Discover Internet of Things

Research

A new intelligent scheduler to improve reactive OpenFlow communication in SDN-based IoT data streams

Ernando Batista[1,2] · Brenno Alencar[1,2] · Eliabe Silva[1] · João Canário[4] · Ricardo A. Rios[2] · Schahram Dustdar[3] · Gustavo Figueiredo[2] · Cássio Prazeres[2,3]

Received: 11 June 2024 / Accepted: 9 September 2024
Published online: 01 October 2024

Abstract
The significant adoption of the Internet of Things (IoT) has increased the challenges in providing adequate IoT infrastructures meeting essential requirements, such as dynamicity networks and low latency. In this context, the Software-Defined Networking (SDN) paradigm and the OpenFlow protocol provide new possibilities for IoT networks. Based on the global view of network elements enabled by the Controller, SDN allows the programmability and control of the infrastructure according to the actual demands of applications. The OpenFlow protocol defines the exchange of messages between controllers and switches, enabling communication and network control. OpenFlow implements three operation modes: proactive, reactive, and hybrid. Due to the dynamic characteristic of the IoT data stream, the reactive mode is mainly used and indicated for IoT environments. As the OpenFlow controller installs rules dynamically, there is no need to know the network's sources, destinations, and paths in advance. Although reactive mode introduces dynamicity, it can generate additional delay due to switch-controller communication. This delay increases the response time of the IoT data stream. We propose an SDN-IoT scheduler based on Deep Neural Networks (DNN) to predict the time between data stream changes from IoT devices and install rules in advance, suppressing the existing delay in reactive mode. The proposal automatically uses previous data from the IoT data stream to calculate the time of the following communication from IoT devices. Our results indicate that predicting IoT data stream changes and installing OpenFlow rules in advance reduced about 51% of communications response time.

Keywords OpenFlow · Software-Defined Networking · Deep Neural Network · LSTM · Internet of Things · Intelligent scheduler

1 Introduction

Internet of Things (IoT) enables the combination and coordination of physical and virtual objects connected to the Internet, using different but interoperable types of protocols and communication devices [1]. The Internet of Things is pervasive enough to enable its use in a large number of applications. Such applications can be grouped into domains, and depending on which "thing" (IoT device/sensor) may vary and play different roles. In various industries, it can

✉ Ernando Batista, ernando.passos@ifba.edu.br; Brenno Alencar, brenno.mello@ufba.br; Eliabe Silva, silva.eliabe@ufba.br; João Canário, joao.canario@stone.com.br; Ricardo A. Rios, ricardoar@ufba.br; Schahram Dustdar, dustdar@tuwien.ac.at; Gustavo Figueiredo, gustavobf@ufba.br; Cássio Prazeres, prazeres@ufba.br | [1]Computer Science Department, Federal Institute of Bahia (IFBA), Bahia, Brazil. [2]WISER: Web, Internet and Intelligent Systems Research Group, Computer Science Department, Federal University of Bahia (UFBA), Bahia, Brazil. [3]Distributed Systems Group, TU Wien, Vienna, Austria. [4]Stone Co., Bahia, Brazil.

be a product or equipment; in intelligent environments, it can refer to a tree, room, or building; in health systems, it can be an equipment that collects patient data. Taking into account this set of things that can be connected to the Internet, an IoT environment can be deployed in different domains such as healthcare, transportation and logistics, agriculture, security and home automation [1–3].

To provide and manage these diverse IoT applications, a set of services must be provided that meet the requirements and architectural challenges imposed by the applications. Among them, low latency and dynamic infrastructure requirements stand out [4–6]. Low latency and RTT (Round-Trip Time) is a crucial factor in meeting the performance requirements of IoT infrastructures, and managing network elements plays a pivotal role in achieving it. To ensure low RTT in IoT applications across various domains, extensive research has been conducted in recent years. For example, in the field of UAVs, low RTT ensures self-driving and precise responses to various eventualities [7]. In IoT networks, it enables the detection of anomalies and the proper execution of services [8]. In Smart Homes, low RTT and latency support the architectures for the operation of autonomous intelligent objects [9]. Additionally, in e-Health systems, low RTT allows quick decision-making in severe health cases [10].

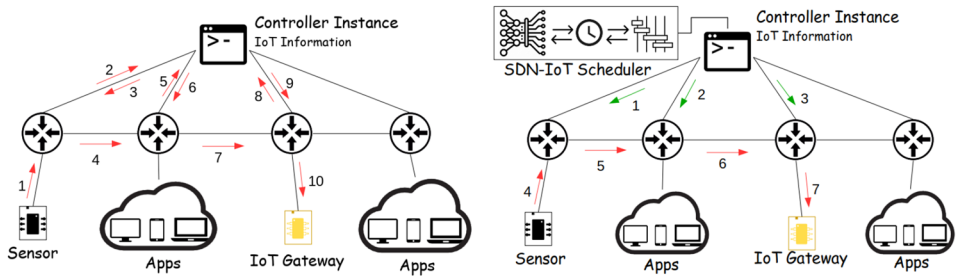
Regarding the management of network elements, the Software Defined Networking (SDN) paradigm provides the ability to implement programmability and to give a proper treatment of the network traffic. SDN allows the separation of switches control logic and the data traffic forwarding actions. Thus, while switches become just forwarding devices, the control of the network is carried out in a centralized way by a programmable Controller Network Operating System NOS that communicates with them through the OpenFlow protocol, for example [11, 12].

The OpenFlow protocol standardizes the communication between controllers and switches in SDN architectures. It implements three operation modes: proactive, reactive, and hybrid. In proactive mode, the actions taken by each network node over traffic flows are defined and installed by the controller/network operator in advance. Applying OpenFlow proactively becomes a complex task for most corporate networks, as it is necessary to know in advance all the nodes in the network and their possible combinations in future communications to enable the early installation of rules in the switches. In reactive mode, on the other hand, action rules to be applied in traffic flows are dynamically created in response to traffic changes. The hybrid mode incorporates features from both proactive and reactive modes, dynamically adding new flow entries in response to network conditions or in anticipation of potential traffic changes - as long as they are correctly detected/predicted [11, 13]. Ensuring efficient operation in IoT scenarios using either proactive or hybrid mode is a challenging task, as it requires the controller to have prior knowledge of the data stream, which can be highly variable due to communication fluctuations between nodes (source/destination).

In the reactive mode, the rules in the flow table are dynamically generated according to the actual traffic on each network node. Thus, OpenFlow in reactive mode is traditionally considered the most suitable approach for the dynamic nature of data streams in IoT environments and corporative networks. However, it may generate additional delay due to the switch-controller communication, which in turn, increases response time. This occurs when there is no previous rule compatible with the received packet/request in the flow table of the switch. Thus, the switch sends an OpenFlow request to the Controller, which processes the data received and installs the appropriate rule. This process, which involves the exchange of messages between the controller and switches, is depicted in Fig. 1a. In the example, switches do not initially have OpenFlow rules in their flow tables. Switches should consult the Controller (steps 2–3, 5–6, and 8–9 in Fig. 1a) to obtain the specific rule that dictates how to handle packets from a given flow. This interaction may potentially lead to an increased response time.

To address the issue of increased response time in IoT data streams, we propose an SDN-IoT scheduler that incorporates a prediction method for changes in the data stream. This solution eliminates the extra communication time between switches and the controller, resulting in a reduction in the final response time between IoT gateways and sensors. Our solution prevents the scenarios depicted in Fig. 1a and improves response times for IoT data streams. The core idea of our solution is to estimate the timing of future communication by analyzing changes in the IoT data stream. The SDN-IoT scheduler utilizes a Deep Neural Network (DNN) architecture for time series prediction called Long Short-Term Memory (LSTM), as described in Sect. 3.3. Our solution employs an incremental approach to efficiently model IoT data streams. By using concept drift techniques (see Sect. 3.2), the IoT data stream is analyzed to identify the time between data change points. Using this information, the model is trained to predict the next time interval in which the sensor should produce new data. This approach allows for incremental updating of the model in the event of a change in the time frame interval of data sent by the sensors.

