Dynamic Bandwidth Slicing for Time-Critical IoT Data Streams in the Edge-Cloud Continuum

Fawzy Habeeb, Khaled Alwasel, Ayman Noor, Devki Nandan Jha, Duaa AlQattan, Yinhao Li, Gagangeet Singh Aujla, *Senior Member, IEEE*, Tomasz Szydlo, and Rajiv Ranjan

***Abstract*—Edge computing has gained momentum in recent years, as complementary to cloud computing, for supporting applications (e.g., industrial control systems) that require time-critical communication guarantees. While edge computing can provide immediate analysis of streaming data from Internet of Things devices, those devices lack computing capabilities to guarantee reasonable performance for time-critical applications. To alleviate this critical problem, the prevalent trend is to offload these data analytic tasks from the edge devices to the cloud. However, existing offloading approaches are static in nature as they are unable to adapt varying workload and network conditions. To handle these issues, we present a novel distributed and quality of services based multilevel queue traffic scheduling system that can undertake semiautomatic bandwidth slicing to process time-critical incoming traffic in the edge-cloud environments. Our developed system shows a great enhancement in latency and throughput as well as reduction in energy consumption for edge-cloud environments.**

***Index Terms*—Bandwidth slicing, cloud, data stream, edge, Internet of Things (IoT), multiqueues, software-defined networking (SDN), time critical.**

Manuscript received 30 December 2021; revised 30 March 2022; accepted 12 April 2022. Date of publication 25 April 2022; date of current version 9 September 2022. Paper no. TII-21-5937. *(Corresponding author: Ayman Noor.)*

Fawzy Habeeb is with the Newcastle University, NE4 5TG Newcastle Upon Tyne, U.K., and also with the University of Jeddah, Jeddah 21959, Saudi Arabia (e-mail: fawzyhabeeb1991@gmail.com).

Khaled Alwasel is with the Saudi Electronic University, Riyadh 13323, Saudi Arabia (e-mail: kalwasel@gmail.com).

Ayman Noor is with the College of Computer Science and Engineering, Taibah University, Madinah 42353, Saudi Arabia (e-mail: anoor@taibahu.edu.sa).

Devki Nandan Jha is with the University of Oxford, OX1 2JD Oxford, U.K. (e-mail: i.dnjha@gmail.com).

Duaa AlQattan is with the Newcastle University, NE4 5TG Newcastle Upon Tyne, U.K., and also with the Technical and Vocational Training Corporation, Sakakah 72389, Saudi Arabia (e-mail: d.alqattan@gmail.com).

Yinhao Li and Rajiv Ranjan are with the School of Computing, Newcastle University, NE4 5TG Newcastle Upon Tyne, U.K. (e-mail: frankleesd@gmail.com; raj.ranjan@newcastle.ac.uk).

Gagangeet Singh Aujla is with the Durham University, DH1 3LE Durham, U.K. (e-mail: gagi_aujla82@yahoo.com).

Tomasz Szydlo is with the Institute of Computer Science, AGH University of Science and Technology, 30-059 Kraków, Poland (e-mail: tomasz.szydlo@agh.edu.pl).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TII.2022.3169971.

Digital Object Identifier 10.1109/TII.2022.3169971

I. Introduction

INTERNET of Things (IoT) is an emerging paradigm that shifts routine daily workloads into smart, automated mechanisms by gathering and processing an unprecedented amount of data in a continuous manner [1]. It tracks and monitors surrounding activities (e.g., automated industrial setup) to make better decisions, increase efficiency, and improve the quality of life. Coinciding with this paradigm, IoT-based applications adopt several integrated ecosystems—from edge and cloud computing to software-defined networking (SDN) and software-defined wide area network (SD-WAN) [2], [3]. Each ecosystem offers rich features to process and transmit data according to the given quality of services (QoS) of IoT applications.

The IoT paradigm with its associated industrial ecosystems delivers unprecedented advances in technological developments. However, its heterogeneous computing and network elements still encounter two fundamental problems, which can be defined as 1) a transmission mismatch and 2) a processing mismatch [4], [5]. The former problem occurs when incoming data streams at a given network arrive faster than the network can handle and transmit. This is typically due to several reasons, such as the spike and fluctuation of incoming data and the instability of network connectivity between IoT ecosystem elements (senders and receivers) [6]–[8]. On the other hand, the processing mismatch problem arises when a given computing resource cannot process its incoming requests immediately or in a timely fashion due to the sharing mechanisms of computing resources [9], [10]. These two problems must be dealt with, especially in the context of real-time IoT applications where network and/or processing delays could lead to catastrophic incidents.

The two problems mentioned above have been tackled in different ways. For example, a data buffer technique is one typical solution that holds new arrival data for a period of time before being processed [11], [12]. Another typical solution is the leverage of classical congestion control mechanisms where new incoming data are dropped when a given buffer is overloaded [13]. Such techniques suffer from nonnegligible delays at both transmission and processing levels, especially when IoT applications are latency sensitive. Also, dropping any part of data introduces a further problem that would lead to data inconsistency with serious consequences in domains such as Industrial IoT [14]. Moreover, such techniques ignore the power of priority mechanisms at both network and host levels, which can hardly guarantee the QoS for time-critical IoT applications.

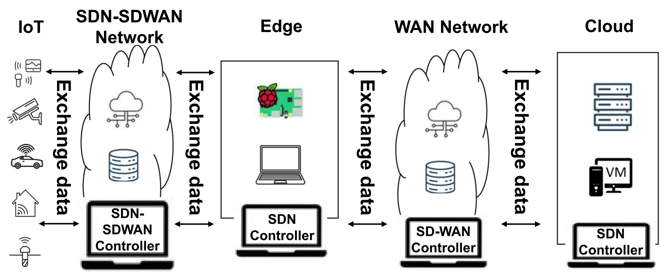


Fig. 1. IoT edge-cloud continuum modular architecture.

Several efforts have been made to address the problem of transmission and processing mismatching. For example, [15] leverage computation offloading mechanisms where data and tasks that require intensive computational resources are forwarded to an external platform (e.g., cloud datacenters). Another study [16] explores a congestion control approach focusing on tuning data transmission rates based on QoS requirements. However, the usefulness of these studies is limited to conventional environments (e.g., cloud datacenters, edge computing, etc.) without considering the bigger range of IoT ecosystems along with cutting-edge approaches, such as dynamic network slicing, load balancing, and prioritization.

Overall, this article tries to solve the research question:

What is the best way to satisfy the latency constraints along with accelerating data transmissions for IoT safety-critical applications?

To address the research question, this article presents a novel distributed IoT framework which is based on a multilevel network–host queuing mechanism, prioritization, and SDN network traffic slicing. The system is designed to make the best utilization of network and host resources in the edge-cloud continuum (see Fig. 1). It diminishes queuing delays and increases the QoS assurance of IoT applications with high-latency sensitivity as much as feasible. To do so, our proposed system deploys global network agents in SDN and SD-WAN controllers for data stream scheduling based on prioritization along with slicing bandwidth based on each IoT stream priority. The system also deploys IoT agents within each node (e.g. edge nodes, cloud nodes, etc.) to schedule IoT task executions based on multilevel queuing and prioritization. Given these systems, we formulate two different optimization problems to find the best solution for every IoT application such that the overall execution time is minimized while network bandwidth is utilized at maximum.

