A Distributed SDN Controller for Distributed IoT

RAMIN FIROUZI AND RAHIM RAHMANI, (Senior Member, IEEE)
Department of Computer and Systems Sciences (DSV), Stockholm University, 164 07 Kista, Sweden
Corresponding author: Ramin Firouzi (ramin@dsv.su.se)

ABSTRACT Over the past decade, the Internet of Things (IoT) has undergone a paradigm shift away from centralized cloud computing towards edge computing. Hundreds of billions of things are estimated to be deployed in the rapidly advancing IoT paradigm, resulting in enormous amounts of data. Transmitting all these data to the cloud has recently proven to be a performance bottleneck, as it causes many network issues relating to latency, power consumption, security, privacy, etc. However, existing paradigms do not use edge devices for decision making. The use of distributed intelligence could strengthen the IoT in several ways by distributing decision-making tasks to edge devices within the network, rather than sending all data to a central server. In this approach, all computational tasks and data are shared among edge devices. To achieve efficient distribution of IoT intelligence and utilization of network resources, it is necessary to integrate the transport network control with distributed edge and cloud resources to provide dynamic and efficient IoT services. This challenge can be overcome by equipping an edge IoT gateway with the intelligence required to extract information from raw data and to decide whether to actuate itself or to outsource complex tasks to the cloud. Distributed intelligence refers to a collaboration between the cloud and the edge. In this context, we first introduce a distributed SDN-based architecture for IoT that enables IoT gateways to perform IoT processing dynamically at the edge of the network, based on the current state of network resources. Next, we propose an algorithm for selecting clients for the training process of Federated Learning Applications, based on the context information of the network. In the last step, we deploy Federated Learning Applications in our distributed SDN-based architecture, using the gateways to provide distributed intelligence at the edge of the network, and conduct a comprehensive and detailed evaluation of the system from several perspectives.

INDEX TERMS Distributed intelligence, SDN, federated learning, edge computing, Internet of Things.

I. INTRODUCTION
As technology continues to evolve (e.g., radio frequency identification (RFID), wireless networks, Bluetooth low energy (BLE), etc.), the Internet will consist of an ever-increasing number of devices in the future. It is estimated that by 2025, 55.7 billion devices will be connected worldwide, 75% of which will be connected to an IoT platform [1]. Not only will these produce vast amounts of data (73.1 ZB by 2025), but these data will have particular characteristics such as higher velocity, more modes, higher data quality, and heterogeneity. Sending and processing these quantities of data in the cloud can cause excessive network congestion, which creates a bottleneck for IoT applications such as manufacturing, transportation systems, and healthcare, which apply strict real-time criteria. These types of applications typically require on-the-fly processing of continuous data. Researchers have recently attempted to solve these problems by bringing computation and data storage closer to where they are needed, and this has led to the emergence of a new computation paradigm called edge computing. Edge computing offers energy efficiency, security, mobility, heterogeneity, and, most importantly, improved response times and reduced bandwidth by processing data close to the end devices [2].

To implement the concept of edge computing in IoT, many works have proposed modifying the functionality of IoT gateways. Rather than collecting data and sending it to the cloud, IoT gateways process the data and make decisions at the edge level. Carrying out processing and decision making by IoT gateways rather than in the cloud can mitigate latency problems for real-time applications. Devices with adequate computing capacity, such as single-board computers, can act as IoT gateways at the edge layer to process the raw data and make decisions [3]. It is important to emphasize that the use of edge computing to increase the computing power of edge devices is not a substitute for the cloud; instead, these are complementary paradigms that should be used together [4]. For instance, the cloud will make a decision when a gateway...
cannot process the data and decide due to its limited computing capacity.

Processing the data and making a decision at the edge layer, as in IoT gateways, requires artificial intelligence (AI), which can be distributed thanks to the nature of edge computing and IoT. In view of this, a distributed machine learning mechanism called Federated Learning Applications has been developed by Google [5]. This is a learning technique that trains a model with the help of a heterogeneous set of edge devices or siloed data centers without the need to transfer data among devices. Federated Learning exploits the computing resources of various devices while maintaining privacy [6]. Edge computing is an ideal setting in which to implement the Federated Learning framework, due to the similarities between edge computing and Federated Learning. As a result, edge Federated Learning has attracted attention from researchers in academia and industry over recent years. Federal learning is a three-step procedure, which is repeated until the expected accuracy is achieved. In the first step, a central server passes an initialized global model to the clients. In the second step, the clients train the initialized global model individually with their local data. In the third step, the clients send the trained local model back to the central server rather than sending their local data.