Our proposal considers the LSTM model result to previously install the OpenFlow rules, avoiding the increase in the final response time shown in Fig. 1a. When using the IoT data stream prediction and early installation strategy, the communication will be as shown in Fig. 1b. In this case, as the installation of rules (steps 1, 2, and 3) are performed before



(a) Reactive Scenario: data traffic between Controller and switches occurs during communication. The scenario contains the steps necessary for communication between the IoT devices (sensor and IoT Gateway).

(b) Prediction scenario: data traffic between Controller and switches occurs previously to communication. The scenario contains the steps necessary for communication between the IoT devices (sensor and IoT Gateway).

Fig. 1 OpenFlow operation modes

the IoT device request, the response time will be counted only for steps (4, 5, 6 and 7). Therefore, when applying our proposal to the OpenFlow-SDN network, we incorporate hybrid mode of OpenFlow operation: proactive mode for the IoT data streams and reactive mode for other types of network traffic. Our results indicate that predicting IoT data stream changes and installing OpenFlow rules in advance, avoiding the scenarios in Fig. 1a, reduced about 51% of communications response time.

In the SDN architecture, communications between the Controller and switches support both inband and outband communication. This paper focuses on outband communication, where specific communication between the Controller and switches occurs through independent links (see Fig. 1). In this type of communication, latency results are not interfered by impacts arising from overload on conventional network links (between sensors and switches, for example). The SDN architecture also enables switch-Controller communication via conventional links, called inband communication. Although inband communication is traditionally simpler, mainly because it requires fewer links in the network, in experimental environments it is likely to produce latency interference due to background traffic or link overload. Thus, to essentially evaluate the impact of our proposal and its impact on response time, outband communication is used in the scenarios covered in this paper.

The remainder of this paper is organized as follows. Section 2 describes some state-of-the-art works related to our proposal. Section 3 presents concepts related to IoT data streams, and LSTM neural networks. Section 4 presents the problem statement and indicates the impact of additional switch-controller time on IoT data stream response time. Section 5 describes the solution modeling, detailing data preparation, neural network architecture, and the components implemented in the SDN-IoT scheduler, and controller. The design of the experiments and the analysis of the results are presented in Sect. 6. Finally, we present some final remarks and recommendations for future works in Sect. 7.

2 Related work

In recent years, researchers have dedicated efforts to develop solutions in SDN networks aimed at reducing latency in communication, particularly concerning the management of OpenFlow rules and flow tables in switches. These initiatives aim to address the increasing demand for solutions that prioritize low latency, especially in SDN-IoT network contexts. Related works are presented in this Section and, in order to highlight and compare our proposal in contrast with these state-of-the-art works, we consider the following characteristics:

- **Domain**: Referring to the main application domains of the solution;
- **Proposed Strategy**: Strategy used to achieve the final objective of the solution;
- **Algorithm**: Implemented/used machine learning algorithm;
- **Prediction Dinamicity**: The level at which the algorithm/solution adapts to changes in the network data stream/traffic;
- **Data Source**: Specifies the data source utilized for conducting experiments and validating the solutions;
- **Implementation Fidelity**: Referring to the level of fidelity of the solution in comparison to real environments. It is also related to how easy it is to deploy the proposal in a real or production environment, considering codes, libraries, and technologies. As fidelity levels increase, potential adaptations or deployment costs decrease;

- **SDN Controller version**: Version/platform adopted as controller for implementing and evaluating the solution;
- **Proposed solution**: Specifies the main objective of the proposal outlined in the work.

Awan et al. [14] introduce a mechanism for proactively installing OpenFlow rules on SDN network switches. The proposal addresses delays in rule installation for reactive in-band SDN-OpenFlow communications. The paper outlines scenarios where delays in installing OpenFlow rules lead to increased latency and packet loss rates. The solution utilizes statistical network information obtained by the Controller to maintain real-time data on link delays among switches. Using this data, the mechanism identifies the total path delay and delays the packet at the first switch on the route until it reaches the destination with matches along the entire route. The proposal, as described in [14], targets a specific issue in SDN scenarios operating in in-band mode, relying only on network information storage for intelligence. In contrast, our proposed SDN-IoT scheduler operates independently of switch-controller connection type (in-band/out-band) and utilizes intelligence based on machine learning algorithms. Finally, it is worth noting that the prediction and proactive installation of OpenFlow rules by the SDN-IoT scheduler can also prevent the occurrence of the highlighted issue addressed by the authors in [14].

Sanabria-Russo, Alonso-Zarate, and Verikoukis [15] propose a solution for detecting periodic IoT messages and preemptively installing OpenFlow rules in SDN networks. Their approach, known as the Pre-emptive Flow Installation Mechanism (PFIM), installs OpenFlow rules in advance using Phase Registration and Phase Learning modules, where the timing of rule installations is statistically adjusted. The authors present results from real-world scenarios that demonstrate the effectiveness of PFIM in reducing delay, particularly in IoT environments with periodic messages. However, its effectiveness needs to improve in bursty traffic scenarios. In contrast, our approach leverages the learning and intelligence capabilities of the SDN-IoT scheduler, accommodating more dynamic traffic patterns and accounting for the environmental context characteristic of IoT settings.

Cai, Wang, and Tsai [16] propose a solution based on and expanded from PFIM [15] named Dynamic Adjustment for Proactive Flow Installation Mechanism (DAPFIM), focusing on managing SDN-OpenFlow flow tables. The proposal enables Controllers to detect transmission periods in IoT messages, simplifying the management and prior installation of OpenFlow rules. To identify the arrival of new IoT messages, the solution dynamically adjusts the definition of the window among IoT messages in a given flow. Consequently, with each new interaction between IoT and SDN, the proposed algorithms' parameters are updated. The study presents results considering hit rate metrics and the impact of timeout duration, comparing them with previous proposals by the same authors. To validate DAPFIM, the authors apply their proposal to statistical/synthetic data and in a high-level SDN environment, without specifying controller usage, OpenFlow versions, and datasets. Furthermore, the solution's impact on latency in IoT communications is not assessed. In our proposal, we acknowledge that the solution's feasibility is directly tied to the fidelity of its application in the real environment. Therefore, we utilize data from real-world scenarios and two implemented and available versions of SDN controllers, in conjunction with the SDN-IoT scheduler.

Isravel, Silas, and Rajsingh [17] propose a framework for predicting flows in SDN networks. The proposal relies on a model that utilizes the Multivariate Singular Spectrum Analysis (MSSA) time series algorithm, in conjunction with the Randomized Singular Value Decomposition (RSVD) method for computational cost reduction. The authors state that they utilize statistical information on network traffic to predict new flows, although they do not explicitly specify which parameters are utilized. The solution underwent evaluation through simulations, considering the metric of the number of correctly predicted packets (visualized in time series) and an analysis of the computational overhead of the solution. Additionally, the solution was compared with the LSTM algorithm, albeit without providing a detailed description of the specific architecture of the LSTM network used for comparison. In contrast to the SDN-IoT scheduler proposal, the paper addresses SDN in a more abstract manner, without specifying controller versions or providing actual code applicable in available controller implementations. Furthermore, there is no direct application of predictions in practical solutions involving OpenFlow rules or latency reduction.

Zafar et al. [18] propose a proactive solution for dynamically migrating switches in SDN-IoT networks. This solution, which is based on traffic prediction, anticipates the migration of switches that may generate excessive traffic volume in the control plane to alternative controllers. The authors conducted simulated experiments using the Mininet tool and the Floodlight controller. Although the proposal by Zafar et al. acts in a scenario similar to the one proposed in this work, there is a distinction in the application, as it is a solution aimed at managing the control plane and processing the SDN controller. This aims to mitigate controller overload and consequently reduce controller-specific response time.

Hajar, Reza and Movahedinia [19] propose a method to improve scalability in Fog Computing by detecting critical IoT requests. They achieve this by employing the Support Vector Machine (SVM) supervised learning algorithm. The

proposed solution determines, based on the detection of critical requests and latency requirements, which fog nodes will be responsible for processing these requests. In situations where all fog nodes are overloaded, the controller forwards requests directly to the Cloud. Fog's response time and resource usage percentage metrics were evaluated and compared with the traditional cloud approach. Although this solution is situated in the IoT-Fog context, it primarily focuses on detecting critical requests and defining the processing destination. In contrast, our SDN-IoT scheduler is based on predicting IoT messages and pre-installing OpenFlow rules. Furthermore, the authors do not consider or implement the solution directly on real SDN-OpenFlow controllers.