Solving the above question might lead to insufficient use of network resources. This can be formalized in a question context as "**How can we indicate the network slicing percentage among several priority lists such that every slice is fully used by every list?**". It is known that network bandwidth is a scarce resource where network slicing percentage should be divided according to application priority ranks. One simple solution is to use a static percentage value for each list (e.g., 50%, 30%, and 20% for three lists of high, medium, and low, respectively). However, sometimes a network bandwidth slice is not fully used by a given priory list, which leads to insufficient use of network resources. To solve this problem, we propose a heuristic auto-adaptation algorithm to dynamically tune bandwidth slicing depending on the observed network utilization of every priority.

In summary, the contributions of this article are as follows.

1) We formulate the transmission and processing mismatch problem in the edge-cloud environment.
2) We propose a novel distributed and QoS-based multilevel queues traffic scheduling system.
3) We evaluate the performance of our proposed approach using a self-driving car test case scenario.

II. Formal Model

In this section, we present the system description necessary to represent our research problem (Section II-A). Using these definitions, we formulate our problem (Section II-B). Table I summarizes all notations used in this article.

A. System Overview

Our infrastructure system X consists of four infrastructure elements and is represented as a quadruple $\langle \mathcal{D}, E, C, N \rangle$. $\mathcal{D}$ is a set of IoT devices $\mathcal{D}_i$ and is denoted by $\mathcal{D}_i = \{id_i, \delta_i\}$. Here, id_i represents the identifier of the IoT device $\mathcal{D}_i$ and δ_i represents the data rate of IoT device $\mathcal{D}_i$. E is a set of edge devices E_e with each $E_e = \{id_e, h_e\}$. id_e and h_e represents the identifier and the set of host machines $h_{e1}, h_{e2}, \ldots$ for the edge device E_e, respectively. C represents a set of cloud datacenters C_c. Each C_c is represented as $C_c = \{id_c, h_c\}$, where id_c is the identifier of the datacentre and h_c is the set of host machines $h_{c1}, h_{c2}, \ldots$. Regardless of the host type, i.e., cloud host h_{c_i} or edge host h_{e_i}, each host h_k has hardware h_k^H and software h_k^S capabilities to satisfy the requirements of the application. Now, host h_k consists of a set virtual environment $v_{1h_k}, v_{2h_k}, v_{3h_k}, \ldots$, where each v_{lh_k} can be either a virtual machine (VM) vm or a container cn. Similar to the host h_k, each virtual environment v_{lh_k} also has a hardware specification $v_{lh_k}^H$ and software specification $v_{lh_k}^S$ defined such that $\sum_l v_{lh_k}^H = h_k^H$ and $\sum_k v_{lh_k}^S = h_k^S$. Abstracting the hardware and software processing capabilities as P, we can represent the processing capability of an edge-virtual environment as $P^{E_{v_l}}$ and for cloud-virtual environment as $P^{C_{v_l}}$. Finally, N represents the network connection between $\mathcal{D}$, E, and C and is a subset of $(\mathcal{D} \times E) \cup (\mathcal{D} \times C) \cup (E \times E) \cup (E \times C) \cup (C \times C)$. A set of switches $\mathcal{S} = \{\mathcal{S}_1, \mathcal{S}_2, \ldots\}$ and SDN controllers $\sigma = \{\sigma_{\mathcal{D}}, \sigma_E, \sigma_C\}$ facilitate the network connectivity in the existing system. An IoT application A_i is defined as a directed acyclic graph of microservice $A_i = \{A_i^{\mu_1}, A_i^{\mu_2}, \ldots\}$, where each $A_i^{\mu_j}$ represents a microservice to execute. Each $A_i^{\mu_j}$ has specific hardware (H), software (S), and QoS ($\mathcal{Q}$) requirements. Equation (1) shows the combined requirements $\mathcal{R}(A_i^{\mu_j})$ for a microservice

$$\mathcal{R}(A_i^{\mu_j}) = H^{\mu_j} + S^{\mu_j} + \mathcal{Q}^{\mu_j}. \quad (1)$$

The overall requirement of A_i is given by the sum of requirements of all the microservices as given below

$$\mathcal{R}(A_i) = \sum_{\forall j} \mathcal{R}(A_i^{\mu_j}). \quad (2)$$

At any point of time t, numerous applications $A_1, A_2, \ldots$ need to be executed on the given infrastructure X. Depending on the type of application A_i, some of them require critical response while others can handle some delay. To allow a smooth execution sequence, a priority $\mathcal{P}_i$ is associated with each application A_i. IoT devices are actively generating data. We consider the IoT device $\mathcal{D}_i$ as a passive entity, i.e., it does not process any data, but transfers to the edge device. The data transfer happens on a per second basis, therefore, the total amount of data received by the edge device e_i will also be δ_i multiplied by time t. IoT devices are connected to a switch or an SDN controller σ which then forwards the data to the respective edge device. Consider the maximum bandwidth available to the IoT device d is $\mathcal{B}_d$, the time taken to transfer the data from the IoT device d to the switch/SDN controller σ can be computed as

$$T^{d\rightarrow\sigma} = \frac{\delta_d}{\mathcal{B}_{d\rightarrow\sigma}}. \tag{3}$$

The controller then forwards the data to the respective edge e while consuming $T^{\sigma\rightarrow e}$ time. Given the bandwidth of the controller as $\mathcal{B}_\sigma$, it is divided among different communication flows based on how many IoT devices are connected to it. Only an effective bandwidth $\mathcal{B}^{ef}_{\sigma\rightarrow e}$ is available for transferring one IoT device's data as

$$\mathcal{B}^{ef}_{\sigma\rightarrow e} = \frac{\mathcal{B}_{\sigma\rightarrow e}}{\text{count}_t}. \tag{4}$$

Here count_t is the number of IoT devices using the communication channel of the controller at time t. The time consumed by transferring data from controller to edge device for IoT device d is computed as

$$T^{\sigma\rightarrow e} = \frac{\delta_i}{\mathcal{B}^{ef}_{\sigma\rightarrow e}}. \tag{5}$$

Similarly, the data transfer time between edge devices e and between edge and cloud c is computed as given below

$$T^{e\rightarrow e} = \frac{\delta_e}{\mathcal{B}^{ef}_{e\rightarrow e}};\quad T^{e\rightarrow c} = \frac{\delta_c}{\mathcal{B}^{ef}_{e\rightarrow c}}. \tag{6}$$

Effective bandwidth is computed at each step by the network switch or the SDN controller, thus, allowing the data to follow a defined path. For any application A_i, the component microservice A^{μ_i} executes on numerous edge and/or cloud hosts, therefore, the total transmission time for application A_i is as given below

$$T^E_{A_i} = T^{d\rightarrow\sigma} + T^{\sigma\rightarrow e} + \sum_{\forall e1,e2\in E'} T^{e1\rightarrow e2} + \sum_{\forall e\in E,'\forall c\in C'} T^{e\rightarrow c}. \tag{7}$$

The propagation time p is computed at the start of all transmissions. Given the velocity of propagation of any transmissions as V, and the distance between the sender and the receiver as D, now, we can calculate the propagation time for the transfer time between IoT device, switch/SDN controller, edge, and cloud as given in the following equations:

$$Tp^{d\rightarrow\sigma} = \frac{D_{d\rightarrow\sigma}}{V}; Tp^{\sigma\rightarrow e} = \frac{D_{\sigma\rightarrow e}}{V}; Tp^{e\rightarrow e} = \frac{D_{e\rightarrow e}}{V}; Tp^{e\rightarrow c} = \frac{D_{e\rightarrow c}}{V}. \tag{8}$$

Following the processing happening as given in (7), the total propagation time for A_i is given below

$$T^p_{A_i} = Tp^{d\rightarrow\sigma} + Tp^{\sigma\rightarrow e} + Tp^{e\rightarrow e} + Tp^{e\rightarrow c}. \tag{9}$$