However, it is not easy to provide extra computational resources at the edge of the network and to set up this complement while making it available everywhere [4]. In addition to the complexity of these technologies, there are several key hurdles such as edge mobility, scalability, heterogeneity, reliability, security, and privacy. Some of these issues have already been discussed in [7] and [8].

One of the most critical challenges in terms of realizing targeted pervasive scenarios is network management. Most network management tools focus on devices rather than the service or application, so there is no real way to correlate what the user is experiencing with the conditions in the network. SDN is a network management tool that can hide the complexity of this heterogeneous environment from the end-user, and particularly from developers [9]. SDN is an outstanding approach that enables the programming of network devices by separating the control plane from the data plane. It simplifies the management of the network by letting the user define the flows of the network without requiring upgrades to the devices’ firmware. Decoupling the control plane from the data plane increases control over the network, and enables more efficient use of the available resources. The advantage of this is that SDN can mitigate the complexity of edge computing. Since most existing systems extend traditional cloud data centers to edge servers, the SDN control mechanism can dynamically manage the traffic originating at the edge and provide high-quality services to users. Edge devices are simple and constrained, meaning that they cannot perform complex activities such as orchestration and service discovery. SDN can alleviate this problem with edge devices by collecting network information via a software-based controller. In other words, SDN manages a multilayer edge computing infrastructure that meets the QoS requirements of IoT applications such as performance and delays, and can improve user satisfaction [4].

Our innovations are as follows:

- We design a distributed SDN-based architecture for IoT that enables IoT gateways to perform processing dynamically at the edge of the network;
- We propose an algorithm for selecting clients for the training process of Federated Learning based on contextual information about the network;
- We deploy Federated Learning in our distributed SDN-based architecture using these gateways, and conduct a comprehensive and detailed evaluation of the system from different perspectives.

The rest of the paper is organized as follows. Section 2 introduces the main topic of this research and gives an overview of related works. In Section 3, we describe the main characteristics of the proposed model and the goals of the system in detail. In Section 4, we carry out a performance evaluation. Finally, Section 5 presents our conclusions.

II. RELATED WORKS

Many researchers have proposed SDN-based architectures for IoT over the last decade. The authors of [10] focused on the lack of a standardized communication method for east/west interfaces between distributed SDN controllers, which poses a major problem for deployment in heterogeneous cross-domain network environments. They proposed a distributed SDN control plane framework (DSF) in which a data-centric RTPS communication model was applied between control plane entities. In this framework, to overcome the problem of tight synchronization due to the holistic topology of the network, which represents the main challenge of a distributed SDN architecture, the control plane entities exchanged link discovery updates in the form of a LUMP message that enabled the routing of data packets across multiple domains.

Li et al. [11] proposed an SDN-based architecture for IoT in which the IoT gateway was SDN-enabled. The use of an open programmable interface in the proposed architecture enabled IoT gateways to be upgraded on the fly when a new application was introduced. They also defined further gateway functions such as node management, protocol conversion, and security control. In addition, their approach offered the possibility of downloading simple data processing methods or rules for the gateways. However, the proposed gateway did not have a distributed architecture, and multiple applications were prevented from using the same gateway.

Salman et al. [12] also proposed an SDN-based architecture for IoT. Their approach aimed to implement SDN in the fog layer to reap the benefits of both paradigms: the programmability capabilities of SDN, and the mobility capabilities of fog. The gateways in the proposed architecture provided key gateway services by ensuring compatibility between different communication protocols and heterogeneous networks. The proposed architecture also
included a standard northbound interface and a programmability feature that enabled dynamic gateway updates. However, the gateway did not have a distributed structure.

In [13], the authors proposed an architecture for the optimal distribution of IoT processes to the edge of the network based on network resources by integrating SDN with the IoT. This architecture enabled better control over IoT flows and allowed for the use of techniques for the avoidance of traffic congestion. The authors also proposed and experimentally evaluated the architecture of an SDN-enabled container-based edge node, which was developed to enable seamless interaction with the IoT-enabled SDN and a cloud orchestration platform.