Volkov et al. [20] propose a new approach to monitor and predictive analysis of the load imposed on SDN controllers in a 5G communication network. The method's intelligence relies on neural networks and consists of predicting the load on SDN controller based on flows and messages exchanged between the switches and the controller. The paper aims to address the issue of dependence on the hardware and Operating System (OS) on which the controller is implemented. With a high coupling between SDN solutions and the OS, there is an increased probability of controller failure due to incorrect operation between the application and system. The results indicate that it is possible to predict the load received by an SDN controller, as a network application, by remotely analyzing the OpenFlow flows. However, the approach proposed by the authors is focused only on load prediction in controllers considering the independence of the hardware and the Operating System. Our paper proposes, in addition to flow predictions in the IoT context, the performance of a pre-installation of OpenFlow rules in network elements.

Youssef and Rysavy [21] proposes a method for the identification of the most suitable candidate rules in the OpenFlow switches by observing the existing traffic. The rules are extracted from the decision tree build based on the observed NetFlow traffic, and the solution considers data such as gateway, protocol type and QoS routing. In the experiments, the authors employed the Mininet tool to establish the SDN infrastructure and utilized DOROTHEA to generate network traffic based on NetFlow datasets. The solution generates rules from datasets and installs them permanently on OpenFlow switches. The proposal described in [21] differs from our work, as it does not aim to predict the timing of communications between network devices. Thus, without obtaining this data, unfeasible the creation of dynamic OpenFlow rules from a timeout perspective.

Praveen, Sivanesan, and Sathyaprakash [22] propose a heuristic method that takes into account IoT traffic congestion during SDN/OpenFlow flow rule insertions. The authors aggregate more specific OpenFlow rules into more generic rules - considering the QoS requirements of the IoT network. The authors applied the proposed solution in a simulated IoT environment under an SDN network with the Ryu Controller. The simulation results indicate a 12% reduction in delay in SDN switches associated with IoT applications. Although the solution works directly with SDN/OpenFlow rule manipulation, it does not directly focus on time series prediction for IoT environments. Furthermore, it presents a 12% reduction in delay without addressing the impact of timeout on experiments and results.

Isyaku et al. [23] propose an algorithm to manage and dynamically adjust the idle-hard timeout of OpenFlow rules in switches. The proposed solution installs OpenFlow rules with timeout values calculated based on the packet history of a given flow (source/destination). Installing OpenFlow rules with specific timeouts aims to reduce the number of messages between switches and the SDN Controller (packet-in). Consequently, this reduces the demand for processing on the SDN controller. Thus, the solution facilitates the implementation of scalable SDN/OpenFlow networks. The results show that the proposed algorithm reduces the number of packet-in message to 35.2% on average. While this solution manages the timeout of OpenFlow rules, it primarily focuses on optimizing flow table memory (TCAM). Hence, there is no prediction of Data Stream traffic or installation of rules along the entire path between network nodes.

Kuzniar et al. [24] conducted an analysis and performance study of OpenFlow/SDN switches. This work evaluates the data exchange and the respective rule installations. It also checks for possible inconsistencies and vulnerabilities existing in the OpenFlow protocol implementations on real devices. The proposal is based on the evaluation of flows that represent rule updates. For this, the uniformity between what is indicated in the control plane and what is implemented in the data plane is considered. As a result, the authors developed a tool capable of evaluating the performance of the control plane and its interaction with the data plane. Although the paper discusses and analyzes the installation of OpenFlow rules on switches, it does not deal with techniques for pre-installing rules, improving performance, nor with flow prediction to improve the SDN network.

Overall, several studies propose solutions aimed at managing SDN-OpenFlow rules policies using machine learning algorithms ([18, 19, 20, 21]). However, these studies fail to adequately address or evaluate the impact of these solutions in scenarios with real data, a characteristic that we consider fundamental for the viability of these approaches. Two solutions presented in this section ([18, 20]) primarily focus on load prediction and controller balancing. Although approaches for configuring and installing rules to reduce latency have been discussed in ([14, 23]), these proposals do not incorporate

the intelligence provided by machine learning techniques and are based on synthetic data. Additionally, Isravel, Silas and Rajsingh[17] presents a solution based on machine learning for flow prediction with real data, but it is characterized by high-level code and difficulty in applying it to real environments or devices.

From the discussion presented in this section, we understand that there is a lack of intelligent solutions in the context of managing SDN-OpenFlow rules aimed at reducing latency. Furthermore, considering the data collected and presented in Table 1, our proposed SDN-IoT scheduler conducts experiments based on real data, encompassing tests on different versions of Controllers and maintaining a high level of implementation fidelity.

3 Background

In this background section, we provide an overview of IoT data stream concepts, defining their characteristics and possible scenarios in IoT. We then introduce the concept drift, explaining how the nature of IoT data streams can change over time, focusing on the Page-Hinkley algorithm. Finally, we describe the LSTM neural network, describing its architecture and suitability for handling temporal dependencies and non-stationary data. This discussion supports understanding how these three elements interplay with the proposed SDN-IoT scheduler and LSTM architecture.

3.1 Internet of Things data streams

Data Stream is characterized by a sequence of data in flow continuously produced over time and, eventually, at high speed, i.e. $X = \{x_1, x_2, ..., x_t, ..., x_n\}$. Data Stream introduces new characteristics for the V's studied in traditional Big Data: Volume, Velocity, and Volatility. The volume of data produced by devices is typically high, exceeding traditional database systems' processing and storage capabilities. The speed at which the data stream is produced from several devices is very high. In this context, discrete models are usually adopted to reduce the collecting time interval. The variety is related to the diversity of devices and data sources found in IoT systems. Finally, the veracity is also affected once the collected data is frequently affected by environmental and transmission noise [25].

Figure 2 shows scenarios for data stream generation in an IoT environment. In the example depicted in Fig. 2a, the IoT gateway requests data from devices based on a specific time interval. Such communication uses the IoT data stream pattern based on predefined requests, i.e., data from the devices is sent in a continuous stream at the defined time interval. Another communication pattern between IoT devices, Fig. 2b, relies on the occurrence of a predetermined event for the subsequent data sending. In this scenario (event-based IoT data streams), the event can be defined according to the need of the IoT application, such as variation in the temperature value of a sensor or activation of a motion sensor. The event-based data stream pattern is useful, as long as it is appropriately used, to avoid unnecessary requests and consequently consume less resources (storage, network, and processing) [26].

We propose using event-based communication to transfer data between the device and the IoT gateway. This modeling allows prediction algorithms to learn the event occurrence pattern for future communications among IoT devices. Future communications predictions in the proposal use the DNN model called LSTM. Sections 3.3 and 5.2 present further proposal details. Furthermore, our proposal uses the concept drift as an event, defined as a conditional change in the data distribution. Section 3.2 provides a more in-depth discussion of concept drift.

3.2 Concept drift in IoT data stream

The data produced by IoT devices are considered non-stationary due to the dynamic nature of the monitoring environments, thus leading to changes in the data distribution over time, commonly known as concept drift. In summary, the concept drift area studies changes in the conditional distribution of the output of a target variable concerning the input. Although we can estimate changes in advance for specific real-world situations, concept drifts are generally characterized as unexpected and unpredictable [27].