Depending on the application A_i, virtual environment $E^{v_{lh_k}}$ of edge device E^k processes the data and sends the processed data to either another virtual environment $E^{v'_{lh_k}}$ on edge or out of the cloud datacentre. Given the processing capability of an edge and cloud virtual environment, the processing time of any application microservice $A_i^{\mu_j}$ at both edge and cloud host is computed as given below

$$T^{P_e} = \frac{\mathcal{R}(A_i^{\mu_j})}{\mathcal{P}^{E_{v_l}}}; T^{P_c} = \frac{\mathcal{R}(A_i^{\mu_j})}{\mathcal{P}^{C_{v_l}}}. \tag{10}$$

Following the processing happening as given in (7), the total processing time is computed as given in (11). Here, $E' \subseteq E$ and $C' \subseteq C$ are the edge and cloud hosts executing the application microservice $A_i^{\mu_j}$, respectively

$$T^P_{A_i} = \sum_{\forall e\in E'} T^{P_e} + \sum_{\forall c\in C'} T^{P_c}. \tag{11}$$

Since the processing capability of edge/cloud virtual environment v_h is limited, a queue Q_{v_h} is associated with each of them. Data are buffered intermittently while the v_h is busy with the execution. The waiting time for the application A_i in the queue is considered to be the queuing time $T^Q_{A_i}$. The overall execution time for any application A_i is given by the combination of execution, transmission, and queuing time as given below

$$T_{A_i} = T^P_{A_i} + T^E_{A_i} + T^Q_{A_i} + T^p_{A_i}. \tag{12}$$

B. Problem Definition

Definition: Given a set of IoT applications $A = \{A_1, A2, \ldots\}$ and the infrastructure $X = \{\mathcal{D}, E, C, N\}$, a suitable deployment solution Δ_m is defined as a mapping for $A_i \in A$ to X ($\Delta_m : A_i \rightarrow X \forall A_i$) if and only if:

1) $\forall A_i^{\mu_j} \in A_i, \exists(A_i^{\mu_j} \rightarrow v_h)$ where, $h \in \{h_e \cup h_c\}$;
2) $\forall A_i^{\mu_j} \in A_i$, if $A_i^{\mu_j} \rightarrow v_h$, then $H^{\mu_j} \preceq v_h^H$ & $S^{\mu_j} \preceq v_h^S$;
3) $\sum_{\mu_j} H^{\mu_j} \leq v_h^H$ and $\sum_{\mu_j} S^{\mu_j} \leq v_h^S$.

The definition given above considers all the requirements to find a suitable deployment solution. Requirement 1 states that for all the microservices belonging to the IoT application A_i, a mapping must exist between $A_i^{\mu_j}$ and a virtual environment $v_h | h \in \{h_e \cup h_c\}$. Requirement 2 confirms that if a microservice $A_i^{\mu_j}$ is deployed to a virtual environment v_h, the hardware and software requirements of the microservice must be satisfied by v_h. Finally, requirement 3 limits the number of microservices a virtual environment can execute at any time.

The main aim of this research is to find the best solution for all the applications A_i, such that the overall execution time T_{A_i} is minimum while the effective bandwidth $\mathcal{B}^{ef}$ is utilized at maximum. In addition to this, the queuing time $T^Q_{A_i}$ for the highest priority application $A_{\mathcal{P}}$ should be as low as possible. Given these requirements, we can represent our problem as given

TABLE I
SYMBOL TABLE

Symbol	Description
X	System infrastructure
$\mathcal{D}_i$	An IoT device
δ_i	The data rate of IoT device
E	A set of edge devices
h	A set of host machines
C	A set of cloud datacenters
v	Virtual environment
vm	A virtual machine
cn	A container
P	Processing capabilities
N	The network connection between $\mathcal{D}$, E, and C
$\mathcal{S}$	A set of switches
σ	An SDN controller
A_i	An IoT application
S	Software
H	Hardware
$\mathcal{Q}$	QoS
R	Requirements
$\mathcal{P}_i$	Priority
$\mathcal{B}$	The maximum bandwidth available
$T^{d\to\sigma}$	The time taken to transfer the data from IoT device to SDN controller
$T^{\sigma\to e}$	The time taken to transfer the data from SDN controller to edge device
$T^{e\to e}$	The time taken to transfer the data from edge device to edge device
$T^{e\to c}$	The time taken to transfer the data from edge device to cloud
$\mathcal{B}^{ef}$	Effective bandwidth
count	The number of IoT devices using the communication channel of the controller
$T^{E}_{A_i}$	The total transmission time for an application
V	The velocity of propagation of any transmissions
D	The distance between the sender and the receiver
Tp	The propagation time
$T^{p}_{A_i}$	The total propagation time for an application
T^{P_e}	The processing time of any application's microservices
Q	A queue
$T^{Q}_{A_i}$	The queuing time of any application
T_{A_i}	The overall execution time for any application
SC_i	The final priority score
$ratio_i$	Compute size from megabytes to ratio
$size_i$	The IoT application size in megabytes
λ	The static deciding factor among $\mathcal{P}_i$ and $size_i$
$path$	The channel inside the bandwidth
F_i	A flow
PCT_i	The priority percentage for each path
$pathSize_i$	An amount of data inside the path
$total$	An amount of data inside all paths
B	Bandwidth

below

$$\textbf{minimize}\, T_{A_i} + \textbf{maximize}\, \mathcal{U}_{\mathcal{B}^{ef}} \tag{13}$$

$$\textbf{subject to}:$$

$$T_{A_i} \le T_{A_j} \,\text{if}\, \alpha_{A_i} < \alpha_{A_j} \,\text{and}\, \mathcal{P}_{A_i} > \mathcal{P}_{A_j} \tag{13a}$$

$$\forall i \in A_i \quad \forall j \in \mu_j \; \exists (A_i^{\mu_j} \to v_h). \tag{13b}$$

Constraint (13a) specifies that if application A_i arrives before application A_j, i.e., $\alpha_{A_i} \le \alpha_{A_j}$ and the priority of application A_i, $\mathcal{P}_{A_i}$ is higher than the priority of application A_j, $\mathcal{P}_{A_j}$, i.e., $\mathcal{P}_{A_i} > \mathcal{P}_{A_j}$, then the overall execution time for application A_i, T_{A_i} must be less than the execution time for application A_j, T_{A_j}, i.e., $T_{A_i} > T_{A_j}$. Constraint (13b) states that all the microservices of the application $A_i^{\mu_j}$ should be executed in some virtual environment v_h.

C. Complexity Analysis

The knapsack problem can be used to prove other nondeterministic polynomial time (NP)-hard problems by reduction. The knapsack problem is an NP-hard problem that is not solvable in a polynomial time [17]. It is defined as: given a maximum weight capacity W and a set of K items (0, 1, ..., K) each having a weight and value of w_i and v_i, respectively, maximize the sum of the values of the items (**maximize** $\sum_{i=0}^{K} v_i\, x_i$) while the overall sum of the weights is less than or equal to the maximum weight capacity ($\sum_{i=0}^{K} w_i\, x_i \le W$) with an item either selected or not ($x_i \in \{0, 1\}$).

Proposition 1: Finding an optimal subset of applications A_i for a given set of application A is an NP-hard problem.