Hakiri et al. [14] explored the challenges that needed to be overcome to design efficient IoT systems, including communication and network levels. Based on these challenges, they presented a data-centric architecture based on a symbiotic relationship between DDS and SDN with the aim of enabling agile and flexible network orchestration. In other words, the purpose of their scheme was to enable IoT systems to support SDN. A DDS northbound interface was added to the SDN controller to provide it with all the necessary functions of IoT network applications, and to enable network-agnostic support for IoT systems. Both smart objects and gateways could use the DDS middleware to Publish/Subscribe data. The IoT gateways were virtualized to allow for interconnection of the IoT devices in the proposed architecture. Gateway virtualization improved IoT multi-homing by allowing sensors to be dynamically connected to multiple networks.

Qureshi et al. [15] developed a distributed SDN approach to improve the overall energy efficiency of smart grid systems in which the controllers are logically centralized but physically distributed. These distributed controllers, each of which is responsible for a particular domain, communicate with each other to update the global state. When the data flow reached any data plane in a distributed SDN via the end devices, the proposed technique allowed the data plane to increase the response time of the controllers. Moreover, the controller determined whether the data needed a prioritized flow to be sent locally or globally, using elephant or mice flows.

Gonzalez et al. [16] proposed an open flow-based clustering management system. A domain a cluster head, a gateway, and sensor nodes. The proposed system managed communication between clusters through cluster heads, which were SDN controllers (SDNCHs). An SDNCH acted as the domain coordinator, while the gateway acted as the bridge node between the sensor nodes and the SDNCH. They also proposed a routing protocol for communication between SDNCHs in the SDN environment, and developed a testbed to evaluate the proposed protocol. However, their work did not include a performance analysis such as measuring communication overheads or message delays.

Table 1 summarizes the related works reviewed here. For each paper, we indicate which requirements were met and which were not. It is noticeable that none of these works considered data processing by the IoT gateway at the edge of the network. IoT gateways are not considered as data processors at the edge of the network in any of these papers; their role is simply to transmit data to the upper layer.

In this paper, we propose a distributed SDN system for distributed IoT intelligence, in which IoT gateways play a crucial role in terms of storing and processing distributed data.

### III. INTEGRATING SDN INTO IOT DISTRIBUTED INTELLIGENCE

#### A. DISTRIBUTED SDN

The control plane is the main unit of SDN, and consists of a set of control software that is responsible for communication between the applications and devices [17]. In other words, the control plane passes requests from the applications to the data plane and receives the information required by the applications from the data plane. In terms of their physical nature, SDN controller architectures can be divided into centralized and distributed structures [18]. The main reason for distributing the control plane over the network using controllers is that a single controller cannot handle growth in the network,
and this becomes a performance bottleneck. It also makes the network more vulnerable to attacks, as a single point of failure can disable the system. In addition to their physical classification, distributed SDN controller architectures can be conceptually divided into logically centralized and logically distributed types, based on how knowledge is diffused across the controller instances.

In a logically centralized scheme, all controllers have the same knowledge about the network. By contrast, in a logically distributed control plane, individual controllers only keep track of one piece of the overall network, known as the local domain. At least one of the participating controllers must maintain a holistic view to enable data forwarding among local domains and to communicate with northbound applications as a logically centralized control plane. This requires that controllers exchange local domain network information. There are two types of strategies for disseminating network state information across inter-domain distributed controllers: flat and hierarchical.

In a flat topology, distributed controllers act as peer nodes. Changes in the local network status are published by nodes. Nodes obtain information updates on the network state of all other peers. The global view of the network’s administration is created and maintained from the updates stored in each node. Every node in the flat model maintains a global view of the network [10].

In a hierarchical topology, distributed controllers follow a chain of command, using a tree topology structure. Child nodes update parent nodes on their local state information. The parent stores and combines the state information of the child domains, thus generating a holistic view. Each parent node is a child node of a larger administrative domain entity, and the combined view is then shared with these upper-tier entities, which are known as global controllers. A global controller forms the root of the tree, and collects state information on the local and administrative domains. The global controller generates a global view for northbound network management applications [17].

### B. FEDERATED LEARNING

The world of machine learning is constantly evolving, and new data processing techniques are emerging every day. Due to the rapid growth in data production, sending data to the server requires high bandwidth, and is almost impossible to achieve in practice. To overcome this challenge, distributed machine learning methods are widely used nowadays. As mentioned in the introductory section, one of these methods is federation learning, which was introduced in 2016 by McMahan et al. [5]. Federation learning is a distributed machine learning architecture in which a large number of clients collaborate with each other to train a model with their local data under the supervision of a server. In this architecture, no data are shared between clients and servers, and only training parameters are transmitted. One of the major challenges in Federated Learning involves synchronizing the communication between clients and servers. There are four schemes for synchronizing communication in Federated Learning: bulk synchronous parallel SGD (BSP-SGD), stale synchronous parallel SGD (SSP-SGD), asynchronous parallel (ASP), and local SGD [19]. BSP-SGD [20] is a simple approach for parallel SGD in Federated Learning in which each worker receives the entire model from the server, and they all start training the model simultaneously. The gradients are aggregated by the server before the next iteration is begun.