Formally, the concept drift between two points at time t_0 and time t_1 can be defined as follows:

$$\exists X : p_{t0}(X, y) \neq p_{t1}(X, y), \tag{1}$$

Table 1 Comparison related works

Authors	Context	Proposed Strategy	Algorithm	Prediction Dinamicity	Data source	Implementation Fidelity	Evaluated Metrics	SDN Controller version	Solution Proposal
Awan et al.	Agnostic	In-band link delay detection	–	–	Synthetic data	High	Latency and packet loss rate	POX	Adjustment of OpenFlow rules installation
Cai, Wang e Tsai	IoT	Statistical adjustment of IoT message window parameters	–	High	Synthetic data	Low	Hit rate and flow table duration	Not used	Flow table management
Isravel, Silas and Rajsingh	Agnostic	Time series prediction	MSSA	High	Real data	Low	Hit rate and CPU consumption forecast	Not used	Network traffic prediction
Zafar et al.	IoT	Time series prediction	Normalized LMS	High	Synthetic data	High	Controllers load and controller latency	Floodlight	Switch migration among controllers
Hajar, Reza, and Movahedinia	IoT	IoT request detection	SVM	Medium	Not Specified	Medium	Latency and Fog nodes processing	Not used	Improve allocation of critical IoT requests
Volkov et al.	Agnostic	PacketCount and byteCount OpenFlow prediction	Neural network	High	Synthetic data	Low	MSE	Not used	Load monitoring and prediction in the SDN Controller
Youssef and Rysavy	Agnostic	Automatic generation of OpenFlow rules	Decision Tree	High	Synthetic data	High	Decision tree accuracy	Not specified	Automatic generation of OpenFlow rules
Praveen, Sivanesan and Sathyaprakash	Agnostic	Perfect-hit heuristic method	–	–	Synthetic data	High	Latency and packets dropped	Ryu	OpenFlow rules aggregation
Isyaku et al.	Agnostic	Algorithm for traffic-based timeout calculation	–	–	Synthetic data	High	packet in rate	Ryu	Dynamic idle-hard timeout allocation
Kuzniar et al.	Agnostic	Detection and analysis of OpenFlow table updates	–	–	Synthetic data	High	Inconsistency in data and control plane, flow tables occupancy	NOX, ROFL	OpenFlow rules inconsistency
Sanabria-Russo, Alonso-Zarateand Verikoukis	IoT	IoT periodic message detection and adjustment	–	Medium	Real data	High	Delay, number of OpenFlow messages	Ryu	Flow table management
Our Proposal	IoT	Time series prediction	LSTM	High	Real data	High	Latency (RTT), MAE, RMSE	POX, RYU	SDN-IoT OpenFlow scheduler

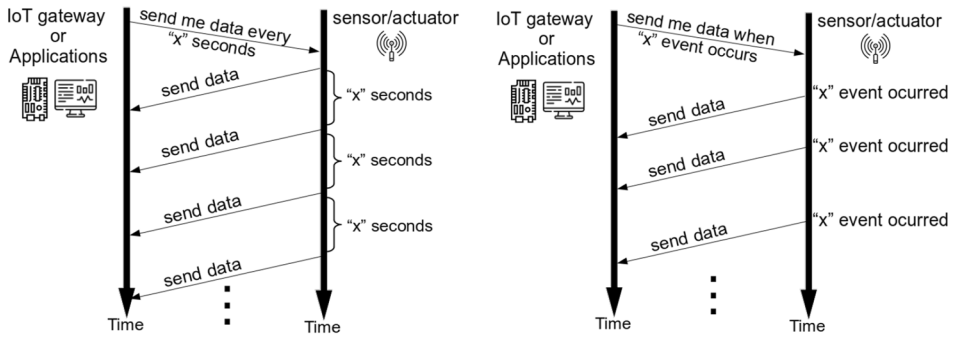


Fig. 2 IoT Data Stream communications

(a) IoT Data Stream communications with the flow based in a period of time.

(b) IoT Data Stream communications based in a predetermined event.

in which p_{t0} is the coexisting distribution of the input variable X and the target variable y at time t_0. Drifts can be understood as changes in data behavior that can occur from different sources: (i) sudden/abrupt, when there is a change from one concept to another; (ii) incremental, when the behavior alternates among several intermediate concepts; gradual, when data behavior slowly changes over time; and (iv) reoccurring, when previous concepts are detected likewise. The main challenge faced by techniques devoted to detecting concept drifts is distinguishing between new concepts and outliers/noises randomly present in real-world data [28].

The Page-Hinkley algorithm (PHT) is a sequential analysis technique used to monitor detection of changes in data [29]. This algorithm was designed to detect changes in the average of a Gaussian signal [30]. In summary, PHT runs a test U_T based on the accumulated difference between the observed values and their average until the present moment, as shown in Equation 2.

$$U_T = \sum_{t=1}^{T} \left(x_t - \bar{x}_T - \delta \right), \tag{2}$$

In such an equation, $\bar{x}_T = 1/T \sum_{t=1}^{T} x_t$ and δ correspond to the magnitude of the allowed changes. To detect changes, PHT calculates the minimum value of $U_T : m_T = \min(U_t, t = 1...T)$ and monitors the difference between U_T and $m_T : PH_T = U_T - m_T$. Next, Concept drift is judged when PH_T becomes greater than the predefined parameter λ, i.e.,

$$PH_T > \lambda. \tag{3}$$

The λ limit sets the allowable false alarm rate. Increasing λ results in fewer false alarms; however, it may lose or delay some detections. The control of this detection limit parameter makes it possible to establish a balance between false alarms and the delay in concept drift detections.

3.3 Long Short-Term Memory (LSTM)

The LSTM architecture is a Recurrent Neural Network (RNN) introduced by Hochreiter and Schmidhuber (1997) [31] to learn long-term dependencies and keep (remember) information for more extended periods. LSTM model is organized in the form of a chain structure with four interaction layers with a single communication method.

A typical LSTM network is made up of blocks of memory called cells. By processing information, two pieces of information are transferred to the next cell: its current and hidden states. Essentially, cells are composed of gates used to extract implicit information from the input data and control memorizing process, i.e., to balance new and past information processed by the network.

Figure 3 illustrates the structure of an LSTM neural network. The first part identifies unnecessary information that will be removed from the cell in the first step. A sigmoid function is usually considered in this step to process the output yielded from previous cell ($H_t - 1$). This step also processes the current input (X_t). Furthermore, the sigmoid function determines which part of the old output will be discarded. This process is performed by forgetting record f_t, where f_t is a vector with values ranging from 0 to 1, corresponding to each number in the state of cell C_{t-1} in Fig. 3.

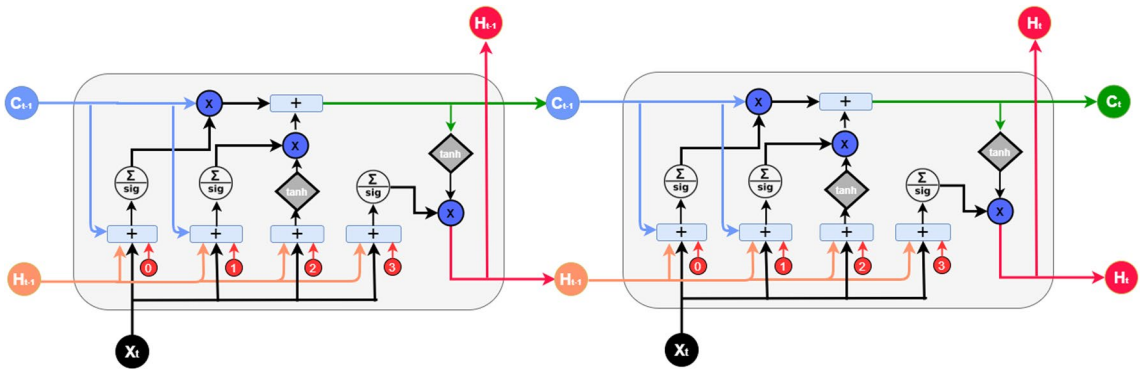


Fig. 3 Structure composition of the LSTM neural network

$$f_t = \sigma\left(W_f\left[h_{t-1}, X_t\right] + b_f\right). \tag{4}$$

In Eq. 4, σ is the sigmoid function, and W_f and b_f are the matrices of weights and biases, respectively.

In the next step, the LSTM network decides how the new entry information (X_t) is used to update the cell state. This step is based on sigmoid and hyperbolic tangent (tanh) layers. First, the sigmoid layer decides whether the new information should be updated or ignored (0 or 1). In the next step, the tanh function weights the transmitted values, deciding their importance level (– 1 to 1). The two values are multiplied to update the cell's new state. This new memory is then added to the old memory C_{t-1}, resulting in C_t.

$$i_t = \sigma\left(W_i\left[h_{t-1}, X_t\right] + b_i\right), \tag{5}$$

$$N_t = \tanh\left(W_n\left[h_{t-1}, X_t\right] + b_n\right), \tag{6}$$

$$C_t = C_{t-1} f_t + N_t i_t. \tag{7}$$

In Eqs. 5, 6 and 7, C_{t-1} and C_t are the states of the cell at time $t-1$ and t. While W and b are the weights and bias matrices, respectively, of the cell state.