Proof: The knapsack problem as per the previous definition can be transformed, i.e., reduced, into the simplest form of our problem in a polynomial time. The transformation is as follows. ■

Consider the problem with only a single application component $A_i \in A$, change the item's value v_i to $q_i = 1$ and the weight w_i to δ_i and maximum weight W to budget $\mathcal{B}_i$, with parameter x_i remaining unchanged. The knapsack problem is already strong NP-hard, thus making our problem $\in$ strong NP-hard.

Inherently, as given in proposition 1, finding a solution to the knapsack problem in polynomial time leads to finding a solution to our problem in polynomial time. As no such algorithm exists for any NP-hard problem, therefore, we need a heuristic algorithm to find a solution.

III. Proposed Framework

To solve the problem specified in Section II, we proposed a novel framework that uses two greedy approaches *multiqueue* and *bandwidth slicing*. The details are provided below.

A. Multiqueues

To reduce the queuing time, we used the concept of *multiqueues* where the waiting queue is divided into a set of priority queues. The principal objective of multiqueues is to dynamically distribute and prioritize the incoming data streams according to a fixed number of queues in edge and cloud. Specifically, the key procedure involves ensuring that the best queue for each IoT application is selected based on priority and size of the IoT application. Algorithm 1 presents the procedure involved in the solution, wherein data δ are transmitted from IoT devices and sent to edge devices at a specific time (t).

Subsequently, the first step is the computation of the Score SC for each IoT application A_i, where the Score SC is the final priority score that will be used to divide the data δ in the queues. Thus, we need to find the $ratio_i$ for each δ_i using (14).

$$ratio^{P_A} = \frac{size^{P_A}}{\sum_i^j size^{P_A}} \tag{14}$$

where $ratio_i$ is the process of converting the $size_i$ that has been provided by the user from megabytes to $ratio_i$, where $ratio_i$ $\epsilon\, \{0, 1\}$ and $size$ is the IoT application A size in megabytes. Next, to separate the data to the queues we need to find the Score SC_i for each IoT application A as shown below

$$SC_i = \mathcal{P}_i \times \lambda + (ratio_i \times (1 - \lambda)) \tag{15}$$

Algorithm 1: Multi-Queues.

1 Data δ_i coming from IoT devices that submitted to edge device E_i within the time interval t. Calculate the Score SC_i for each δ_i
2 $SC_i \leftarrow$ using Eq. 15
3 $waitingList \leftarrow$ to δ_i //Buffering all δ_i to a $waitingList$
4 // Add each δ_i to their specific queue Q_i
5 **for** *(each Q_i (Q_{ls},Q_{nm},Q_{lt}))* **do**
6   **for** *($waitingList$)* **do**
7   **if** *($\delta_i.SC_i$ = Q_i.value)* **then**
8             $Q_i \leftarrow \delta_i$
9   **end**
10   // now send the δ_i to the execution to be processed starting with Q_{ls} queue but first check if the node has enough CPUs
11   **if** *($\delta_i.requireCPUs \leq E_i.currentCPUs$)* **then**
12             $execution \leftarrow$ δ_i $waitingList \leftarrow$ remove δ_i
13   **end**
14   **end**
15 **end**

where λ is a static deciding factor among the priority $\mathcal{P}_i$ and δ_i size of the IoT application A_i, where SC_i and priority $\mathcal{P}_i$ $\epsilon \{0, 1\}$, and $\lambda = \{0.8\}$. The Score SC results comes in three types: 1) low priority where the SC $\epsilon \{0.1, 0.3\}$; 2) normal priority where the SC $\epsilon \{0.4, 0.6\}$; and 3) high priority where the SC $\epsilon \{0.7, 0.9\}$. For example, if we have $\mathcal{P}_i = 0.9$, and $ratio_i = 0.5$, then the Score $SC_i = 0.8$, which means that it is a high priority and should be forwarded to the latency-sensitive queue that we will discuss next. Then, we buffered all the data δ and their Scores SC in the $waitingList$ (Lines 2–6). In the second step, each δ is added to the appropriate queue Q depending on their SC. Specially, we have three types of queues Q: 1) latency-sensitive Q_{ls}; 2) normal Q_{nm}; and 3) latency-tolerant Q_{lt}. Last step, we send the δ to the $execution$ to be processed, starting with Q_{ls}, Q_{nm}, then Q_{lt}, according to the $currentCPUs$ in the edge device.

B. Bandwidth Slicing

Bandwidth slicing is primarily designed to slice the bandwidth statically between the $paths$, where $paths$ is the channels inside the bandwidth. The procedure aims to determine the best slicing percentage for the bandwidth based on the priority and data size of each application. Algorithms 2 and 3 illustrate the bandwidth slicing procedure, where algorithm 2 receives flows, computes the score for each of them, and sorts each of them to the queues depending on the score. Then, algorithm 3 slices the bandwidth on the queues depending on the priority type. So, algorithms 2 and 3 complement each other. In detail, the first stage is the receipt of flows F that is sent to either edge devices or the cloud, where each F contains a packet that include one δ_i from one IoT application A_i. After that, the SC for each F is computed using (15) and buffered in the $flowList$ (Lines 7–11). In order to slice the bandwidth, the number of $paths$ must be known. This is determined by checking the priorities of all the F stored in the $flowList$ and identifying the number of $paths$ (Lines 13–22).

In the next stage $slicing()$ procedure is applied as per the details in Algorithm 3, whereby the slicing of the bandwidth is based on the number of available $paths$. There are two types of slicing, the first takes place when there is only one type of $path$ (e.g., lt, nm, ls, etc.), after which the entire bandwidth is given to that $path$ (Lines 4–10). The second type of slicing occurs where there is more than one type of $path$ (e.g., lt and nm, or lt and ls, or ls and nm, or lt, nm, and ls, etc.). Subsequently, the $weightedAverage$ for each $path$ is calculated using (16) and multiplied by the $availableBw$. After this, the bandwidth is divided among the $paths$ in line with the $weightedAverage$ for each $path$, with the largest percentage of the bandwidth being

Algorithm 2: Bandwidth Slicing.

Input: ls, nm, lt: priority types, $total$: number of flows, $availableBw$: available bandwidth, $usedBw$: used bandwidth, $weightedAverage$: compute the average between multiple $paths$
1 Received flows F contains δ to be sent to node E_i.
2 Calculate the Score SC for each flow F
3 $SC_F \leftarrow$ using Eq. 15
4 $flowList \leftarrow$ to F //Buffering all F to a $flowList$
5 // Count the types of $paths$
6 **for** *($flowList$)* **do**
7     $path_i \leftarrow F_i.SC_i$
8   **switch** $path$ **do**
9   **case** 0 **do**
10             $lt \leftarrow path$
11   **end**
12   **case** 1 **do**
13             $nm \leftarrow path$
14   **end**
15   **case** 2 **do**
16             $ls \leftarrow path$
17   **end**
18   **end**
19 **end**
20 $total$=$flowList$.size
21 **slicing()**

Algorithm 3: Slicing.