SSP-SGD [21] allows each worker to continue training without waiting for the other workers. To average the results, each worker needs to obtain the gradients of n−1 other workers. More computations carried out in one iteration can minimize the frequency of communication required in this approach. In contrast, the ASP framework [22] eliminates the need for synchronization: after each worker computes its gradients, it sends them to the server, which then modifies the global model without waiting for the other workers to finish. In local SGD [23], all workers must train the model for several iterations before the local models are averaged into a single global model. As shown in Table 2, these four schemes have certain advantages and disadvantages. For example, only the BSP method has high model consistency, and the model also converges easily in this approach. However, this scheme suffers from a high frequency of communication and congestion.

This paper studies the communication problems of the BSP scheme in the context of Federated Learning. An attempt is made to alleviate this problem using distributed SDN.

### IV. PROPOSED APPROACH

#### A. SYSTEM MODEL

The system model is abstracted as shown in the architecture in Figure 1. In the proposed model, distributed SDN is integrated into the two-tier architecture reported in our previous...
work [24]. This architecture consists of high-level and lower intelligence. The lower level of intelligence operates at the edge of the network, whereas the high level operates in the cloud. As shown in Figure 1, in order to implement this two-level intelligence, four planes are used: application, control, data, and M2M device layers. The last three of these layers operate at the edge of the network.

The application layer contains the typical network applications and functions, such as intrusion detection systems, load balancing, and firewalls. A traditional network uses a specialized appliance such as a firewall or load balancer, whereas SDN uses an application in which the controller manages the behavior of the data plane. The application layer also contains business applications.

The control plane forms part of the network that controls how data packets are forwarded, meaning how the data are sent from one place to another. In the proposed model, this plane is distributed, and each controller has a partial view of the network. After they have been synchronized with one another, they manage the data plane together. The control plane is described in more detail in subsection C.

The data plane is usually implemented by network devices such as switches and routers. This plane aims to forward data based on control plane instructions. In the proposed model, several new functions have been added to the functions of this plane: in addition to the typical network components, the IoT gateway has also been used to implement these functions. In our model, the data plane has five modules: the forwarding

---

**FIGURE 1. Structure of the data and control plane architecture.**
engine, the processing engine, node management, data storage and caching, and the protocol converter.

The forwarding engine is responsible for forwarding data based on control plane instructions that are implemented by network switches. The processing engine implements distributed intelligence; in this paper, federal learning is used. This module is located inside the IoT gateway. The proposed model and the IoT architecture are shown in Figure 2.

B. DESIGN OBJECTIVES OF THE PROPOSED ARCHITECTURE
The two most important aims of the proposed architecture are as follows:

1) MODULARITY
Our architecture consists of a modular design for efficient deployment of distributed intelligence using distributed SDN and Federated Learning. The fundamental advantage of this modular design relates to decoupling of the activities performed by each component. By isolating the network modules, the overall network performance can be improved as each function is implemented and improved separately from other network functions. Each module can interact with the other modules to perform its assigned task without involving the entire network, thus reducing the network cost. In addition, each module can be improved and tested individually, during or after development.

2) PERFORMANCE
The most important goal of the proposed architecture design is to improve the performance of the IoT network through implementing distributed intelligence. In Section IV, we describe the implementation and testing of our architecture. In the proposed design, each SDN controller monitors a local domain to record link updates and share them with other controllers, in order to maintain a holistic view of the network topology.

C. CONTROLLER ARCHITECTURE AND MODULES
In this section, we describe the three main modules of the distributed controller: the inter-domain, intra-domain, and interface modules, as shown in Figure 2. The intra-domain modules handle communication with other controllers, while the inter-domain modules handle communication with the local data planes. The interface is responsible for the communication between the intra- and inter-domains. The controller manages and communicates with the data plane through a Southbound Interface such as the OpenFlow protocol.