In the final step, the output values (h_t) are based on the state of the output cell (O_t). However, it is a filtered version. Initially, a sigmoid layer decides which parts of the cell state get to the output. Next, the output of the sigmoid part (O_t) is multiplied by the new values created by the cell state in the tanh layer (C_t), with the value varying between −1 and 1.

$$O_t = \sigma\left(W_0[h_{t-1}, X_t] + b_0\right), \tag{8}$$

$$h_t = O_t \tanh\left(C_t\right). \tag{9}$$

In Eqs. 8 and 9, W_0 and b_0 are the matrices of weights and biases, respectively, of the output record.

4 Experimental problem characterization

This section presents a scenario illustrating the impact of reactive mode communication (in SDN networks) on the response time in IoT applications. By characterizing the problem, it also improves understanding of the organization of the components involved in IoT scenarios. Additionally, this section covers the modeling and implementation of the fundamental components, namely the sensor and the IoT gateway. We used the Mininet emulator [32] to conduct the experiment and modeled the main components as described in the next subsections.

To demonstrate the inference of communication requirements in such IoT devices, we performed an experiment that aimed to show the impact that communication between the controller and OpenFlow switches has on response time in the IoT data stream. For that, we proposed two scenarios: (i) default SDN/OpenFlow environment in reactive operating

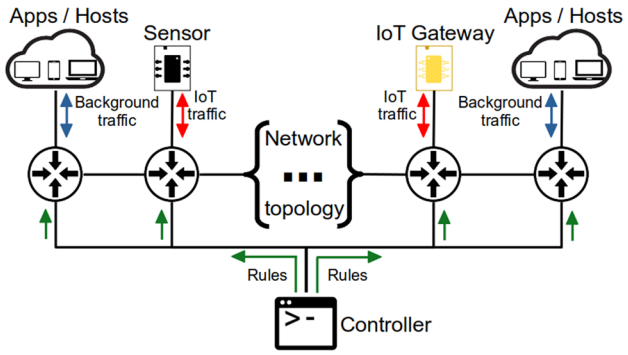


Fig. 4 Main components involved in the experiments with their possible interactions in the network (IoT traffic, background traffic, and rule installation)

mode, (ii) SDN/OpenFlow environment with rules statically fixed on the switches. Notably, the statically fixed rules installation strategy is used only in this experiment for problem statement purposes. In this case, the use in production environments becomes unfeasible as it would flood the topology switches with fixed rules.

4.1 Virtual sensor modeling

The virtual sensor is the primary component within an IoT environment. It involves the implementation that enables the virtual sensor to emulate the behavior of a real sensor. This encompasses the utilization of protocols for the exchange of IoT messages as well as the publication of IoT data from real datasets. For modeling and simulation of a virtual sensor, some basic aspects must be respected: (i) simulation of sample collection (data) consumed from real datasets and data storage of short period; (ii) implantation of the communication protocol responsible for controlling the information flow among devices (sensors, IoT gateways, servers, and applications).

The behavior of a virtual sensor is performed by a service created in the Python programming language. This service, performed at Mininet hosts (in a real environment running on Arduino devices) associated with the libraries and technologies used in the execution process, denotes to the Mininet host the similar behaviors of a real sensor.

4.2 IoT gateway Modeling

The basic management node performs its functionalities through IoT services (storage, security, semantic enrichment, among others), in this paper, used with the storage service. In order to model an IoT Gateway in the Mininet, some basic requirements must be met: (i) use of a framework that implements of the Open Services Gateway Initiative (OSGi) specifications (in this modeling, Apache ServiceMix); (ii) implementation of IoT services in the OSGi framework; (iii) implementation of a communication protocol responsible for controlling the information flow among devices.

IoT gateways and sensors/actuators communicate with each other through message exchange, producing data stream at different times. Such messages transmitted by these devices serve as support for different applications, and based on this IoT data, decisions can be made [33]. Hence, the data stream between them must occur in the shortest time possible, as the sooner the sensor data reaches the IoT gateway, the more better it can make decisions and send back responses to the IoT environment. Additionnaly, Batista et al. [26] presents a comprehensive analysis and description of device implementation, IoT message exchange, and standardization, providing depth insights into these aspects.

4.3 Analysis of evidences

For the experiments described in this Section (see Fig. 4), a set of 25 virtual sensors (generating pseudo samples) and 2 IoT gateways (running storage services) were used. The network infrastructure of the experiments is composed of 10 OpenFlow switches connected to the POX-SDN Controller, constituting a hybrid (combining tree and linear topology) topology with links varying between 5 and 10 MB in capacity (capable of supporting IoT and background traffic without overloads). To perform this experiments, Mininet is used, which is a tool for emulating networks, making it possible to use technologies, protocols and data formats which are widely used in the IoT context [32].

To standardize the data stream among IoT devices, The Accessible Thing Universe Protocol (TATU) [26] was employed. The TATU protocol facilitates the utilization of IoT data streams following traffic patterns: (i) Flow-based: for IoT data streams based on predefined time requests; (ii) GET-based: for IoT data streams where the IoT Gateway

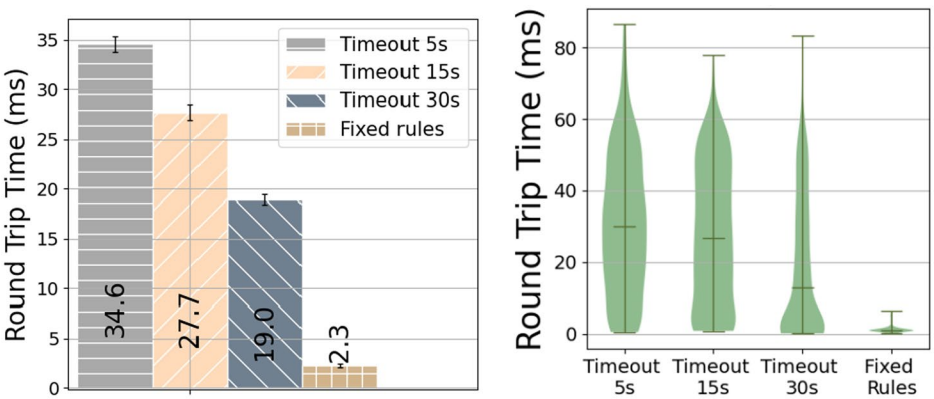


Fig. 5 Experimental problem scenarios

(a) Scenario with timeout variations compared to fixed rules. Averages are grouped by timeout value (5s, 10s, and 15s) and fixed rules. The graph shows the results obtained for the RTT metric according to the timeout variation.

(b) Scenario with timeout variations compared to fixed rules. Data behavior is grouped by timeout value (5s, 15s, and 30s) and fixed rules. The graph visually presents the difference between fixed rules and OpenFlow reactive with different timeouts.

requests data when necessary and the sensor responds to the request; (iii) Event-Based: for IoT data streams associated with the occurrence of specific events. With the objective of providing different temporal patterns of communication to highlight the impact of the reactive communication mode (in SDN networks) on the response time, the specific experiments in this section utilized synthetically generated data in conjunction with the Flow-based TATU protocol. It is important to emphasize that the traffic pattern can be determined based on the application/scenario (e.g., see Fig. 2, and Section 6 with Event-based IoT data streams), and particularly in this section, the Flow-based TATU protocol was utilized for experimental characterization purposes. Moreover, for these experiments, the temporal pattern of IoT data stream was modeled according to a statistical function following the normal probability distribution with a mean defined for 30 (sec).

Figure 5a presents the RTT metric results according to the OpenFlow rules timeout variation (5 s, 15 s, and 30 s). Figure 5a indicates that the longer the timeout, the lower the RTT, as the possibility of the IoT traffic reaching the rule already installed in the network path increases. Furthermore, as expected, defining the fixedly installed rules takes less time than any other configuration. Figure 5b shows the behavior (via boxplot) of the result set for each configuration, making it possible to identify where most of the RTT obtained in the IoT data stream are concentrated (and the average). In this case (Fig. 5b), the results indicate that the RTT metric did not find outliers for the fixed rules scenario, with the majority between 0 and 2 s (reinforcing the data in Fig. 5a). In the following sections, our scheduler proposal to improve the problem presented previously will be discussed in detail.