1 **if** *($total$==0)* **then**
2     $usedBw$ = 0
3 **end**
4 **else**
5   **switch** *(lt, nm, ls)* **do**
6   **case** *(lt ! = 0)* **do**
7             $usedBw$ = $availableBw$ / lt
8   **end**
9   **case** *(nm ! = 0)* **do**
10             $usedBw$ = $availableBw$ / nm
11   **end**
12   **case** *(ls ! = 0)* **do**
13             $usedBw$ = $availableBw$ / ls
14   **end**
15   **case** *(lt & nm ! = 0)* **do**
16             $weightedAverage_{lt,nm} \leftarrow$ using Eq. 16
17             $usedBw$ = $availableBw$ * $weightedAverage_{lt,nm}$
18   **end**
19   **case** *(lt & ls ! = 0)* **do**
20             $usedBw$ = $availableBw$ * $weightedAverage_{lt,ls}$
21   **end**
22   **case** *(ls & nm ! = 0)* **do**
23             $usedBw$ = $availableBw$ * $weightedAverage_{ls,nm}$
24   **end**
25   **case** *(lt & nm & ls ! = 0)* **do**
26             $usedBw$ = $availableBw$ * $weightedAverage_{lt,nm,ls}$
27   **end**
28   **end**
29 **end**

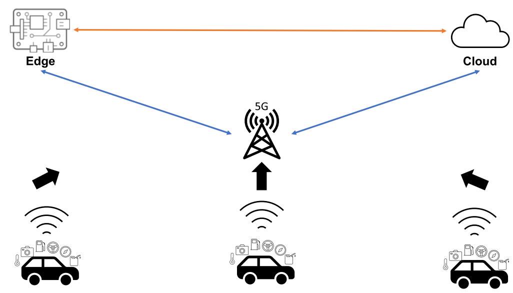


Fig. 2. Data transfer and processing in self-driving cars.

allocated to the $ls\,path$, followed by the $nm\,path$ and the $lt\,path$ (Lines 11–24).

$$weightedAverage = \sum_{i}^{j} \frac{\mathcal{PCT}_i * pathSize_i}{total}. \quad (16)$$

Equation (16) shows the weighted average for each $path$, where i and $j \in \{ls, nm, lt\}$, and $\mathcal{PCT}_i$ is the priority percentage for each $path$ that will be defined by the user. Then, we have a $path$ size that clarifies how many δ_i inside it is represented by $pathSize_i$. Lastly, we have $total$ that represents the total number of δ_i inside all $paths$.

$$\mathcal{NU}\% = \frac{size_i * 100}{availableBw * \Delta t}. \quad (17)$$

Equation (17) shows the network utilization for each $path$, where $size_i$ in bits is multiplied by 100 and divided by $availableBw$ multiplied by the Δt time interval.

IV. Evaluation

In this section, we evaluate our proposed work on a self-driving car test case.

A. Experiment Setup

1) Test Case: Fig. 2 gives a basic architecture of the deployment in a self-driving car. Cars with self-driving systems are contemporary technology in which each car has numerous sensors, cameras, radars, speed controllers, etc. Each sensor will exchange its data with the SDN controller that is located in low-latency 5G towers. The controller will make the decisions about the routing and the priority of the data exchanged. In addition to this, SD-WAN ensures a smooth network traffic flow from and to the self-driving cars and enables the development of self-driving cars being at the same time smarter and safer [18]. Edge and cloud datacenters which are situated at different locations in the city will send the data received from the smart cars to be processed in the host machines residing in the datacenters. For additional processing, the data are sent to other host machines in different edge and cloud datacenters via the controllers and respond back to make runtime decisions. Moreover, communications are also established for edge-to-edge and cloud-to-cloud through the SD-WAN network. Also, the application's response and processing time requirements need to be guaranteed.

In this scenario, a car's IoT device captures raw data and is assigned a priority. Based on the priority, each data packet is

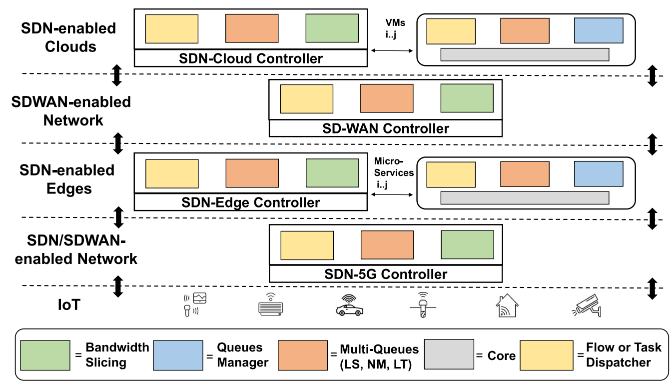


Fig. 3. Scenario process in our IoT edge-cloud environment.

TABLE II
Test Case Configuration

IoT device		Edge device		Host (edge)		VM (cloud)	
IoT type	Car	Edge type	Raspberry Pi	Storage	640 GB	Storage	10 GB
Max BW	100 Mbps	Max BW	100 Mbps	Max BW	10000 Mbps	Max BW	1000 Mbps
Required CPUs	10	Pes	10	Pes	4	Pes	4
Network	5G	RAM size	10000	RAM size	32000	RAM size	512
Max battery cap	100 mAh	MIPS	250	MIPS	1250	MIPS	250

TABLE III
Infrastructure Device Configuration

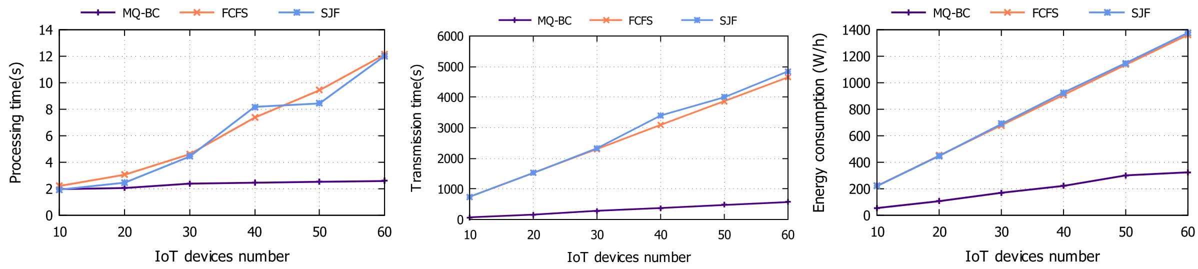
Number of IoT Devices	Number of edge devices	Number of hosts	Number of VMs
10-60	2	2	2

ranked and sent to an edge datacenter. When the data packet arrives at the edge datacenter, it is sorted and buffered into different queues depending on data priority and size, and then sent to the edge devices for processing. Next, the data is sent to cloud datacenters through the SD-WAN in the 5G towers for further processing. The SD-WAN controller buffers the data to make the best and fastest route and slices the bandwidth to fit the data. In the cloud datacenter, same as the edge datacenter, the data will be sorted and buffered in different queues depending on the priority and size of each piece of data to be sent to the VMs for processing. Fig. 3 shows the detailed illustration of the whole process.

2) Configuration: We model the scenario using the open-source simulator *IoTSim-Osmosis* [19]. Table II shows the specific configuration details for the given test case. We vary the number of IoT devices from 10 to 60 for the given test case. The details about the number of devices are given in Table III. We compared our results with two approaches, first come first serve (FCFS) and shortest job first (SJF) methods.

B. Experiment Results

This section presents the results of our proposed multi-queues bandwidth slicing (MQ-BC) approach. Fig. 4(a) shows the average processing time of each test as compared to the FCFS and SJF. As shown in the figure, our proposed approach achieves an average gain of 71% as compared to FCFS and 73% compared to SJF. The trend is also followed for the transmission time with 49% savings as compared to the FCFS and 74% with SJF as shown in Fig. 4(b). The trend is also followed for the queue waiting time with 164% savings as compared to the FCFS

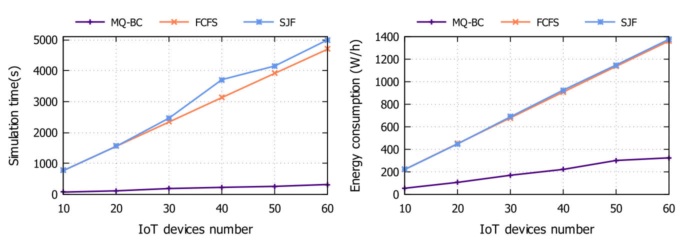


(a) (b) (c)

Fig. 4. Experiment results. (a) Processing time. (b) Transmission time. (c) Queue waiting time.

