Pro-Edge: A Programmable Edge Network Architecture for Industrial Internet of Things

Nurzaman Ahmed [1], Mehbub Alam [2], Rakesh Matam [2], Ferdous Ahmed Barbhuiya [2], and Mithun Mukherjee [2]

[1]Indian Institute of Science
[2]Affiliation not available

October 30, 2023

Abstract

Internet of Things (IoT) with edge computation enhances efficiency, safety, and availability of an industrial automation system. However, there is a continued effort to increase the reliability of the system with minimal downtime. This can be achieved through a modular, re-configurable, and integrable system design approach. In this paper, we propose Pro-Edge, a programmable edge network to reconfigure different services associated with industrial applications and networks. Pro-Edge employs programmable layers at the edge for re-configuring the sensor/actuator network and applications. The lowermost layer allows to reconfigure the communication related parameters and the middle layer consists of a Software-Defined Network (SDN) controller that can dynamically program different modules, handling actuation decisions from the edge. An interfacing protocol between the layers is proposed to provide reliability by considering the optimal configuration parameters among the layers. As a top-layer, a priority forwarding mechanism is designed for SDN core communication in case the sensor and actuator are in different edges. Pro-Edge significantly improves the actuation-latency and is highly energy efficient compared to the existing state-of-the-art.

Pro-Edge: A Programmable Edge Network Architecture for Industrial Internet of Things

Mehbub Alam, *Student Member, IEEE*, Nurzaman Ahmed, *Member, IEEE*, Rakesh Matam, *Member, IEEE*, Mithun Mukherjee, *Senior Member, IEEE*, and Ferdous Ahmed Barbhuiya, *Member, IEEE*

***Abstract*—Internet of Things (IoT) with edge computation enhances efficiency, safety, and availability of an industrial automation system. However, there is a continued effort to increase the reliability of the system with minimal downtime. This can be achieved through a modular, re-configurable, and integrable system design approach. In this paper, we propose *Pro-Edge*, a programmable edge network to reconfigure different services associated with industrial applications and networks. *Pro-Edge* employs programmable layers at the edge for re-configuring the sensor/actuator network and applications. The lowermost layer allows to reconfigure the communication related parameters and the middle layer consists of a Software-Defined Network (SDN) controller that can dynamically program different modules, handling actuation decisions from the edge. An interfacing protocol between the layers is proposed to provide reliability by considering the optimal configuration parameters among the layers. As a top-layer, a priority forwarding mechanism is designed for SDN core communication in case the sensor and actuator are in different edges. Pro-Edge significantly improves the actuation-latency and is highly energy efficient compared to the existing state-of-the-art.**

***Index Terms*—Edge computing, SDN, Fog Computing, Internet of Things, Programmable Edge, Industrial IoT, WSAN.**

I. Introduction

Internet of Things (IoT) enabled industrial automation system involves control-loop communication [1] where an actuator can be controlled wirelessly. A fractional violation of designated constraints (in terms of latency or availability) in such an automation and control system can result in performance drop or even system outage. Applications related to machine control, system health, and electrical control systems popularly use IEEE 802.11/802.15.4-based wireless communication. Further, to ensure the Quality of Service (QoS) requirements, SDN can be employed to meet critical application needs. It separates the network into data and control planes for managing network resources precisely [2]. However, the current SDN based IoT architectures have no special measures for control-loop traffic, which if considered is shown to increase reliability, safety, and uptime of an industrial automation system. In this paper, we propose *Pro-Edge*, a programmable edge network to dynamically reconfigure different services associated with applications and networks. Fig. 1 shows the overview of the proposed solution that considers a programmable approach to reconfigure applications modules (e.g., different actuation policies) and wireless/wired networks (e.g., channels, routing, topology). The programmability feature at the edge and forwarding devices increases the uptime and reduces the actuation time in control-loop communication.

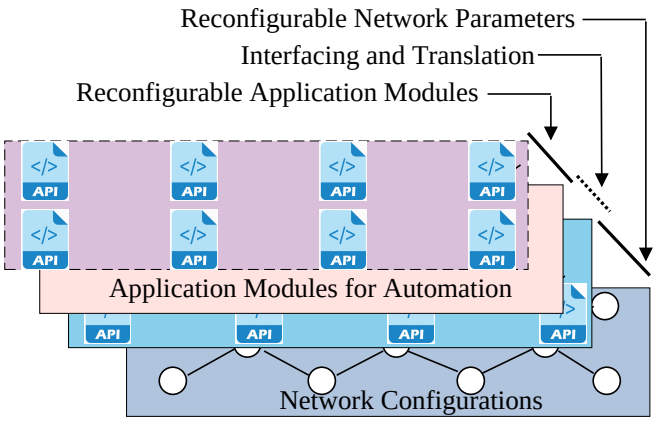


Fig. 1: Overview of the proposed scheme

A. Motivations

IoT systems are widely adopted to automate industrial processes, especially to deal with time-sensitive applications [3]. Typically, latency related challenges can be dealt using SDN. A large number of gateway devices with computing and storage capabilities are being deployed for controlling and provisioning of network infrastructure in IoT. Such gateway devices on the edge can not only provide computing facilities but also support SDN interfacing for enabling core network connectivity. The edge performance can be enhanced by adopting the following:

- Computing on the cloud and edge enable real-time responses, cost reduction, and ensuring the security of data and systems [4], [5].
- A large number of low-cost computing devices can be used at the edge for localized processing and decision making.
- Multiple wireless sensor and actuator networks (WSANs) use different gateways to forward their traffic-flows to the SDN core network. Different edge nodes are also

M. Alam, R. Matam, and F. A. Barbhuiya are with the Department of Computer Science & Engineering, Indian Institute of Information Technology Guwahati, India, 781015.

N. Ahmed is with the Department of Computer Science & Engineering, Indian Institute of Technology Kharagpur, India, 721302.