1) IOT SERVICE MANAGER
This component is responsible for adding, modifying, and removing gateway-supported services; storing and caching policies; and updating data processing and storage center rules and algorithms. In this study, Federated Learning is used to achieve distributed intelligence. This component informs the gateways about the optimization method used by the server.

2) LINK DISCOVERY AND TOPOLOGY MANAGER
The link discovery component uses the Link Layer Discovery Protocol (LLDP), and stores the information about available links and ports in the data storage component. Topology discovery involves computing the network’s topology based on information from two resources: the local domain (provided by the link discovery component), and from other domains, controlled by other controllers via the interface component. Controllers should always have an up-to-date topology of the network, since many services such as load balancing,
routing, traffic scheduling, and network robustness depend on
the network topology.

3) INTERFACE
The main task of this component is to coordinate the com-
munication between the intra- and inter-domains of the con-
troller. It consists of three subcomponents: Link Monitoring,
Topology Monitoring, and device monitoring. The Link Mon-
itoring component has two dedicated threads, one of which is
constantly listening to the link discovery component. When
there is new information about the state of the links, it is
forwarded to the dedicated topic in the Distributed MQTT
Broker. Another thread listens to the topic and notifies the
corresponding components when information is sent from
other controllers. The device and topology schemes are simi-
lar to the Link Monitoring scheme: they use two threads, one
of which listens for changes in the corresponding components
and reports these changes to other controllers via their own
topics in the Distributed MQTT Broker, and another that
applies the same procedure in reverse.

4) MQTT MESSENGER
Controllers that exchange information with each other over
the network must be very efficient. Typical request/response
protocols are not effective in this context. Some researchers
have used Publish/Subscribe protocols that have led to
improvement [14]. The MQTT protocol is a perfect choice
in this regard, as it has a Publish/Subscribe architecture. The
performance of this protocol depends heavily on its broker.
A distributed design based on MQTT has therefore been used
in this work.1

1https://vernemq.com/

D. SDN-ENABLED CONTEXT-AWARE CLIENT SELECTION
As mentioned in Section III, a parallel bulk synchronous
approach has high model consistency, and the model con-
verges easily compared to other schemes. However, its main
problem is the high number of communication repetitions
and congestion, which can be a major challenge for applica-
cations with strict requirements such as low latency. One
of the most promising solutions for addressing this problem
involves deliberately selecting clients rather than selecting
them at random. In this section, we propose an algorithm to
configure flow rules proactively for selected clients based on
the contextual information of the local nodes. The context
refers to the environment in which the devices are located,
and is also used to refer to the circumstances under which the
task is executed [25]. The proposed algorithm uses context
information provided by SDN controllers to find the shortest
path for nodes in each round of communication (as shown in
Algorithm 1). In this method, the utilization of each link is
first calculated (line 3) using Eq. (1). This may be equal to
the utilization of the corresponding port.

\[
\text{Port Utilization} = \frac{\text{Transmitted Data} (\text{bits}) \times 100}{\text{Time Interval} (s) \times \text{Port Speed}(\text{bps})} \tag{1}
\]

As stated in [26], the network throughput starts to decrease
when a link utilization rises above 70%. Thus, links with a
utilization of more than \(T=70\%\) are identified and removed
from the available link list (lines 4 and 5). After this, the
controller updates the network topology based on the new
links list (line 10).

Once the network topology has been updated, the short-
est path for each available node will be calculated using
Dijkstra’s algorithm.

In the last step, the cost of the shortest path to the server
for each node is calculated using Eq. (2), and based on these

FIGURE 3. Example of a topology used for the simulation, consisting of 16 hosts.
costs, the nodes are sorted. (if there is no shortest path for a
client, then the cost is $\infty$)

$$C_{Pi} = \sum \{LU_{jz} | link_{jz} \in P_i\} \quad (2)$$

The top K nodes, which have a lower cost than others, will be
selected for training in the current round of communication.

Algorithm 1: Node Selection Algorithm for Delay-Sensitive
Applications

```
Input: network topology, threshold T, number of clients k, links list L
Output: number of clients for training K
1: Before every communication round, do
2:   for each link l in L, do
3:     LU ← Compute port utilization based on Eq. 1
4:     if LU > T, then
5:       L.remove (l)
8:   end if
9: end for
10: Topo.update (L)
11: for each node i in N, do
12:   Ps_i ← Dijkstra’s algorithm (server, n)
13:   Cpi ← Compute the cost of Ps_i based on Eq. 2
14: end for
15: Cp. sort ()
16: N.sort (Cp)
17: Select first K nodes in N
```

V. EXPERIMENTS AND RESULTS

This section is divided into two parts, in which we describe
the simulation environment and present our results. The pro-
posed Federated Learning architecture in an SDN-based IoT
context is validated based on several key network features.