TABLE IV
COMPARATIVE TABLE FOR THE RESULTS

	Processing time			Transmission time			Queues waiting time		
IoT Device	**MQ-BC**	**FCFS**	**SJF**	**MQ-BC**	**FCFS**	**SJF**	**MQ-BC**	**FCFS**	**SJF**
10	1.9	2.21	1.93	73	748	749	36	105	710
20	2.05	3.06	2.46	160	1526	1527	88	474	1427
30	2.38	4.60	4.42	286	2308	2331	140	978	2140
40	2.45	7.36	8.17	375	3095	3401	192	1713	2858
50	2.52	9.43	8.43	475	3870	3996	240	2368	3571
60	2.58	12.11	12.01	571	4648	4842	292	3259	4284



(a) (b)

Fig. 5. Scalability results. (a) Simulation time. (b) Energy consumption.

and 98% with SJF as shown in Fig. 4(c). Table IV shows a comparison of the results in detail between FCFS, SJF, and our proposed MQ-BC policies.

1) Scalability Result: Fig. 5(a) shows the average simulation time of each test as compared to the FCFS and SJF. As presented in the figure, our approach achieves an average gain of 143% as compared to the FCFS and 149% compared to SJF. Finally, Fig. 5(b) shows the average energy consumption of each test as compared to the FCFS and SJF. As presented in the figure, our approach achieves an average gain of 24% as compared to the FCFS and similar 24% compared to SJF.

In summary, the proposed system makes a significant improvement compared with FCFS and SJF in edge and cloud. It decreases the processing time up to four times and the transmission time in the network from the IoT device to the cloud via edge and SD-WAN up to nine times. Besides the improvements in data processing and transmission times, it is noted that the new system policies contribute to decreasing energy consumption by three times. The more the data, the greater the improvements in both time and energy.

C. Network Utilization

This section describes the network utilization measurement results for all systems from the start to the end of the simulation using (17). Fig. 6 shows the network utilization percentage for FCFS, SJF, and MQ-BC policies. It can be seen that at the beginning, it started to use 100% of the network when it was sending data from IoT devices to microservices. Then immediately after that, it drops to 0% because the data had arrived at destination microservices and started the processing phase. Next, 100% was used from the network, because microservices started to send the data to the cloud. Finally, it drops again to 0% because the data had arrived at destination VMs and started the processing phase. However, our proposed system shows the same way of using the network as in the previous systems, but it decreases the overall time of network usage. So, this illustrates that our system improved the time of network utilization by up to 7 times and 7.5 times compared with the FCFS and SJF systems, respectively.

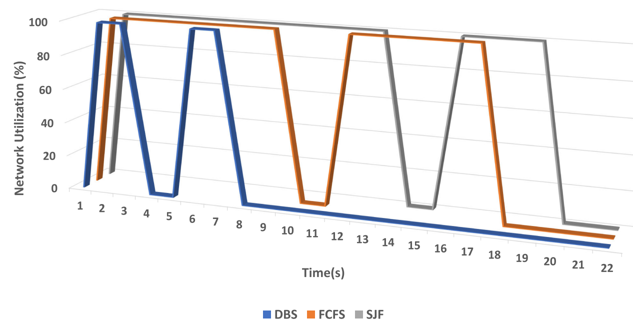


Fig. 6. Comparing the network utilization for the three policies.

D. Auto-Adaptation

Although the results so far show promising optimal performance, sometimes bandwidth static slicing can lead to a degradation in the network utilization. Fig. 7(a) shows an example of how such problems might arise. Note that setup and configuration is similar to the previous experiment but with only ten IoT devices. The Fig. 7(a) has 100 MB of bandwidth, where it is sliced/divided into three parts: 1) 70% is assigned to the latency-sensitive (ls) $path$; 2) 20% is given to the normal (nm) $path$ and 3) 10% is assigned to the tolerant-sensitive (lt) $path$. Suppose that the ls $path$ receives 30 MB of data every second, nm $path$ receives 70 MB of data every second, and lt $path$ receives 100 MB of data every second (as shown in the figure). If the ls $path$ is only using 30 MB per second, then 40% of its sliced network would be wasted. As such, this article contributes

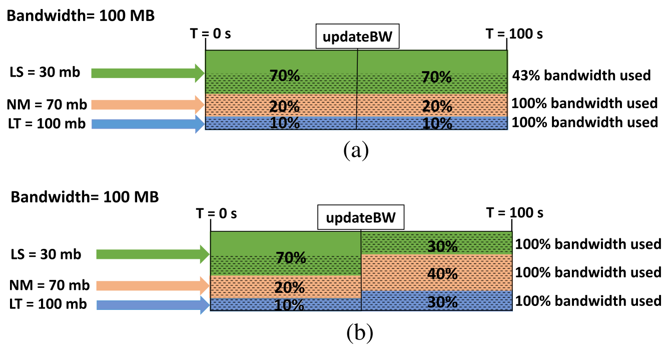


Fig. 7. Auto-adaptation example. (a) Without Auto-adaptation. (b) With Auto-adaptation.

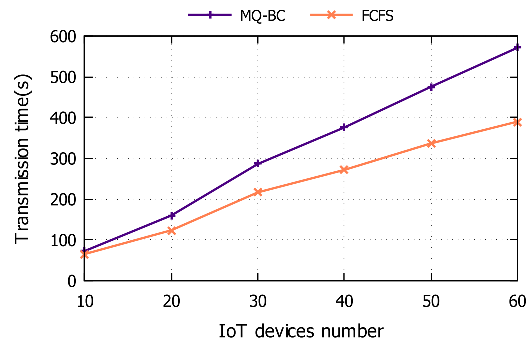


Fig. 8. Auto-adaptation transmission time.

Algorithm 4: Auto-Adaptation.

Input: $minPCT$: The minimal percentage for Auto-Adaptation, $newWeightedAverage_i$: The $weightedAverage_i$ with the new PCT_i, $oldWeightedAverage_i$: The $weightedAverage_i$ that computed in Alg.3

$pathRatio_{lt,nm,ls} \leftarrow$ using Eq. 18 // Measures the network utilization $\mathcal{NU}$ for all $paths$
if $(pathRatio_i \geq minPCT)$ **then**
  $usedBw = availableBw * newWeightedAverage_i$
end
else
  $usedBw = availableBw * oldWeightedAverage_i$
end
return $usedBw$

to solving this problem by proposing an auto-adaptive network slicing algorithm. The algorithm is designed to dynamically tune the network slicing percentage based on the network utilization of each $path$, as shown in Fig. 7(b).

$$pathRatio = \frac{pathFlows_i}{total}. \tag{18}$$

Equation (18) shows the $pathRatio$ of each path, where $pathFlows_i$ is the F numbers of q_i, and q_i is one of our proposed paths (ls, nm, lt), divided by the $total_i$ number of flows in that $path$.

Every $path$ receives multiflows every second, depending on the data coming from IoT Devices. So, after computing the number of flows that are used in every $path$, we use it in our Algorithm 4 to calculate the new percentage for every $path$ every time the bandwidth is updated in the system.