5 Modeling

This section presents the steps and components in preparing data for machine learning applications, focusing on implementing LSTM neural network architectures and developing the SDN-IoT scheduler. We detail the data preparation process, focusing on the proposed model's activity flow. Next, we describe the architecture and implementation of the proposed LSTM networks, highlighting their capacity to handle sequential data and indicating their association with the SDN-IoT solution. Additionally, we address the modeling of SDN-IoT schedulers, presenting components, indicating operations, and describing interaction with the IoT environment.

5.1 Data preparation

The quality of data must be considered for the context of a machine learning solution. In the presented problem, we model data as time series through pre-processing and cleaning the data collected by the sensors.

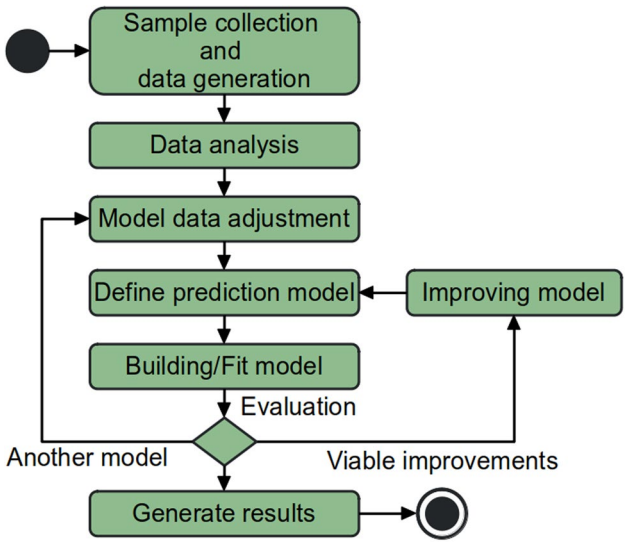


Fig. 6 Activity flow of the proposed model. The diagram shows the activities performed (starting from data collection) to obtain the final results presented in Sect. 6

Figure 6 shows the steps from creating the dataset to obtaining the results. In the data analysis stage, it is important to identify and remove eventual incorrect measurement data (due to sensor failures or problems in data handling) [34, 35]. Removing possible inconsistencies from the data prevents the machine learning model from being driven to incorrect learning patterns that often result in overfitting or underfitting. In this case, the data must be normalized with scales appropriate to the model (e.g., between 0 and 1) as described by Eqs. (1) and (2) (standard deviation and normalization)

$$x_{std} = \frac{x - x_{min}}{\left(x_{max} - x_{min}\right)}, \tag{10}$$

$$x_{scaled} = x_{std} * (x_{max} - x_{min}) + x_{min}, \tag{11}$$

where x_{min} and x_{max} are the minimum and maximum values of a given attribute, x is the current value, x_{scaled} is the normalized value, and x_{std} is the calculated value for the standard deviation. To use machine learning techniques in time series, the data must be prepared to serve as input to possible models, as the prediction of values in time series is based on previous data (previous windows) of the same time series [36, 37]. In a time series, the predicted value is considered to be defined by $f\left(x_1, x_2, \ldots, x_{T-h}\right)$, where h is the size of the forecast horizon, and T is the element/window to be predicted. Therefore, to allocate some temporal data to the model, the input must be a sequence of previous windows with the features defined in the machine learning model.

The defined model will iterate through all windows of the training dataset in order to learn from the found patterns. Thus, the model will have the following prediction function at the end of the building/fit stage (Fig. 6).

$$\begin{aligned} f(t+h) =& f\left(x_{(t-n+1)} x_{(t-n+2)}, \cdots, x_{(t)}\right), \\ & \forall t \in \{n, n+1, \ldots, T-h\}. \end{aligned} \tag{12}$$

Considering the correct preparation of the model, the output produced after receiving a data window of size n will be similar to the values used during the training stage (as intended). Therefore, if the predicted values are not adequate, it means that the input has a different pattern than the training set or the model is not yet adapted enough for the prediction solution. For models that do not present adequate predictions, the training adjustment can start through the alert produced by the concept drift detection. The following section will explain how we apply the LSTM neural network in IoT data stream in more detail.

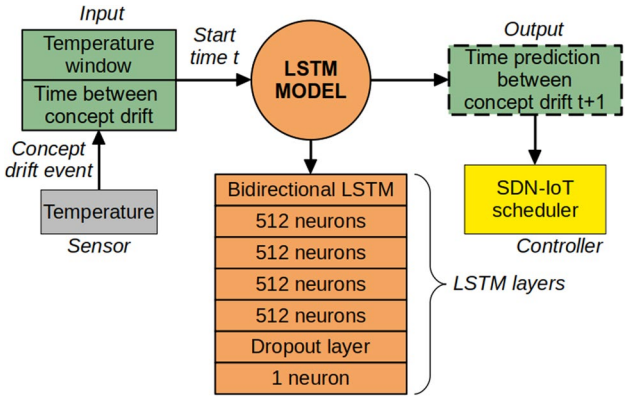


Fig. 7 LSTM model architecture

5.2 LSTM model architecture

This paper proposes investigating the application of LSTM models to predict IoT device communication in SDN networks,[1] considering the unique characteristics of IoT data streams. LSTM is widely used for time series forecasting in applications with long-term dependencies [38–40]. In that regard, LSTM has been applied to optimizing IoT device communications [41–44] due to its architecture focused on learning temporal data. Due to the temporal characteristic of IoT data stream, the problem addressed in this paper aims to make predictions with LSTM from the data stream.

The pre-processing of data for training the neural network is initiated by performing the transformations in the data windows (input_data) for training. From this data, the difference between the observation in time $t-1$with the current observation in time t is calculated. Then, with the difference data, a new series is generated in which the observation in time $t-1$is entered for the observation of time t. After that, the data is divided into training and testing data. Finally, the training and testing data are converted on a scale between -1 and 1 to improve the execution of the activation function applied to the model.

The architecture of the LSTM model used in this work is presented in Fig. 7. LSTM model was trained using data comprising the time intervals between concept drift detections and sensor temperature readings. Concept drift, identified within the temperature data stream generated by the sensors, triggers an alert, along with the associated temperature window where the drift occurred (using TATU protocol based on the concept drift event). This information, stored within Windows, forms the dataset utilized for model training. We built a Bidirectional LSTM Neural Network with two neurons on the entry layer (Temperature and time between Concept Drift), four hidden layers, each one with 512 neurons, a dropout layer (0.75), and one neuron on the output layer (delay). We used "Relu" as the activation function, Adam optimizer with 0,0001 learning, and used 128 epochs with the value 10 to batch size.

LSTM model is designed to forecast the forthcoming communication requirements for transmitting data via the sensor. Leveraging the neural network's predictions, the SDN-IoT scheduler orchestrates the deployment of rules onto switches, facilitating the establishment of communication pathways between the sensor and the IoT gateway.

5.3 SDN-IoT scheduler modeling

The SDN-IoT scheduler and the SDN/OpenFlow controller (Figure 8) are responsible for the machine learning model and rules installation management modules. We implement the SDN-IoT scheduler and it executes its functionalities directly on the same Controller machine (e.g., using Ryu Controller or POX Controller). This approach enables efficient data intermediation and upholds the centralized management proposed by the SDN paradigm. Figure 8 indicates the components that implement and coordinate the operation of the Controllers and the SDN-IoT scheduler.

According to Fig. 8, from the Controller's perspective, there are the following components: **(i) L3/L4 layers protocols** - with the implementation of routing algorithms/protocols; **(ii) OpenFlow libs**—libraries for integration between the Controller and the OpenFlow v1.0 protocol; **(iii) Pox Core / Ryu Core**—defines how will occur the programmability of events, logs, records, and network status.

[1] LSTM Model Code: https://github.com/WiserUFBA/SDN-IoT-Intelligent.