M. Mukherjee is with School of Artificial Intelligence, Nanjing University of Information Science and Technology, China. E-mail: mehbub@iiitg.ac.in, nurzaman@cse.iitkgp.ac.in, rakesh@iiitg.ac.in, m.mukherjee@ieee.org, ferdous@iiitg.ac.in

part of the network view available at the SDN controller. Considering this, an edge controller along with an SDN controller can dynamically program the edge and WSAN for reliable and low latency industrial IoT operations.
- The resources at the edge are crucial for dynamically re-configuring the application servers according to the applicants' requested services. Moreover, reconfiguration of network parameters such as channels, and routing are essential for reliable networking.
- For adopting intelligence techniques at the edge, programmability can dynamically tailor a trained module as per requirements.

A modular, re-configurable and integrable network architecture for IIoT requires to adopt the above mention technologies and solutions. Existing works in this direction present various strategies to enhance different performance metrics, but are mostly application specific. In other words, most of the approaches lack flexibility. To increase the re-usability of network resources and flexibility of network configurations, a programmable edge platform is important. Network virtualisation and slicing allows a single physical network to be logically divided into distinct service levels. These mechanisms facilitate the network to support diverse requirements of different IoT applications [6]. However, a platform is required to efficiently program different communication and computational parameters.

B. Contributions

To overcome the challenges of reliable and low latency actuation, the key *contributions* of this work are as follows:

- We propose a programmable edge to reconfigure different services associated with applications and networks. Our solution creates two programmable layers at the edge for reconfiguring the sensor/actuator/core network and applications, respectively.
- A QoS-aware data forwarding mechanism is proposed for SDN-based communication in case the sensor and the actuator are in different networks.
- We propose a task programming approach to solve the issue of high latency during offloading in the case if the task module is not available in the edge node.

II. Related Works

In the industrial IoT (IIoT) paradigm about the impact of edge computing, a lot of relevant researches can be found in the literature along with the advancements based on learning, prediction, and Blockchain [7]–[10]. In this context, intelligent and accurate resource management by Artificial Intelligence (AI) has a direction which also challenging towards the automated platform. Moreover, resource management is a crucial factor for IIoT. The works in [11]–[13] have modelled different strategies using AI to solved resource management issues up to a certain extent.

Mahmud *et al.* [14] proposed a re-configuration mechanism in IoT through programmability feature using an SDN-based architecture to achieve improved outcomes considering a large-scale IoT scenario. Their proposed IoT virtualization technique (Flow-sensor) aims to reach the sensor nodes when some of the nodes are out of state and inactive for a longer period.Costanzo *et al.* [15] proposed an SDN-enabled IoT architecture for WSN (SDWN) in order to simplify the network configuration and resource management of the system. The authors emphasized SDN integration on WSN networks, mostly its impact on wireless infrastructure-less networking.

When task type changes , it is not always feasible to pick up the devices physically and program them again for different tasks; re-programmability and policy changing technique is needed as a solution. The authors in [16], proposed a scheme to address the issues of network management and policy changing mechanism. They adopted an SDN-based solution named Software-Defined Wireless Sensor Network (SD-WSN) for reprogramming features in the network topology. Sensor OF protocol which is extended from OpenFlow (OF), is used in their scheme.

To integrate IPv6 over low-power wireless personal area networks (6LoWPAN) with SDN core networks, Das *et al.* [17] proposed a mechanism that tackles the issues faces in IoT. In this scheme, the authors highlighted the challenges of high latency, heterogeneity, and packet loss. In [18] and [19], the authors designed two SDN-based wireless sensor network management systems for the purpose of device and network management. For Soft-WSN in [18], without considering network delay and control overhead in the scheme arise questions of efficiency of the system.

Adopting programmable network techniques using SDN and network function virtualization (NFV) for mobile traffic in IoT networks, the authors in [20] proposed an SD-NFV scheme to reduce the end-to-end delay and improve energy consumption at the sensor layer devices. A detailed testbed implementation of SD-NVF is provided in support of the claim that says improvements of 5%–14% in the network data delivery ratio and 70% in the sustainability of 6LoWPAN node operational comparing to traditional approaches. The testbed consists of two nodes, namely *Simple 6LoWPAN* and *Advance* nodes, along with a border gateway. The workflow of the architecture is as follows– upon collecting the sensor data, it forwards to the advanced node; then, from the advanced node, it is further transmitted to the border gateway. The functionalities of the border gateway device includes the job of the SDN controller and coordinator of the network.

μSDN and SD-6LoWPAN are the two architecture proposed in [21], and [22] aiming for flexible network management and also to achieve low control overhead in the network. Implementing both the schemes in Contiki operating system, the parameter considered in μSDN are latency, energy, and packet delivery, whereas SD-6LoWPAN is measured in terms of average latency and overhead control service. However, the drawback of these two schemes is that it takes higher processing delay due to the controller adopter and discovery module. Due to specialized agents, it is not an optimal adaptable solution in any network.The open-source implementation platforms for WSN-based SDN architectures are presented in literature such as TinySDN [23], [24] , CoAP-SDAN [25] etc.

In [26], Rahman *et al.* proposed a mechanism to dynamically reconfigurable edge nodes to provide different services

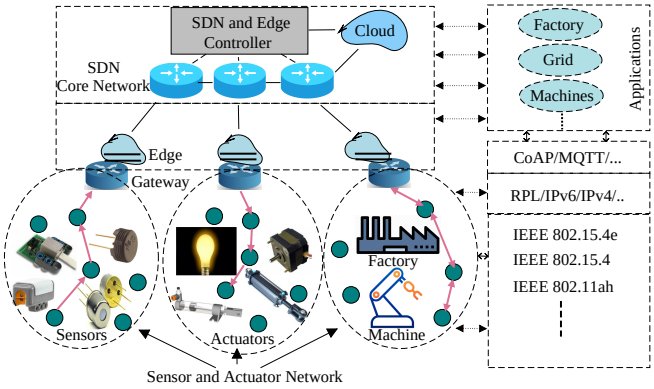


Fig. 2: Pro-Edge Network Architecture and Protocol Stack

in an Industrial IoT scenario. The authors in [27] developed a model for adaptive configuration to host various services at the fog node to provide service demands of sensor nodes at different times. Using edge computing to provide IoT security, Hsu *et al.* [28] introduced a reconfigurable security model that uses a neighbouring edge device that has strong computation capability. Aiming for Edge computing utilization by the reconfiguring computing system, a micro-architecture is implemented by Soliman *et al.* [29].

