accessed: Oct. 20, 2018. [Online]. Available: http://mininet.org/

[17] *Floodlight Supported Switches*. Accessed: Jun. 10, 2019. [Online]. Available: https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/pages/1343519/Compatible+Switches

[18] *The Internet Topology Zoo*. Accessed: Apr. 01, 2019. [Online]. Available: http://www.topology-zoo.org/

[19] T. Hu, J. Lan, J. Zhang, and W. Zhao, "EASM: Efficiency-aware switch migration for balancing controller loads in software-defined networking," *Peer-to-Peer Netw. Appl.*, vol. 12, no. 2, pp. 452–464, 2019.

[20] B. Kang and H. Choo, "An SDN-enhanced load-balancing technique in the cloud system," *J. Supercomput.*, vol. 74, no. 11, pp. 5706–5729, 2018.

[21] W. Yong, T. Xiaoling, H. Qian, and K. Yuwen, "A dynamic load balancing method of cloud-center based on SDN," *China Commun.*, vol. 13, no. 2, pp. 130–137, 2016.

[22] M. Cello, Y. Xu, A. Walid, G. Wilfong, H. J. Chao, and M. Marchese, "BalCon: A distributed elastic SDN control via efficient switch migration," in *Proc. IEEE Int. Conf. Cloud Eng. (IC2E)*, 2017, pp. 40–50.

[23] K. S. Sahoo, M. Tiwary, B. Sahoo, R. Dash, and K. Naik, "DSSDN: Demand-supply based load balancing in software-defined wide-area networks," *Int. J. Netw. Manag.*, vol. 28, no. 4, 2018, Art. no. e2022.

[24] J. Yu, Y. Wang, K. Pei, S. Zhang, and J. Li, "A load balancing mechanism for multiple SDN controllers based on load informing strategy," in *Proc. 18th Asia–Pacific Netw. Oper. Manag. Symp. (APNOMS)*, 2016, pp. 1–4.

[25] J. Cui, Q. Lu, H. Zhong, M. Tian, and L. Liu, "A load-balancing mechanism for distributed SDN control plane using response time," *IEEE Trans. Netw. Service Manag.*, vol. 15, no. 4, pp. 1197–1206, Dec. 2018.

Kshira Sagar Sahoo received the Ph.D. degree in computer science and engineering from the National Institute of Technology Rourkela, Rourkela, India.

He is an Assistant Professor with the Department of IT, VNRVJIET, Hyderabad, India. His research interests are on performance and security in software defined networks.

Deepak Puthal (SM'19) is a Lecturer with the School of Computing, Newcastle University, Newcastle upon Tyne, U.K. His research spans several areas in cyber security, blockchain, Internet of Things, and edge/fog computing.

Mr. Puthal has received several recognitions and best paper awards from IEEE. He serves on the editorial boards of top quality international journals, including the IEEE TRANSACTIONS ON BIG DATA, *IEEE Consumer Electronics Magazine*, *Computers & Electrical Engineering* (Elsevier), the *International Journal of Communication Systems* (Wiley), and *Internet Technology Letters* (Wiley).

Mayank Tiwary is a Software Engineer, SAP Labs, Bengaluru, India. His research interests are on cloud computing, Internet of Things, and software-defined networks.

Muhammad Usman received the Ph.D. degree from the University of Technology Sydney, Ultimo, NSW, Australia.

He is with the Swinburne University of Technology, Melbourne, VIC, Australia. He is a Lecturer with the School of Science, Engineering and Information Technology, Federation University, Ballarat, VIC, Australia. His research interests are on Internet of Things and cyber security.

Bibhudatta Sahoo is an Associate Professor with the Department of Computer Science and Engineering, National Institute of Technology Rourkela, Rourkela, India. His research interests are on distributed computing, cloud computing, Internet of Things, and software defined networks.

Zhenyu Wen received the M.Sc. and Ph.D. degrees in computer science from Newcastle University, Newcastle upon Tyne, U.K., in 2011 and 2015, respectively.

He is a Postdoctoral Researcher with the School of Computing, Newcastle University. His current research interests include multiobjects optimisation, big data processing, and cloud computing.

Biswa P. S. Sahoo is currently pursuing the Graduation degree with the National Taiwan University, Taipei, Taiwan. His research interests are on Internet of Things and 5G networks.

Rajiv Ranjan (SM'15) is a Chair Professor for the Internet of Things Research with the School of Computing, Newcastle University, Newcastle upon Tyne, U.K. He is an internationally established scientist with over 260 scientific publications. He has secured more than $12 million AUD (over 6 million GBP) in the form of competitive research grants from both public and private agencies. He serves on the editorial boards for top quality international journals, including the IEEE TRANSACTIONS ON COMPUTERS from 2014 to 2016, the IEEE TRANSACTIONS ON CLOUD COMPUTING, *ACM Transactions on the Internet of Things*, *Computer* (Oxford University), *Computing* (Springer), and *Future Generation Computer Systems*.