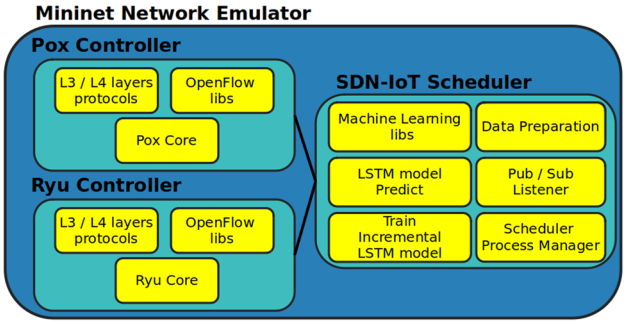


Fig. 8 Controller and Scheduler components

The following components make up the SDN-IoT scheduler: **(i) Machine learning libs** - libraries (e.g., Keras TensorFlow, pandas, numpy) that enable data processing and manipulation of predictive models; **(ii) Data preparation**—performs data treatment and preparation (see Sect, 5.1 for later use in the training module); **(iii) LSTM model predict**—based on the Machine learning libs component, defines and standardizes the use LSTM model, indicating the input/output data format and executing predictions of the time frame interval between concept drifts. (see Sect, 3.3). In addition, it provides the predictions required by other components supporting the proper installation of the rules. The LSTM model is loaded and executed directly in the Controller (with the other modules) and can be used according to Algorithm 1 when the SDN-IoT scheduler needs new predictions; **(iv) Train incremental LSTM model** - performs the training and subsequent creation of the DNN model based on the LSTM architecture; **(v) Pub/Sub listener** - based on the MQTT protocol, this component receives the concept drift (by IoT messages), detected on the data stream, from sensors and sends it to the IoT gateway to trigger the scheduler rule installation process; **(vi) Scheduler process manager**—works together with the Pub/Sub listener, LSTM model, and the Controller. For each new IoT data stream identified, a corresponding process will install the OpenFlow rules in the time indicated by the LSTM predictive model.

Algorithm 1 aims to provide an explanation and overview of some of the main components of the scheduler. In the Algorithm, we treated certain scheduler modules as summarized functions, considering their purpose and sequence of interactions, while other modules were imported libraries (directly called by their functions).

The operation of SDN-IoT scheduler, as presented in Algorithm 1 and Fig. 9, consists of starting a process for each IoT gateway in the environment (Lines 41-43). This process will initially configure MQTT and monitor the received IoT messages (Scheduler process manager component). Upon receiving a message (from the *mqttListener* function) initiating an IoT data stream between the sensor and IoT gateway, the process running in the Scheduler stores/updates data concerning the devices' association (*scheduleList* in Algorithm 1) or ends the communication (STOP TATU message). These messages in the IoT data stream adhere to the standardization provided by the TATU protocol (described and evaluated in [26]). Based on this data, the process invokes the function *buildPredictWindow* (Line 21) to predict the temporal pattern (delay among messages) in the communication among devices using the LSTM model. From the value predicted in *buildPredictWindow* and stored in *scheduleList* params, the process can previously install the rules in the flow tables of the switches in the network path (see processLoop in Fig. 9 and OpenFlow rules installation in Fig. 11).

The identification and scheduling of the installation of OpenFlow rules in a communication flow in the Fog of Things, as presented in Fig. 10, involves initiating a process (in the Controller) for each IoT Gateway in the environment, which will monitor the IoT messages received by the IoT Gateways. In step 1, an IoT message is sent, indicating the start of a flow between the sensor and the IoT Gateway. In step 2, the controller (registered to the topic) identifies the message published via MQTT (step 2) through the Pub/Sub listener module, which is responsible for detecting all messages initiating and modifying flows in the environment. Subsequently, the Controller initiates an individual process for each IoT Gateway, taking into account information about the association of sensors and IoT Gateways (Step 3). Based on this information and in conjunction with the LSTM model, the process identifies the temporal pattern in communication between devices and anticipates the installation of rules in the flow tables of the switches that make up the path.

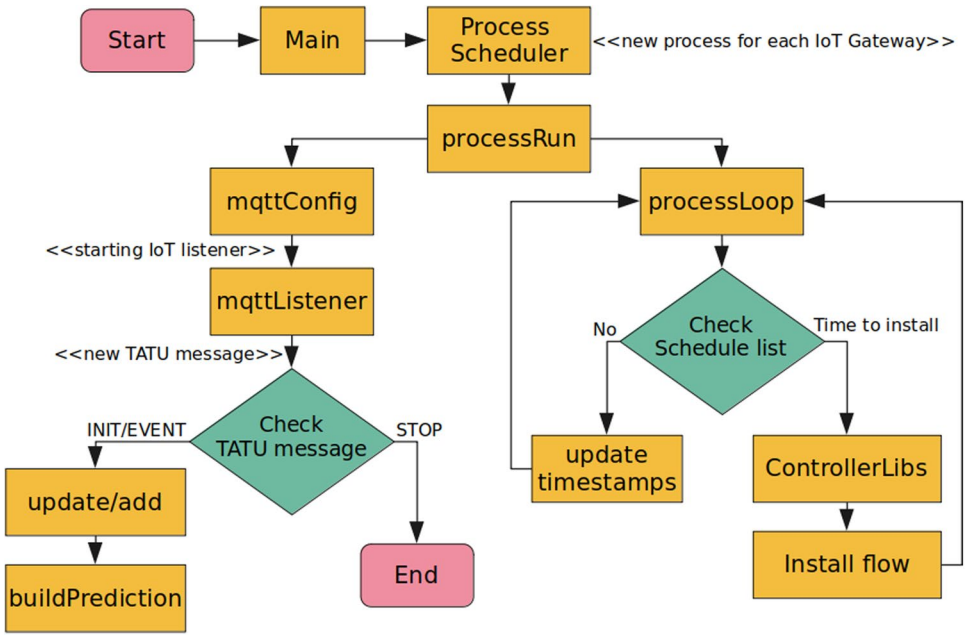


Fig. 9 Diagram with the main tasks performed in the SDN-IoT Scheduler

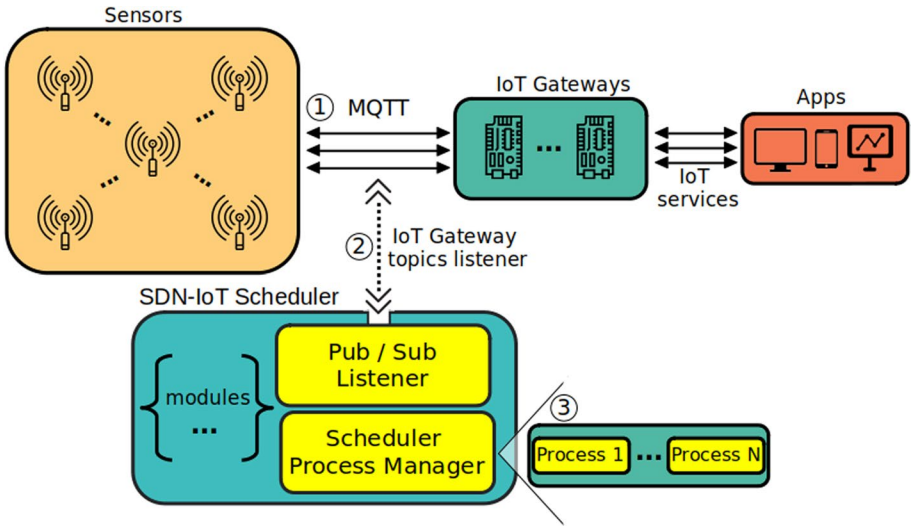


Fig. 10 Scheduler detection

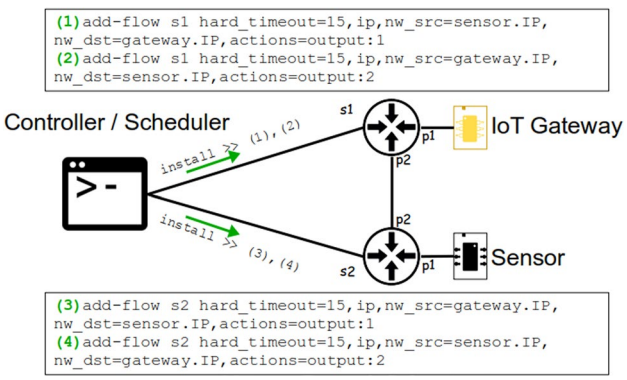


Fig. 11 OpenFlow rules installation scenario. Scenario with data traffic between two IoT devices and basic demonstration of installed rules