TABLE I: Summary of existing works

Paper (Ref.)	Priority	Reconfigure WSAN	Reconfigure App Module	Core Network	Industry
Soliman *et. al* [29]	✗	✓	✗	✓	✗
Rahman *et. al* [26]	✗	✓	✓	✗	✗
Chen *et.al* [27]	✗	✓	✗	✗	✓
Hsu *et. al* [28]	✓	✓	✗	✗	✗
Pro-Edge	✓	✓	✓	✓	✓

The existing literature on Industrial IoT addresses the issues of scheduling for low latency and reliable communications. However, they do not consider the strict time requirements of sensing and actuating traffic. Also, the proposed solutions lack centralized control over multiple wireless and backend communication for monitoring such traffic flows throughout the network. These solutions lack the programmability of computation and communication for rendering flexibility in different edge computing operations. A summary of the existing works, as compared to Pro-Edge, is presented in Table I.

III. Pro-Edge: A Programmable Edge Network for Industrial IoT

Herein, we propose a programmable edge network for reducing the actuation latency and improving the reliability in terms of packet delivery and availability of actuation policy. Automation related decisions are processed by a set of configurable application modules and the related network configurations (e.g., WSAN and SDN Core) are programmed dynamically by the SDN and edge controller; we call it a Pro-Edge Controller. In case if sensor and actuator are from different WSAN networks, traffic flows are forwarded to the destined edge nodes using the OpenFlow protocol.

A. System Model

Fig. 2 gives an overview of the proposed network architecture. The edge nodes are equipped with a programmable server close to the edge device. The proposed system considers a set of sensors/actuator nodes $S = \{S_1, S_2, .. \cdots S_n\}$, where n is the maximum number of devices deployed in an industry field. There can be more than one sensor or actuator in a single device, and a single device may contain a sensor and actuator together. A set of gateway devices $G = \{G_1, G_2, \cdots, G_m\}$, where m is the maximum number of gateway devices, are responsible for providing network infrastructure to the sensor and actuator node and interfacing to the SDN core network. The SDN core network has a set of forwarding nodes $C = \{C_1, C_2, \cdots, C_l\}$, where l is the maximum number of core network devices. The gateway and core devices are controlled by an SDN controller named $Pro - Edge\ controller$. The $Pro - Edge\ controller$ programs the devices for dynamic data forwarding and actuation policy implementation. We keep a set of actuation policies $A = \{A_1, A_2, \cdots A_y\}$, where y is the maximum number of policies, in an edge module. The edge modules are installed over the gateway devices. Also, the controller can also program the configurable parameters such as sleep cycle, topology, etc., for the WSAN.

B. Edge Layer for Sensor and Actuator Network

An edge node handles the reconfiguration of an existing task or configuring a new task, related to WSAN. This, also helps for effectively routing data to a destination. Other network and device configurations such as channels, device sleep cycles, power control, etc., are considered for dynamic programming by the controller. The proposed layer support WSANs such as 6LoWPAN under the control of the 6LBR and TCP/IP under WiFi-AP.

C. Edge Layer for Industrial Applications

The SDN controller functionality is designed to manage edge nodes that are responsible for executing a set of actuation policies, using application programming interfaces (APIs). The controller can configure different parameters related to a set of actuation policies based on a request from the edge node. The proposed programmability allows creating different slices of functions over the same physical network. For example, a set of threshold values, $a_{\varepsilon_i} = \{a_1, a_2, \cdots, a_t\}$, where t is the maximum threshold, are maintained for an edge node i. This enables the edge nodes to execute certain actuation decisions based on threshold values. The controller can dynamically alter these criteria in the same network.

Provisioning the limited resources at the edge, every end-user task cannot be processed by the connected edge devices. In such a scenario, the current status of each edge device is used to determine whether to accept or reject task. To compute the status, we consider some of the main parameters such as next availability of service time for delay-sensitive tasks, CPU utilization of the edge device, remaining energy, availability of RAM and storage. The status of each edge node is calculated as:

$$F_i = \{F_i^c, F_i^e, F_i^m, F_i^s\}, \quad (1)$$

where F_i^c, F_i^e, F_i^m and F_i^s are i^{th} edge device's status of time of availability and computation capacity, energy, memory (RAM) and storage, respectively. We calculate these parameters to obtain the status of an edge device as

1) *Time and CPU utilization status:* To find next availability of service time, it needs to take two factors into consideration, i.e., total computation time of a task and time required by tasks waiting at the queue, which can be calculated as:

$$T_i^{avl} = T_{i,k}^c + \sum_{Q_K}^{k \in K} T_{Q_K}, \tag{2}$$

where $T_{i,k}^c$ and T_{Q_K} are computation time required at edge node i for a task k and the sum of time required to compute all the tasks holding by the queue, respectively. As a result the total computation time is calculated as

$$T_{i,k}^c = \frac{V_{data}^k}{\sum_{p=1^k}^{p_{max}^k} r_{k,p}^i}, \tag{3}$$

where p_{max}^k denotes the maximum number of periods that the task k can hold. Whereas, $r_{k,p}^i$ denotes allocated computation resource at the i^{th} edge node for the task, k during the time period, p. And V_{data}^k is the input data for the task, k.

$$p_{max}^k = \left\lfloor \frac{D_k}{X_i} \right\rfloor, \tag{4}$$

D_k and X_i represent the delay for task k and the constant time of each period p, respectively. Note that $T_{i,k}^c \propto V_{data}^k$ and $T_{i,k}^c \propto \frac{1}{r_{k,p}^i}$, it derives that if volume gets increased, the computation time also gets increased. But with a cost of increased allocated resources, the computation time decreases. We calculate R_{need}^k in Eq. 5 that finds the required resource for task k in million instructions per second (MIPS). For processing 1-bit data of image, i.e., the processed data R_{need}^k MIPS needed.