A. SIMULATION ENVIRONMENT

In this work, we are primarily concerned with enabling
IoT gateways to provide a reliable and robust network
for distributed intelligence. We use Federated Learning to
implement distributed intelligence. A multilayer perceptron
called 2NN, with two hidden layers each consisting of
200 neurons, was used in [5]. A ReLu activation func-
tion is employed for each neuron. The activation func-
tion for the output layer is a SoftMax function, and we
use a cross-entropy loss function. The MNIST dataset has
$28 \times 28 = 784$ inputs and 10 outputs, meaning that the 2NN
has $(784 + 1) - 200 + (200 + 1) - 200 + (200 + 1) - 10 = 199,210$ trainable parameters.

For the SDN network, each controller was assigned two
switches and two hosts per switch. The IoT gateway was
run on a Raspberry Pi 4 Model B. The Raspberry Pi ran a
Raspbian OS, based on Debian OS. The server was run on
a 64-bit laptop with Ubuntu 14.04.5 LTS. To simulate the
SDN switch function, Vswitch was installed on a Raspberry
Pi 2. Figure 3 shows an example of the simulation topology,
consisting of 15 gateways, one server, two controllers, and
20 switches.

To implement our proposed controller, Floodlight [27],
an open-source OpenFlow controller, was used. The green
hatched modules in Figure 1 were taken directly from the Java
source code of Floodlight. The other software modules were
written in Java.

B. COMPARATIVE RESULTS

For our experimental results and analyses, the following
metrics were considered: delay, response time, throughput,
and network convergence analysis. These four parameters are
considered to be significant for validating the performance
of the IoT and the network. Our distributed SDN-IoT is
compared with a centralized SDN-IoT architecture and a
traditional architecture (without SDN).

![FIGURE 4. Delay for varying numbers of IoT gateways.](image)

![FIGURE 5. Delay for varying numbers of communication rounds.](image)

1) AVERAGE DELAY

One of the most important metrics for measuring quality in
computer network environments is delay. To achieve better
quality of service, delay must be reduced: a well-designed
network should have a low delay. Figure 4 shows the variance
in the delay as a function of the number of IoT gateways
(Federated Learning workers) for one round of communi-
cation. We compared a system without SDN, systems with

42880
one and two controllers, and a system with two controllers using the node selection algorithm, as shown in Figure 4. The results indicate that the use of SDN in the IoT is essential. The proposed distributed SDN-IoT has a lower delay than the centralized SDN-IoT. The delay is higher in the centralized SDN design since the entire network load falls on a single controller. As there are multiple controllers in the proposed architecture, the delay is reduced. The proposed node selection algorithm is applied to the distributed SDN system, and proactively forwards the packets over links that are less congested. It therefore delivers the packets faster than the existing system. Figure 5 shows the total delay for 20 IoT gateways (Federated Learning workers) as a function of the number of communication rounds. We see that the delay in the proposed distributed SDN augmented by the node selection algorithm for many communication rounds is much lower than for the other systems.

2) RESPONSE TIME
Response time refers to the total time that passes before a response to a request is received, and is one of the metrics commonly used to evaluate the scalability of a system. To evaluate the scalability of our proposed system, we measured the response times with respect to different packet-in message rates and repeated the experiment for different numbers of controllers, as shown in Figure 6. It can be seen that the response time rises gradually up to a certain point, and then suddenly increases by a large amount. This is because the number of incoming messages exceeds the capacity of the system, causing a queue of incoming messages. Figure 6 shows that increasing the number of controllers helps the system to process more incoming messages, thus improving its scalability.

3) THROUGHPUT
We evaluated the throughput of the system by sending 10,000 packet-in messages in succession to each controller and recording the throughput as a function of the number of controllers, as shown in Figure 7. We notice that increasing the number of controllers increases the throughput in an approximately linear way. The reasons for this are that (i) in the proposed system, each controller primarily handles traffic flows within its own domain, and (ii) the controllers do not exchange information with each other while responding to packet-in messages.