The main goal of auto-adaptation is to dynamically allocate the percentage of $paths$ in the bandwidth slicing mechanism. Thus, the procedure seeks the optimal slicing percentage for the bandwidth based on the network utilization $\mathcal{NU}$ for each $path$. Algorithm 4 clarifies the auto-adaptation procedure, which starts by measuring the $\mathcal{NU}$ for each $path$ as per (17). Following this, the resulting percentage $pathRatio$ is compared with the minPCT defined by the user. If the $pathRatio$ is equal to or bigger than the $minPCT$, the new $pathRatio$ is employed in the $weightedAverage$ using (16) to comprise an improved percentage that improves the bandwidth slicing between the $paths$. If the $pathRatio$ is smaller, the static percentage that was previously employed to the $path$ will be utilized. Fig. 8 shows the results of a comparison between the MQ-BC system and the MQ-BC system with the auto-adaptive network slicing algorithm, which showed an improvement in the transmission time of 46%.

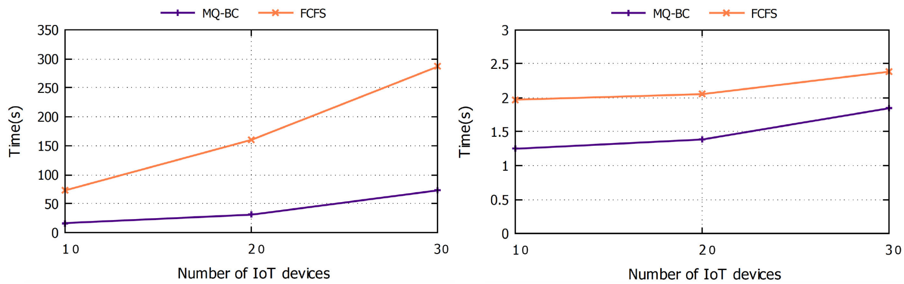
V. Further Evaluation and Validation

We not only evaluate our proposed system in a simulation-based environment, but also validate it in a real-world IoT-based SDN environment. We used real edge hardware devices (three Raspberry Pis, one SDN-enabled switch, and one laptop). We use sensor emulators in order to mimic IoT devices and generate IoT data. We ran the sensor emulator in one Raspberry Pi, an edge processor emulator in the second Pi, and a VM in the third Pi. The Raspberry Pis come with 1.4 GHz 4 cores and 1 GB RAM. On the networking side, we ran an Open vSwitch on a Linux-based switch with an Intel N3700 Processor and 8 GB RAM. We ran a Ryu controller as an SDN controller on the laptop with Intel 4 cores i7-8565 U 1.99 GHz CPU and 16 GB RAM.

Workload and Dataset: We use a real-world smart building (Urban observatory, Newcastle University) dataset to generate a realistic workload. This dataset consists of samples that are collected from temperature, NO2, and gas, etc. We used the message queuing telemetry transport (MQTT) protocol to send and receive the data between the devices. We applied our multiqueues policy on the edge emulator and the VM, to prioritize the data depending on the priority of each sensor. Also, we implemented the bandwidth slicing policy in the SDN controller to manage the bandwidth in the network between the devices.

Methodology: We have three applications for testing, 10, 20, and 30 sensors. Starting from the sensor emulator, we set the input rate to 10–30 record/s, then, the records will be sent to the edge. Next, in the edge, the data will be sorted through the multiqueues policy to be processed by the edge. Then, after processing is finished the data will be sent to the VM via the switch. The switch will redirect the data to the SDN controller, it will manage the routing and the bandwidth slicing depending on the priority, then, it will send the data to the VM for further analysis. The VM will sort and process the data.

Results: We measured the average transmission time starting from the sensor via the switch until it reached the VM, for all three apps, and then we compared it with the average transmission results from the simulation experiment. Also, we measured



(a) (b)

Fig. 9. Validation results. (a) Average transmission time. (b) Average processing time.

TABLE V
COMPARISON OF VARIOUS SCHEDULING SYSTEMS WITH THE ONE PROPOSED

Systems	Features								
	Cloud processing	SDN support	Auto-Adaptation	BW slicing	stream processing	Queuing delay	Edge processing	IoT devices	Latency
LEO [20]	✓					✓			✓
MAUI [21]	✓				✓	✓	✓		✓
Frontier [22]			✓		✓	✓	✓	✓	✓
Approxiot [23]			✓		✓		✓	✓	✓
Nebulastream [24]	✓				✓			✓	✓
Homa [25]	✓					✓			✓
pHost [26]						✓			✓
NDP [27]						✓			✓
SDQ [28]		✓				✓	✓		✓
NS [29]		✓		✓		✓			✓
QJUMP [15]	✓					✓			✓
Proposed	✓	✓	✓	✓	✓	✓	✓	✓	✓

the average processing time in the edge and the VM for all three apps, and then we compared it with the average processing time results from the simulation experiment. As shown in Fig. 9, the higher the number of sensors, the higher the processing and transmission time. As a result, it can be seen that the results have a positive correlation, which reveals that the accuracy and correctness of our simulation-based results are comparable to the real IoT-based SDN environment.

VI. RELATED WORK

There has been much prior work on related topics, such as cloud task offloading, IoT stream processing, bandwidth slicing, and congestion control. In the field of mobile computing, general-purpose offloading refers to offloading tasks to the cloud. It is necessary to consider the influence of different computation resources on data transmission. LEO [20] has optimized energy consumption through performing various multiple-sensor processing tasks on mobile devices. Nonetheless, the dynamicity of the IoT network was not considered. MAUI [21] does not take into account the queuing delay from edge to cloud even though they are deemed to be diverse resources. Thus, it would be advantageous for these networks to be improved by our system.

Advancements to edge computing have moved cloud-based data processing towards the ground, which has led to a substantial reduction in process latency. The researchers in [22] proposed an edge-based stream processing system to process data from multiple IoT devices in parallel. Nonetheless, the key objective of this new system is to enhance the reliability to changes in wireless network conditions rather than addressing bandwidth slicing issue. Moreover, the authors in [23] concentrate on enhancing analytical task performance instead of taking into account queuing delays when processing stream data. NebulaStream [24] is a platform that directs data streams towards different processing tasks for specific data-flow programs using application programming interfaces (APIs). Nonetheless, this system is unable to differentiate between the latency sensitivity of different IoT applications. Therefore, it cannot manage queue delays. Our proposed system is considered an effective traffic scheduling system that can be used throughout the IoT edge-cloud continuum environments, especially those with different types of data records and specific QoS requirements. In the field of networking, software defined queuing (SDQ) [28] proposed a solution that selects the best queue and route for each incoming flow to decrease network workload imbalances. However, the cloud network and the bandwidth slicing were not considered. In network slicing (NS) [29], the authors proposed a network slicing-based communication solution. However, the bandwidth slicing, edge, and cloud processing were not considered.

Congestion control is a common feature throughout the network community and is usually achieved by restricting the transmission rate and sending network packets to destinations. The queues JUMP (QJUMP) system [15] enables messages to be forwarded to different queues according to their priority levels. This is very similar to the multilevel queues management feature in our system. However, QJUMP does not support stream data processing applications. Several receiver-driven flow-control systems (including Homa [25], pHost [26], and novel datacenter protocol (NDP) [27]) can effectively reduce the latency of small-scale messages, but such networks contain switch-based mechanisms that are based primarily on an assumption that ingress throughput and egress throughput are equal. Thus, they are considered to be invalid in the IoT edge-cloud continuum. Our system effectively combines dynamic bandwidth allocation and holistic traffic coordination at the application layer. It is thus sufficiently flexible to enable throughput throttling and bandwidth adjustments during data streaming processes. The detail properties of recent and our proposed systems are compared in Table V.