$$R_{need}^k = I_k^i . V_{data}^k, \tag{5}$$

After obtaining the required resource for a task, k a condition arises that we should allocate a little more resources for the task. This condition can be seen in Eq. 6 as below:

$$R_{need}^k \leq \sum_{p^k}^{p=1} r_{k,p}^i, \tag{6}$$

Moreover, the total resources allocated for ith node is greater than the resource needed by all the tasks at i for the time period p is:

$$\sum_{k \in K_p^j} r_{k,p}^i \leq R^i, \tag{7}$$

Finally, we calculate the remaining computation time at i^{th} node as:

$$\sum_{T, p_{max}^t} R^i(k) = \sum_{T, p_{max}^t} \left(R_{need}^k - \sum_{p=1}^{p'k} r_{k,p}^i \right) = F_i^c, \tag{8}$$

where, $\sum_{T p_{max}^t} R^i(k)$ defines the remaining computation resource for all the task need to be completed before the time period p_{max}^t. Here $\sum_{p=1}^{p'k} r_{k,p}^i$ is the resource allocated, $p'k$ is the number of period that the task k has run and R_{need}^k is the resource needed as calculated in Eq. 5.

2) *Energy status:* We calculate the amount of energy consumed at different states for a task as:

$$E_{total}^i = E_{tx}^i + E_{rx}^j + E_c^i + E_{id}^i + E_{sl}^i, \tag{9}$$

where $E_{tx}^i, E_{rx}^j, E_c^i, E_{id}^i$ and E_{sl}^i represent the amount of energy consumption at transmission, reception, computation, idle and sleeping state, respectively. Energy consumed at each computation state can be calculated as

$$E_\partial^i = \omega_\partial^i \rho_\partial^i, \tag{10}$$

where, ω_∂^i and ρ_∂^i are the power need, and the total time needed by the edge node i in state ∂. To calculate the remaining energy at an edge node i, we assume that the edge node has $E_{initial}$ Joule amount of energy at the beginning. We find the remaining energy as

$$E_{rem}^i(p) = E_{initial} - E_\partial^i(p) \quad = F_i^e, \tag{11}$$

here, $E_\partial^i(p)$ is the total amount of consumed energy that is calculated as

$$E_\partial^i(p) = \sum_{P=0}^{p} E_{total}^i(P), \tag{12}$$

here $p \in P$ is a specific time quantum.

3) *Memory status:* We calculate the amount of memory needed by a task k as

$$M_{need}^i = M_{need}^{i,k_1} + \sum_{k_2}^{k_m} M_{Queue}^i(K), \tag{13}$$

here, M_{need}^{i,k_1} is the memory amount needed for storing instructions of the code/program in the RAM by the task k_1 which is currently available in i^{th} edge node for execution and $K = \{k_1, k_2, k_3, ..., k_m\}$, where m is the maximum number of tasks. Also, k_1 is the task currently in execution state and rest are the tasks are in the task queue of i^{th} edge node. So, $\sum_{k_2}^{k_m} M_{Queue}^i(K)$ represents the memory needed for queuing tasks in the RAM. Now, we calculate the remaining memory as

$$M_{Rem}^i = M_{initial}^i - M_{need}^i = F_i^m, \tag{14}$$

where $M_{initial}^i$ is the edge nodes dedicated memory.

4) *Storage Status:* Similarly, for memory status calculation, we first calculate the storage needed by a task k at i^{th} edge node as

$$S^{i,p}_{need} = S^{i,p}_{k} + S^{i,p}_{data} + S^{i,p}_{result} = F^{s}_{i}, \quad (15)$$

here, $S^{i,p}_{k}$ is the required storage size for task k to complete in time period p, $S^{i,p}_{data}$ is the required storage size to store the sensed data which need to be processed for time period p and $S^{i,p}_{result}$ is the required storage size for storing results of processed data for time period p. Now, in Eq. 16, we find the remaining storage status of the edge node i as

$$S^{i}_{Rem} = S^{i}_{initial} - S^{i,p}_{need}, \quad (16)$$

here $S^{i}_{initial}$ is the edge nodes dedicated storage capacity and $S^{i,p}_{need}$ is the total amount of storage utilizing by i^{th} edge node.

Algorithm 1 presents the proposed programming operation for the module. The controller maintains a list of modules and their priority. We assume that the required library for all the tasks is already available in the targeted edge node. Whenever there is a requirement of programming (i.e., installing or configuring) a particular module, the controller first checks the priority. Accordingly, the most important task to the controller is programmed initially. However, to provide fairness for failed programmability, the associated task is uplifted with an increment in its priority value.

Algorithm 1: Application Module Programming

Inputs: A set of programmable parameters, $\mathcal{P} = \{P_1, P_2, P_3, \cdots, P_z\}$
Output: Priority and fairness-based programmability

```
for Request of programming do
    Check parameter(s) and to which module it/they belong(s) to
    Check priority of the module's policy
    Send a message to edge node requesting programming
    if Positive acknowledgement received then
        Create API with selected P
        Start programming module with highest priority p_m
    else if Negative acknowledgement received then
        p_m = p_m + 1
    else
        Try sending a message again after time τ
```

D. Priority-aware Flow-placement

We redefined a flow in the network considering the resources and introduced a new field in the flow table and header to identify the proposed flow type. The packet_IN header is modified to add resource version and name within the data field. Once the resource name is known to the controller, the associated flow-rules are dynamically placed at the requested device. The proposed flow placement scheme prioritizes the traffic dealing with the programmability of resources, as depicted in Algorithm 2. The resource manager module in the controller keeps a record of all essential available resources for making future programming decisions.