4) CONTROLLER CONVERGENCE
Controller convergence is defined as the time taken for the controllers to synchronize and obtain a holistic view of the network after discovering the links and updating the topology. Table 3 shows the controller convergence times for different numbers of controllers.

| Number of controllers | 2   | 3   | 4   | 5   |
|-----------------------|-----|-----|-----|-----|
| Time taken (s)        | 0.40| 0.52| 0.59| 0.60|
| Standard error of the mean (SEM) | 0.05| 0.08| 0.04| 0.07|

As the number of controllers per network configuration increases, the proposed model converges within half to six-tenths of a second, on average. Except for the network with two controllers, which takes less than half a second to converge, the standard error of the mean (SEM) for the 10 repetitions is generally the same for all network configurations.

VI. CONCLUSION
In this paper, we propose an architecture that provides distributed intelligence through IoT gateways using SDN technology. SDN enables the reuse of the same gateway functions in different flows for different applications. Thus, an IoT gateway can play the role of a worker for different Federated Learning models. As a first step in this direction, we developed a prototype of a distributed SDN-based controller to provide distributed intelligence at the edge of the network for self-organizing structuring. In the second step, we developed an algorithm to select clients for the training process in Federated Learning based on the context information of the network. Finally, we implemented Federated Learning in our distributed SDN-based architecture through the gateways.
and conducted a comprehensive and detailed evaluation of the system from various perspectives. The results show that there is a significant advantage from building a distributed SDN-based IoT gateway. A series of large-scale emulations will be needed in future research to uncover the important parameters that affect performance in a multi-domain distributed environment.

REFERENCES

[1] V. Ganti, W. Wang, and J. Yang, “IoT Growth Demands Rethink of Long-Term Storage Strategies,” IEEE Commun. Mag., vol. 53, no. 1, pp. 148–154, Jan. 2015.

[2] M. Jammal, T. Singh, A. Shami, R. Asli, and Y. Li, “Software defined networking: State of the art and research challenges,” Comput. Netw., vol. 121, pp. 100–111, Feb. 2017, doi: 10.1016/j.comnet.2017.03.019.

[3] H. Dinh, C. Lee, D. Niyato, and P. Wang, “A survey of mobile cloud computing: Architecture, applications, and approaches,” Wireless Commun. Mobile Comput., vol. 13, no. 18, pp. 1587–1611, Dec. 2013, doi: 10.1002/wcm.1203.

[4] X. Xiang, C. Lin, and X. Chen, “EcoPlan: Energy-efficient downlink and uplink data transmission in mobile cloud computing,” Wireless Netw., vol. 21, no. 2, pp. 453–466, Feb. 2015, doi: 10.1007/s11284-014-0795-3.

[5] B. Almadani, A. Beg, and A. Mahmoud, “DSF: A distributed SDN control plane framework for the east/west interface,” IEEE Access, vol. 9, pp. 26735–26754, 2021, doi: 10.1109/ACCESS.2021.3057690.

[6] Y. Li, X. Su, J. Riekki, T. Kanter, and R. Rahmani, “A SDN-based architecture for horizontal Internet of Things services,” in Proc. IEEE Int. Conf. Commun. (ICC), May 2016, pp. 1–7, doi: 10.1109/ICC.2016.7511053.

[7] O. Salman, I. Elhajj, A. Kayssi, and A. Chehab, “Edge computing enabling the Internet of Things,” in Proc. IEEE 2nd World Forum Internet Things (WF-IoT), Dec. 2015, pp. 603–608, doi: 10.1109/WF-IoT.2015.7389122.

[8] R. Muñoz, R. Vilalta, N. Yoshikane, R. Casellas, R. Martínez, T. Tsutriani, and I. Motti, “Integration of IoT, transport SDN, and edge/cloud computing for dynamic distribution of IoT analytics and efficient use of network resources,” J. Lightw. Technol., vol. 36, no. 7, pp. 1420–1428, Apr. 1, 2018, doi: 10.1109/JLT.2018.2800660.

[9] A. Akhri, P. Berthou, A. Gokhale, and S. Abdellatif, “Publish/subscribe-enabled software defined networking for efficient and scalable IoT communications,” IEEE Commun. Mag., vol. 53, no. 9, pp. 48–54, Sep. 2015, doi: 10.1109/MCOM.2015.7263372.

[10] X. Li, X. Su, J. Riekki, T. Kanter, and R. Rahmani, “A distributed software defined networking model to improve the scalability and quality of services for flexible green energy internet for smart grid systems,” Comput. Electr. Eng., vol. 84, Jun. 2020, Art. no. 106634, doi: 10.1016/j.compeleceng.2020.106634.