VII. Conclusion

This article presented a novel distributed and QoS-based multilevel queues traffic scheduling system. This system was designed to maintain the general system throughput while diminishing the queuing delay and increasing the QoS assurance of applications with high latency. Our scheduling system relied on multilevel queues for incoming traffic depending on their latency sensitivity. It also relied on bandwidth slicing, which divides the bandwidth of the network on the incoming traffic depending on their latency sensitivity. Moreover, the bandwidth slicing of our system was synchronously auto-tuned by analyzing network utilization at the time. Using these two methodologies in our system greatly enhanced latency and throughput for edge-cloud environments. The results showed that the processing latency in edge and cloud hosts has been reduced by up to 4× and the network by up to 9× comparing with the state-of-the-art (i.e., FCFS and SJF). In addition, the energy consumption of edge and cloud hosts and the network has been reduced by 3×. In future work, we will focus on advanced and more complex algorithms aimed to find the optimal solution for the bandwidth slicing problem.

References

[1] A. Botta, W. De Donato, V. Persico, and A. Pescapé, "Integration of cloud computing and Internet of Things: A survey," *Future Gener. Comput. Syst.*, vol. 56, pp. 684–700, 2016.

[2] 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.

[3] W. Rafique, L. Qi, I. Yaqoob, M. Imran, R. U. Rasool, and W. Dou, "Complementing IoT services through software defined networking and edge computing: A comprehensive survey," *IEEE Commun. Surv. Tut.*, vol. 22, no. 3, pp. 1761–1804, Jul.–Sep. 2020.

[4] C.-L. Hu, L.-X. Kuo, Y.-H. Chen, T. Tantidham, and P. Mongkolwat, "QoS-prioritised media delivery with adaptive data throughput in IoT-based home networks," *Int. J. Web Grid Serv.*, vol. 17, no. 1, pp. 60–80, 2021.

[5] K. Ha *et al.*, "You can teach elephants to dance: Agile VM handoff for edge computing," in *Proc. 2nd ACM/IEEE Symp. Edge Comput.*, 2017, pp. 1–14.

[6] T. Olsson, E. Lagerstam, T. Kärkkäinen, and K. Väänänen-Vainio-Mattila, "Expected user experience of mobile augmented reality services: A user study in the context of shopping centres," *Pers. Ubiquitous Comput.*, vol. 17, no. 2, pp. 287–304, 2013.

[7] H. Lasi, P. Fettke, H.-G. Kemper, T. Feld, and M. Hoffmann, "Industry 4.0.," *Bus. Inf. Syst. Eng.*, vol. 6, no. 4, pp. 239–242, 2014.

[8] A. P. Plageras, K. E. Psannis, C. Stergiou, H. Wang, and B. B. Gupta, "Efficient IoT-based sensor Big Data collection-processing and analysis in smart buildings," *Future Gener. Comput. Syst.*, vol. 82, pp. 349–357, 2018.

[9] T. Buddhika and S. Pallickara, "Neptune: Real time stream processing for Internet of Things and sensing environments," in *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, 2016, pp. 1143–1152.

[10] P. Bonte, R. Tommasini, E. D. Valle, F. De Turck, and F. Ongenae, "Streaming massif: Cascading reasoning for efficient processing of IoT data streams," *Sensors*, vol. 18, no. 11, 2018, Art. no. 3832.

[11] M. Chowdhury and I. Stoica, "Coflow: A networking abstraction for cluster applications," in *Proc. 11th ACM Workshop Hot Topics Netw.*, 2012, pp. 31–36.

[12] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom, "Models and issues in data stream systems," in *Proc. 21st ACM SIGMOD-SIGACT-SIGART Symp. Princ. Database Syst.*, 2002, pp. 1–16.

[13] C.-Y. Wan, S. B. Eisenman, and A. T. Campbell, "Coda: Congestion detection and avoidance in sensor networks," in *Proc. 1st Int. Conf. Embedded Netw. Sensor Syst.*, 2003, pp. 266–279.

[14] A. A. Rabileh, K. A. A. Bakar, R. R. Mohamed, and M. A. Mohamad, "Enhanced buffer management policy and packet prioritization for wireless sensor network," *Int. J. Adv. Sci., Eng. Inf. Technol.*, vol. 8, no. 4, pp. 1770–1776, 2018.

[15] P. Matthew *et al.*, "Queues {don't} matter when you can {JUMP} them!," in *Proc. 12th {USENIX} Symp. Netw. Syst. Des. Implementation*, 2015, pp. 1–14.

[16] M. M. Hasan, S. Kwon, and J.-H. Na, "Adaptive mobility load balancing algorithm for LTE small-cell networks," *IEEE Trans. Wireless Commun.*, vol. 17, no. 4, pp. 2205–2217, Apr. 2018.

[17] R. M. Karp, "Complexity of Computer Computations," in *Proc. Symp. Complexity Comput. Comput.*, 1972, pp. 85–103.

[18] J. Pisarov and G. Mester, "The impact of 5G technology on life in 21st century," *IPSI BgD Trans. Adv. Res.*, vol. 16, no. 2, pp. 11–14, 2020.

[19] K. Alwasel *et al.*, "Iotsim-osmosis: A framework for modeling and simulating IoT applications over an edge-cloud continuum," *J. Syst. Architecture*, vol. 116, 2021, Art. no. 101956.

[20] P. Georgiev, N. D. Lane, K. K. Rachuri, and C. Mascolo, "Leo: Scheduling sensor inference algorithms across heterogeneous mobile processors and network resources," in *Proc. 22nd Annu. Int. Conf. Mobile Comput. Netw.*, 2016, pp. 320–333.

[21] E. Cuervo *et al.*, "Maui: Making smartphones last longer with code offload," in *Proc. 8th Int. Conf. Mobile Syst., Appl., Serv.*, 2010, pp. 49–62.

[22] D. O'Keeffe, T. Salonidis, and P. Pietzuch, "Frontier: Resilient edge processing for the Internet of Things," in *Proc. VLDB Endowment*, 2018, pp. 1178–1191.

[23] Z. Wen *et al.*, "ApproxIot: Approximate analytics for edge computing," in *Proc. IEEE 38th Int. Conf. Distrib. Comput. Syst.*, 2018, pp. 411–421.

[24] S. Zeuch *et al.*, "The NebulaStream platform: Data and application management for the Internet of Things," in *Proc. 10th Biennial Conf. Innovative Data Syst. Res.*, 2020.

[25] B. Montazeri, Y. Li, M. Alizadeh, and J. Ousterhout, "Homa: A receiver-driven low-latency transport protocol using network priorities," in *Proc. Conf. ACM Special Int. Group Data Commun.*, 2018, pp. 221–235.

[26] X. Peter *et al.*, "pHost: Distributed near-optimal datacenter transport over commodity network fabric," in *Proc. 11th ACM Conf. Emerg. Netw. Experiments Technol.*, 2015, pp. 1–12.

[27] M. Handley *et al.*, "Re-architecting datacenter networks and stacks for low latency and high performance," in *Proc. Conf. ACM Special Int. Group Data Commun.*, 2017, pp. 29–42.

[28] A. N. Abbou, T. Taleb, and J. S. Song, "A software-defined queuing framework for QoS provisioning in 5G and beyond mobile systems," *IEEE Netw.*, vol. 35, no. 2, pp. 168–173, Mar./Apr. 2021.

[29] H. Khan, P. Luoto, S. Samarakoon, M. Bennis, and M. Latva-Aho, "Network slicing for vehicular communication," *Trans. Emerg. Telecommun. Technol.*, vol. 32, no. 1, 2021, Art. no. e3652.