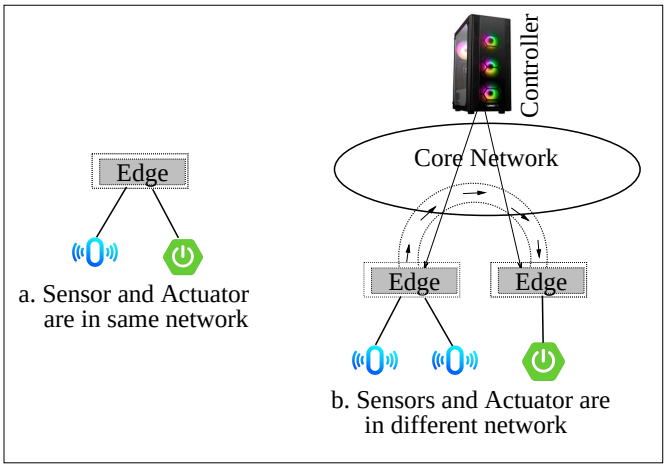


Fig. 3: Sensor and actuator placement in WSAN

Algorithm 2: Automation-intensive priority flow placement

Inputs: Flows with priority
Output: Flow placement and forwarding

```
Classify flows with decreasing priority as per the resources
if Automation-related packet received then
    Check priority value in the table
    if Multiple priority flows detected then
        Process flow with highest priority value
    else
        Process flow as first come, first serve basis
else
    Continue default actions
```

E. Sensor-Actuator Delay Model: Actuation Latency

We present an analysis of communication delay between a sensor and its associated actuator device. The various factors that contribute to this delay are contention at the sensor, propagation delay, processing delay, and the time taken at the actuator. Moreover, due to higher priority traffic (for some of the critical applications), a significant delay can be observed at a lower priority traffic caused by the channel contention and admission control.

A Raspberry-Pi-4 device with a quad-core processor is used to analyze multi-processor CPU Scheduling and to measure transmission latency. The network is setup such that a transmission is facilitated on a channel at a specific time slot that is chosen after contention. For independent flows in a multi-processor system, the time units in the CPU for processing a

flow and the particular time slot where a packet transmission occurs are considered equivalent. We map a processor to each deployed channel as many-to-many cardinality. Tasks are executed on a multiprocessor; each flow is mapped to a task. Based on the quad-core processor of the device utilised in our test-bed, we make the following analysis for response time: only one transmission is allowed in each channel at each time slot, provisioning that WSAN does not allow concurrent transmission.

For m number of channels are scheduled for flow $FL = \{FL_1, FL_2, FL_3, ..., FL_n\}$. Each flow is calculated as:

$$FL_x = (D_x, P_x, S_x, R_x, C_x), \tag{17}$$

where D_x, P_x, S_x, R_x and C_x are deadline, period, start time, route and transmission count, respectively. All the links between the senor to actuator at S_x for communication medium is defined by the route R_x. We assume at most x (re)transmissions are scheduled for one link in order to improve reliability and we obtain C_x=$|R_x| \times$. P_x, i refers to i^{th} packet of FL_x. We calculate end-to-end delay for the packet $P_{x,i}$ as:

$$\mathcal{D}_{x,i} = C_d^f + C_{x,i}^t + C_x, \tag{18}$$

here C_d^f and $C_{x,i}^t$ denotes conflict delay and contention delay, respectively.

Further, to evaluate the latency in complete task execution, the following two cases are considered:

1) Both sensor and actuator are in the same WSAN.
2) Sensor and actuator are in the different WSAN.

These instances are elaborated in Fig. 3. Total delay in both the scenarios is depicted by equation 19 and equation 20, respectively.

$$\delta_1 = \delta_{sn}^{cont} + \delta_{prop}^{s-e} + \delta_{edge}^{proc} + \delta_{prop}^{e-a} + \delta_{act} \tag{19}$$

$$\delta_2 = \delta_{sn}^{cont} + \delta_{prop}^{s-e} + \delta_{edge}^{proc} + \delta_{core} + \delta_{prop}^{e-a} + \delta_{act} \tag{20}$$

Here, $\delta_{sn}^{cont}, \delta_{prop}^{s-e}, \delta_{edge}^{proc}, \delta_{prop}^{e-a}$ and δ_{act} represent different delays i.e., delay representing contention at sensor device, delay in propagation from sensor to edge device, processing delay at the edge, propagation delay from edge to actuator device and finally delay at the actuator device, respectively.

IV. Performance Evaluation

The performance of the proposed scheme is analyzed through simulations and on a test-bed. The details of the parameters related to these experiments are mentioned in Table II. We consider existing edge and cloud-based actuation control solutions as benchmarks for comparing with Pro-Edge.

TABLE II: System and other parameters used in the performance analysis

Testbed Parameter	**Value**
No. of Edge nodes	6
No. of Controller	1 (POX) [30]
APIs	Python/C-Socket API, Openflow [31]
Switch Platform	OVS [32]
Edge and Gateway Device	RaspberryPi
End device	ESP8266
Sensors and Actuator	Motion, Ultrasonic, and LED
Simulation Parameter	**Value**
Simulator tool	WorkflowSim [33] and iFogSim [34]
# of clients (Max)	160
# of edge nodes (Max)	60
# of Tasks	20
Cloud RAM	49150 Mb
Edge node RAM	4096 Mb
Terminal Node RAM	1024 Mb
Cloud CPU Cycle	5000 mips
Edge CPU Cycle	3000 mips
Client node CPU Cycle	700 mips
Cloud bandwidth	(100(up), 10000(down))Mbps
Edge node bandwidth	(10000(up), 10000(down))Mbps
Client node bandwidth	(10000(up), 10000(down))Mbps
Programmable Parameter	**Value**
Example Modules	Sensing and Actuating
Decision Parameters	Policies, Threshold values, Variables
SDN Core Parameters	Flow - Add, Modify, Modify-Strict, Delete, Delete Strict
WSAN Parameters	Channels, Slots, Superframe, Paths

A. Experimental Setup

An experimental test-bed is setup as shown in Fig. 4. We consider an industrial automation scenario, where motion and ultrasonic sensors are used to monitor the movement and distance of automated robots. We consider three different types of tasks: (i) to check the movements of a robot (RoboMovement), (ii) to check how far the machine hands are (RoboDistance), (iii) to generate an alert based on a threshold value (RoboAlert). All these tasks are run by the nearest Pro-Edge node (one of the Raspberry Pi devices). Multiple requests for such tasks are generated from multiple clients. At first, the ultrasonic and motion sensor collect the data that is programmed through the mote (ESP8266); the mote is connected to an edge node with an access point using the constrained application protocol (CoAP). The edge node compares the data with a pre-defined threshold and makes an actuating decision. Concurrently, it also checks if the actuator is connected to it. If it is, then it immediately implements the actuation decision. Otherwise, it requests the Pro-Edge controller to forward the data to the targeted edge node. The Pro-Edge controller uses the core network (nodes) to forward the actuation related data to the destination edge node. Lastly, if the actuation policy is to be modified/changed, the Pro-edge controller uses an edge reconfiguration interface to program the task.