[11] C. Gonzalez, O. Flauzac, F. Nolot, and A. Jara, “A novel distributed SDN-secured architecture for the IoT,” in Proc. Int. Conf. Distrib. Comput. Sensor Syst. (DCOSS), May 2016, pp. 244–249, doi: 10.1109/DCOSS.2016.22.

[12] F. Bannour, S. Souihi, and A. Mellouk, “Distributed SDN control: Survey, taxonomy, and challenges,” IEEE Commun. Surveys Tuts., vol. 20, no. 1, pp. 333–354, 1st Quart., 2018, doi: 10.1109/COMST.2017.2782482.

[13] Y. E. Oktian, S. Lee, H. Lee, and J. Lam, “Distributed SDN controller system: A survey on design choice,” Comput. Netw., vol. 121, pp. 100–111, Jul. 2017, doi: 10.1016/j.comnet.2017.04.038.

[14] Z. Tang, S. Shi, X. Chu, W. Wang, and B. Li, “Communication-efficient distributed deep learning: A comprehensive survey,” Mar. 2020, arXiv:2003.06307. Accessed: Oct. 5, 2021.

[15] Q. Ho, J. Cipar, H. Cui, S. Lee, J. K. Kim, P. B. Gibbons, G. A. Gibson, G. Ganger, and E. P. Xing, “More effective distributed ML via a stale synchronous parallel parameter server,” in Proc. Adv. Neural Inf. Process. Syst., vol. 26, 2013, pp. 1223–1231. Accessed: Oct. 8, 2021. [Online]. Available: https://proceedings.neurips.cc/paper/2013/hash/5b7b359b6c0caae2dd08de0f097d1062-Abstract.html.

[16] L. G. Valiant, “A bridging model for parallel computation,” Commun. ACM, vol. 33, no. 8, pp. 103–111, Aug. 1990, doi: 10.1145/7973.7981.

[17] K. N. Qureshi, R. Hussain, and G. Jeon, “A distributed software defined networking model to improve the scalability and quality of services for flexible green energy internet for smart grid systems,” in Proc. Adv. Neural Inf. Process. Syst., vol. 28, 2015, pp. 2373–2375. Accessed: Oct. 8, 2021. [Online]. Available: https://proceedings.neurips.cc/paper/2015/hash/452bf2089f913229668557278b6feef-Abstract.html.

[18] S. S. Stich, “Local SGD converges fast and communicates little,” Mar. 2019, arXiv:1805.09767. Accessed: Oct. 8, 2021.

[19] R. Firouzi, R. Rahmani, and T. Kanter, “An autonomic IoT gateway for smart home using fuzzy logic reasoner,” Proc. Comput. Sci., vol. 177, pp. 102–111, Jan. 2020, doi: 10.1016/j.procs.2020.10.017.

[20] R. Duc, C. Wu, Y. Tsutomu, and Y. Ji, “Context-aware clustering for SDN enabled network,” in Proc. IEEE 28th Int. Conf. Netw. Protocols (ICNP), Oct. 2020, pp. 1–6, doi: 10.1109/ICNP49622.2020.9259403.

[21] S. Attarha, K. Haji Hosseiny, G. Mirjalily, and K. Mizani, “A load balanced congestion aware routing mechanism for software defined networks,” in Proc. Iranian Conf. Electr. Eng. (ICEE), May 2017, pp. 2206–2210, doi: 10.1109/IranianCEE.2017.7985428.

[22] Floodlight Documentation—Floodlight Controller—Confluence. Accessed: Apr. 7, 2022. [Online]. Available: https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/overview/homepageId=1343545

RAMIN FIROUZI received the M.Sc. degree in computer science from Tarbiat Modares University, Tehran, Iran. He is currently pursuing the Ph.D. degree in computer science with the Department of Computer and Systems Sciences, Stockholm University, Sweden. His research interests include distributed computing systems, edge computing, distributed intelligence, and distributed IoT.

RAHIM RAHMANI (Senior Member, IEEE) received the Ph.D. (technical) degree in computer engineering from Mid Sweden University. He is currently a Professor with the Department of Computer and Systems Sciences, Stockholm University, Sweden. He was one of the driving forces of a new research group focused on distributed IoT and immersive networking with the University of Stockholm. His current research interests include distributed data processing in distributed IoT, cognitive IoT, event knowledge processing and large-scale decentralized systems (blockchain), decentralization and spatial computing for real metaverse, pervasive computing, and adversarial machine learning.