B. Performance Metrics

The following metrics are used to evaluate the performance of the proposed programmable network.

1) Actuation Time: It is the time difference between sensing and actuation. The edge node uses the sensed data for decision making and actuation.
2) Network Usage: It is the total network bandwidth used for data and task-related communication. The proposed scheme forwards data and control packets throughout network.

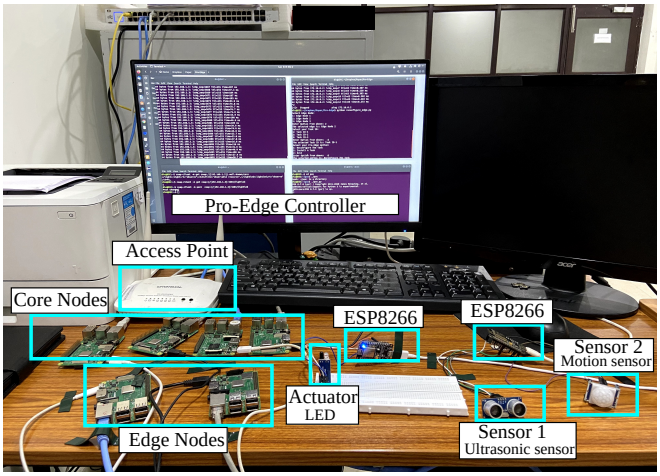


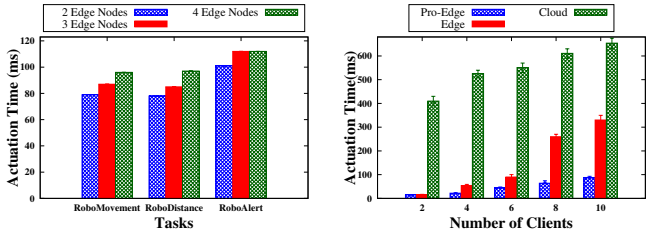
Fig. 4: A prototype of Pro-Edge with the components – (i) Sensors, (ii) Actuators, (iii) Mote, (iv) Core nodes, (v) Access point (vi) Edge nodes, and (vii) Pro-Edge controller

3) Energy Consumption: This is the average energy consumed by the edge nodes and clouds. Due to task processing and transmission/reception of data, each device consumes energy.
4) Execution Time: Time taken to execute a task at the computing devices. As most of the devices are resource constrained, it is a crucial parameter.
5) Cost: It is the amount of network and computing resources spent for execution of all the given tasks.

C. Performance Results

1) Test-bed Evaluation: For implementation, we use Raspberry Pi devices connected by ESP8266 for connectivity with WSANs. In additions these Raspberry Pi devices with computing capability can be regarded as edge nodes. The edge consists of two major components – WiFi-AP and OpenFlow Virtual Switch (OVS). Using Ethernet and WiFi interface attached with gateway devices, the traffic flows are sent from localized network Openflow-based forwarding. It deserves to mention that the WiFi-AP successfully assigns IP addresses to both interfering networks, i.e., WSAN and SDN core network. In our proposed configurable software module for the edge nodes with APIs handles the actuation decisions and WSAN network scheduling for control loop traffic.

In our initial experiments, we aim to quantify the difference in latency if decisions are made through Pro-Edge versus the traditional cloud and edge architecture for an increasing number of clients. Later, we measure the service response time in case of task availability through programming in comparison to traditional task offloading schemes. Data forwarding using SDN for real-time task processing is compared with non-SDN based solutions. For individual tasks, we measure the overall actuation time as shown in Fig. 5a. The controller places flow rules for forwarding among the edge-cum-router nodes. With an increasing number of edge nodes, the delay increases as it may require to forward the data/alert among multiple nodes.



(a) (b)

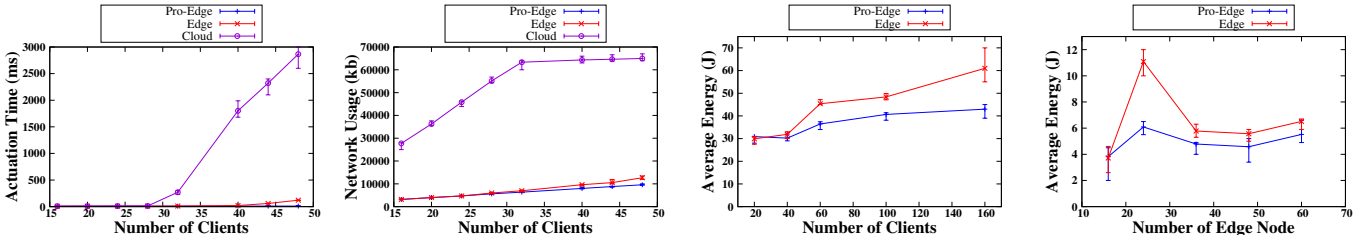
Fig. 5: Actuation time in the testbed prototype

Fig. 5b shows the test-bed results in terms of average actuation time with an increasing number of clients (i.e., number of requests). The proposed scheme significantly reduces the actuation time with the use of dynamic task reconfiguration and data forwarding. With more number of requests for tasks, it may require to offload the same to other edge through core nodes, but with a cost of increased latency in traditional approach. Pro-Edge with SDN controller reduces actuation delay up to 200% and 400% as compared to Edge and Cloud-based decisions, respectively. Dynamic configuration to the actuation policies, dynamic data forwarding, and new module installation in the proposed scheme, shows improved results in terms of actuation latency.

2) Simulation analysis of the proposed scheme: We consider 20 core nodes with sensor and actuator devices in different regions of the network. Fig. 6a shows the average actuation time (sensing to actuation) in the network. It can be observed that the delay increases significantly as the number of clients increase. Latency in cloud-based actuation is high when the number of clients cross 30. Although decisions made through an edge-device significantly reduces the delay, Pro-Edge further lowers it by re-configuring the actuation policy and flow rules at run-time. Thus, waiting time is reduced using the proposed scheme. Similarly, programmability on the decision parameters of edge-devices, reduces the number of edge nodes and the overall network usage (can be seen in Fig. 6b) for executing a task. In contrast, cloud and edge based solutions find other computing nodes in case the current edge is overloaded.

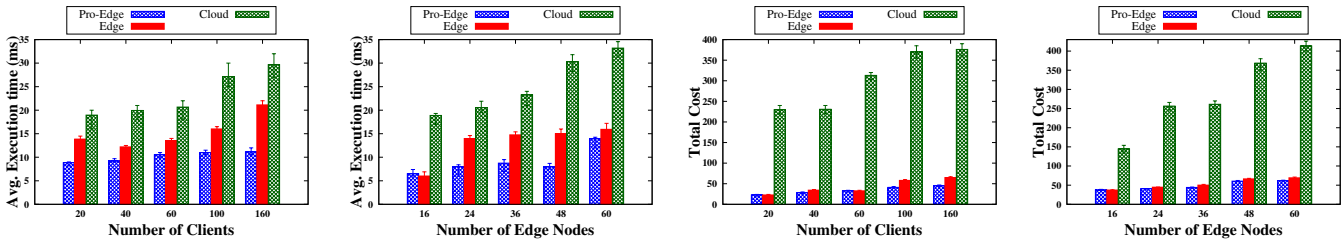
We also measure the average energy consumption in the network to execute a task as requested by the clients. As shown in Fig. 6c, with an increasing number of clients, the energy consumption is higher. The key energy consumption elements are processing and communication. Pro-Edge lowers the processing requirement and overall communication by utilizing a fewer number of edge nodes in the network. Also, with more number of edge nodes, a slightly better energy consumption can be seen using Pro-Edge (refer Fig. 6d).

Further, to analyze the effect on average execution time, we experimented with four edge nodes for an increasing number of clients as shown in Fig. 6a. We observed a significant improvement in terms of execution time when the client count is higher. Fig 6b shows the performance of the proposed scheme with 100 clients and number of edge nodes up to 60. In a network with resource-constraint IoT devices and backbone routers, executing a task and forwarding the data



(a) Actuation Time (b) Network Usage (c) Energy Consumption (d) Energy Consumption

Fig. 6: (a) Average actuation time, (b) Overall network usage, (c) Average energy consumed with 4 edge nodes, (d) Average energy consumed with 100 clients



(a) Execution Time (b) Execution Time (c) Cost (d) Cost

Fig. 7: (a) Execution time with 4 edge nodes, (b) Execution time with 100 clients, (c) Total cost with 4 edge nodes, (d) Total cost with 100 clients

to the respective edge and gateway nodes for actuation is challenging. Our solution executes multiple tasks with a lesser number of edge nodes, which reduces the number of decisions in terms of offloading and data forwarding. Also, the SDN controller placed the required flow-rules for forwarding with low latency to the respective destination node.

The proposed scheme reduces overall cost in using the resources such as number of network components, CPU, and storage (refer Fig. 6c). With the use of programmable edge and dynamic flow-rule placement, *Pro-Edge* utilizes the available resource in the network in an efficient manner. As shown in Fig. 6d, with the increase in number of edges, total cost increases significantly in case of cloud processing. Utilizing the edge devices for computation, this cost is reduced in edge computing approaches. Moreover, our scheme utilizes an optimal number of edges for the computing purposes. Therefore, *Pro-Edge* shows the least cost as compared to the traditional edge and cloud-based approaches.

V. Conclusion

This paper presented *Pro-Edge*, an edge computing architecture for future IIoT applications. The proposed solution reduces actuation and offloading time, increase task availability, and provides dynamic actuation policy creation. Our approach reduces latency and increases response time significantly compared to edge and cloud-based architecture. Although Pro-Edge is experimented for an industrial automation scenario, it is suitable for all applications that involve sensor and actuator operations. As part of future work, mechanisms to handle the failure of edge devices is planned.

References

[1] G. Berardinelli, P. Mogensen, and R. O. Adeogun, "6G subnetworks for Life-Critical Communication," in *6G Wireless Summit (6G SUMMIT)*, pp. 1–5, IEEE, 2020.

[2] V. Balasubramanian, M. Aloqaily, and M. Reisslein, "An sdn architecture for time sensitive industrial iot," *Computer Networks*, vol. 186, p. 107739, 2021.

[3] I. Al Ridhawi, S. Otoum, M. Aloqaily, Y. Jararweh, and T. Baker, "Providing secure and reliable communication for next generation networks in smart cities," *Sustainable Cities and Society*, vol. 56, p. 102080, 2020.

[4] J. Al-Jaroodi, N. Mohamed, and I. Jawhar, "A service-oriented middleware framework for manufacturing industry 4.0," *ACM SIGBED Review*, vol. 15, no. 5, pp. 29–36, 2018.

[5] W. Z. Khan, M. Rehman, H. M. Zangoti, M. K. Afzal, N. Armi, and K. Salah, "Industrial internet of things: Recent advances, enabling technologies and open challenges," *Computers & Electrical Engineering*, vol. 81, p. 106522, 2020.

[6] A. Ksentini and N. Nikaein, "Toward enforcing network slicing on RAN: Flexibility and resources abstraction," *IEEE Communications Magazine*, vol. 55, no. 6, pp. 102–108, 2017.

[7] H. Liao, Z. Zhou, X. Zhao, L. Zhang, S. Mumtaz, A. Jolfaei, S. H. Ahmed, and A. K. Bashir, "Learning-based context-aware resource allocation for edge-computing-empowered industrial iot," *IEEE Internet of Things Journal*, vol. 7, no. 5, pp. 4260–4277, 2019.

[8] E. Oyekanlu, "Predictive edge computing for time series of industrial IoT and large scale critical infrastructure based on open-source software analytic of big data," in *2017 IEEE International Conference on Big Data (Big Data)*, pp. 1663–1669, IEEE, 2017.

[9] T. Kumar, E. Harjula, M. Ejaz, A. Manzoor, P. Porambage, I. Ahmad, M. Liyanage, A. Braeken, and M. Ylianttila, "BlockEdge: blockchain-edge framework for industrial IoT networks," *IEEE Access*, vol. 8, pp. 154166–154185, 2020.

[10] M. Lavassani, S. Forsström, U. Jennehag, and T. Zhang, "Combining fog computing with sensor mote machine learning for industrial iot," *Sensors*, vol. 18, no. 5, p. 1532, 2018.

[11] W. Zhang, Z. Zhang, S. Zeadally, H.-C. Chao, and V. C. Leung, "Masm: A multiple-algorithm service model for energy-delay optimization in edge artificial intelligence," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 7, pp. 4216–4224, 2019.

[12] Y. Hao, Y. Miao, L. Hu, M. S. Hossain, G. Muhammad, and S. U. Amin, "Smart-edge-cocaco: Ai-enabled smart edge with joint computation,

caching, and communication in heterogeneous iot," *IEEE Network*, vol. 33, no. 2, pp. 58–64, 2019.

[13] H. Li, K. Ota, and M. Dong, "Learning iot in edge: Deep learning for the internet of things with edge computing," *IEEE network*, vol. 32, no. 1, pp. 96–101, 2018.

[14] A. Mahmud, R. Rahmani, and T. Kanter, "Deployment of flow-sensors in internet of things' virtualization via openflow," in *2012 Third FTRA International Conference on Mobile, Ubiquitous, and Intelligent Computing*, pp. 195–200, IEEE, 2012.

[15] S. Costanzo, L. Galluccio, G. Morabito, and S. Palazzo, "Software Defined Wireless Networks: Unbridling SDNs," in *2012 European Workshop on Software Defined Networking*, pp. 1–6, 2012.

[16] T. Luo, H.-P. Tan, and T. Q. Quek, "Sensor OpenFlow: Enabling software-defined wireless sensor networks," *IEEE Communications letters*, vol. 16, no. 11, pp. 1896–1899, 2012.

[17] R. K. Das, N. Ahmed, F. H. Pohrmen, A. K. Maji, and G. Saha, "6LE-SDN: An Edge-Based Software-Defined Network for Internet of Things," *IEEE Internet of Things Journal*, vol. 7, no. 8, pp. 7725–7733, 2020.

[18] S. Bera, S. Misra, S. K. Roy, and M. S. Obaidat, "Soft-WSN: Software-defined WSN management system for IoT applications," *IEEE Systems Journal*, vol. 12, no. 3, pp. 2074–2081, 2016.

[19] A. De Gante, M. Aslan, and A. Matrawy, "Smart wireless sensor network management based on software-defined networking," in *2014 27th Biennial Symposium on Communications (QBSC)*, pp. 71–75, IEEE, 2014.

[20] B. R. Al-Kaseem, Y. Al-Dunainawi, and H. S. Al-Raweshidy, "End-to-end delay enhancement in 6LoWPAN testbed using programmable network concepts," *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 3070–3086, 2018.

[21] M. Baddeley, R. Nejabati, G. Oikonomou, M. Sooriyabandara, and D. Simeonidou, "Evolving SDN for low-power IoT networks," in *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*, pp. 71–79, IEEE, 2018.

[22] M. L. Miguel, E. Jamhour, M. E. Pellenz, and M. C. Penna, "SDN architecture for 6LoWPAN wireless sensor networks," *Sensors*, vol. 18, no. 11, p. 3738, 2018.

[23] B. T. De Oliveira, L. B. Gabriel, and C. B. Margi, "TinySDN: Enabling multiple controllers for software-defined wireless sensor networks," *IEEE Latin America Transactions*, vol. 13, no. 11, pp. 3690–3696, 2015.

[24] B. T. de Oliveira, R. C. A. Alves, and C. B. Margi, "Software-defined wireless sensor networks and internet of things standardization synergism," in *2015 IEEE Conference on Standards for Communications and Networking (CSCN)*, pp. 60–65, IEEE, 2015.

[25] J. Kim, F. Filali, and Y.-B. Ko, "A lightweight CoAP-based software defined networking for resource constrained AMI devices," in *2015 IEEE International Conference on Smart Grid Communications (SmartGridComm)*, pp. 719–724, IEEE, 2015.

[26] T. Rahman, X. Yao, G. Tao, H. Ning, and Z. Zhou, "Efficient edge nodes reconfiguration and selection for the Internet of Things," *IEEE Sensors Journal*, vol. 19, no. 12, pp. 4672–4679, 2019.

[27] L. Chen, P. Zhou, L. Gao, and J. Xu, "Adaptive fog configuration for the industrial internet of things," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 10, pp. 4656–4664, 2018.

[28] R.-H. Hsu, J. Lee, T. Q. Quek, and J.-C. Chen, "Reconfigurable security: Edge-computing-based framework for iot," *IEEE Network*, vol. 32, no. 5, pp. 92–99, 2018.

[29] W. G. Soliman, B. K. Priya, D. A. Reddy, P. Anusha, and D. R. K. Reddy, "Reconfigurable microarchitecture-based pmdc prototype development for iot edge computing utilization," *IEEE Sensors Journal*, vol. 21, no. 2, pp. 2334–2345, 2020.

[30] S. Kaur, J. Singh, and N. S. Ghumman, "Network programmability using POX controller," in *ICCCS International Conference on Communication, Computing & Systems, IEEE*, vol. 138, p. 70, sn, 2014.

[31] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: enabling innovation in campus networks," *ACM SIGCOMM computer communication review*, vol. 38, no. 2, pp. 69–74, 2008.

[32] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, *et al.*, "The Design and Implementation of Open VSwitch," in *12th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 15)*, pp. 117–130, 2015.

[33] W. Chen and E. Deelman, "WorkflowSim: A toolkit for simulating scientific workflows in distributed environments," in *2012 IEEE 8th International Conference on E-Science*, pp. 1–8, 2012.

[34] R. Mahmud and R. Buyya, "Modelling and simulation of fog and edge computing environments using iFogSim toolkit," *Fog and edge computing: Principles and paradigms*, pp. 1–35, 2019.