Algorithm 1: Main tasks performed in the SDN-IoT Scheduler

```
import PoxCarpController.network, RyuController.network as Controller;
import Controller.OpenFlow_v10;
import Config;
import PahoMQTT;
import TATU;
import LstmModel,BuildParserWindow;
Process Scheduler (gateway):
    PredAttr={predTime, currentTime, initTime};
    scheduleList=[ ];
    Function mqttListener(mqttMsg):
        tatuDevice=TATU.getDevice(mqttMsg);
        tatuDevice.add(new PredAttr());
        tatuType=tatuDevice.getMessageType();
        if tatuType==TATU.INIT or tatuType==TATU.EVT then
            if scheduleList.contains(tatuDevice) then
                scheduleList.update(tatuDevice);
            else
                scheduleList.add(tatuDevice);
            buildPredictWindow(mqttMsg);
        else if tatuType==TATU.STOP then
            scheduleList.remove(tatuDevice);
    Function buildPredictWindow(mqttMsg):
        windowSamples=BuildParserWindow.msgParser(mqttMsg).getWindow();
        windowSamples=BuildParserWindow.normalizing(windowSamples);
        windowSamples=BuildParserWindow.reshape(windowSamples);
        listPred=LSTM.model.load('LSTM.h5').predict(windowSamples);
        listPred=BuildParserWindow.unormalizing(listPred);
        scheduleList.get(TATU.getDevice(mqttMsg)).setPredTime(listPred.mean());
    Function configMQTT():
        client=PahoMQTT.client(protocol=mqtt.MQTTv31);
        client.onMessage = mqttListener;
        client.connect(gateway.IP, Config.brokerPort, Config.KeepAlive);
        client.loopForever();
    Function processRun():
        configMQTT();
        while True do
            foreach scheduleList as L do
                if L.getPredAttr().currentTime >= L.getPredAttr().predTime
                 then
                    Controller.network.set(OpenFlow_V10);
                    Controller.network.installFlow(L.gateway,L.sensor,Config.netTimeout);
                    break;
                L.currentTime=System.timestamp()-L.initTime()
Function main(gatewayList):
    foreach gatewayList as G do
        Process.start(G)
```

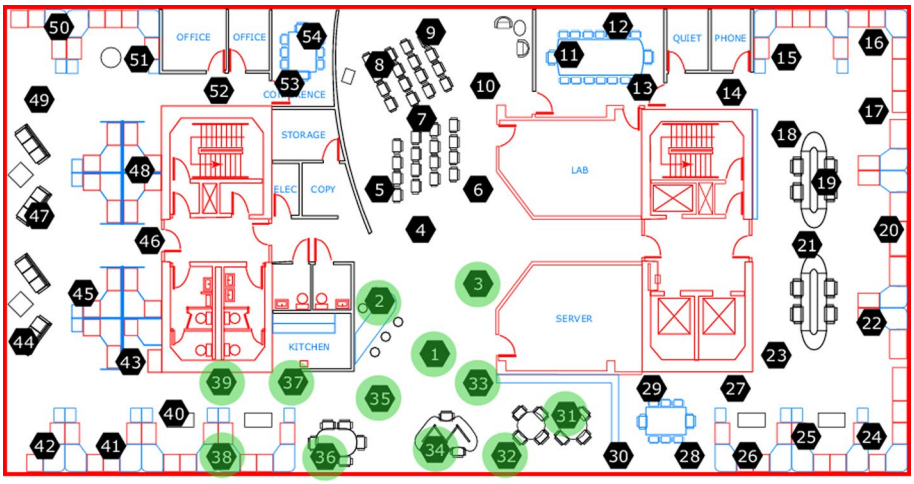


Fig. 12 Location of intel lab data sensors (dataset): Graphical representation of the internal location of sensors in the building

Figure 11 shows an example scenario with prediction and rule installation. Although the installation scheduling is performed individually by the Schedule Process Manager module, the installation of the rules is executed in collaboration with the native codes and libraries of the Controllers (POX and RYU). This approach ensures that the solution aligns seamlessly with the L3 layer algorithms already utilized and implemented in the controller versions. In Fig. 11, the rules are installed on switches that are in the path that the 'future' request will pass. In addition, the rules installed on the switches via prediction already consider and install the necessary rule to return the same request, in which case the roles of the source (nw_src) and destination (nw_dst) devices are reversed, example of rules (1) and (2) installed in the switch (s1).

6 Numerical examples

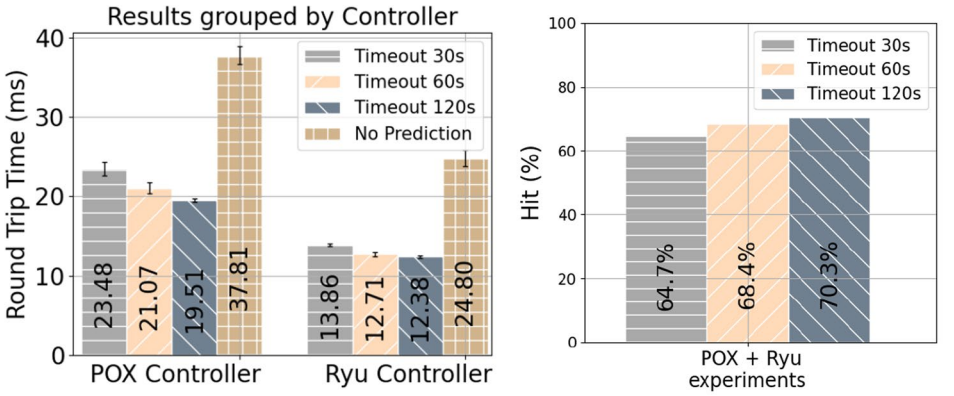
This section describes aspects related to the environment's setup, execution of experiments, and dataset description. In the pre-processing data stage, the adaptation of the original dataset was performed to obtain better results with the LSTM model (described in Sect. 5.2). At this stage, the main activities performed were: (i) removal of records containing samples with inadequate values to the standard of the evaluated sensor; (ii) dataset normalization; (iii) generation of a final dataset containing only the sensors chosen in the design of the experiments.

We chose Intel Lab Data [45] as the dataset to support the experiments and train the LSTM model. This public dataset contains data from different sensor nodes, totaling 3 million records, and mapping the following sample types: temperature, humidity and luminosity. The data collected during approximately 35 days was located in the same building, and its internal placements can be seen in Fig. 12.

Data from 12 sensors (Intel Lab Data) supported the results presented in this section. Thus, the experiments with these sensors were submitted to the same scenarios (IoT data stream and network configuration) with variation in the installation mode of the OpenFlow rules, being: (i) reactive mode - default, (ii) installation of rules (30s, 60s, and 120s timeout) based on prediction of IoT data stream. The following metrics were collected: (i) RTT delay, analyzing the impact of IoT data stream on response time, (ii) mean absolute error (MAE), (iii) root mean squared error (RMSE). Such errors (MAE and MSE) consider the time difference between the installation of the rule and the arrival of the IoT message. The MAE and RMSE metrics calculation uses the global approach (considering hits and misses) and the hit approach (considering only hits). The global approach considers the RTT of all IoT messages, regardless of whether the prediction model and the OpenFlow rules scheduler installed the rules at the right time. The hit approach only considers the RTT of IoT messages that previously encountered all OpenFlow rules installed on the network path.

We ran the experiments using an Intel core i5-4210 Processor and 12 GB RAM with Ubuntu 14.04 LTS operating system. Three simulations were performed for each scenario in order to generate the results of the design of the experiments used in this work. Each simulation lasted 2200 s (approximately 37 min), long enough to provide variation in the workload (IoT messages by TATU protocol [26]) and provide different behaviors in the collected values from the temperature sample

Fig. 13 Results by RTT metrics and percentage of hits



(a) Results grouped by controller version. RTT metric data according to the timeout variation of the OpenFlow rules and the no prediction scenario.

(b) Percentage of hits of the proposed machine learning model according to the OpenFlow rules timeout variation. The results indicate the probability of an IoT message arriving at a switch and binding to the previously installed OpenFlow rule (by prediction).

(dataset). Such experiments consider two different versions of SDN Controllers - POX Controller[2] and Ryu Controller[3] (both with OpenFlow V1.0 protocol). Thus, it is shown that the proposed IoT data stream prediction and previous installation of OpenFlow rules can be applied to the appointed Controllers and with possibilities of extension to other versions. All values generated in this stage were used, since the results obtained in each scenario did not present significant changes. In the experiments, we used a hybrid (combining tree and linear topology) with standard mininet links and bandwidth ranging from 5 to 10 Mb/s.

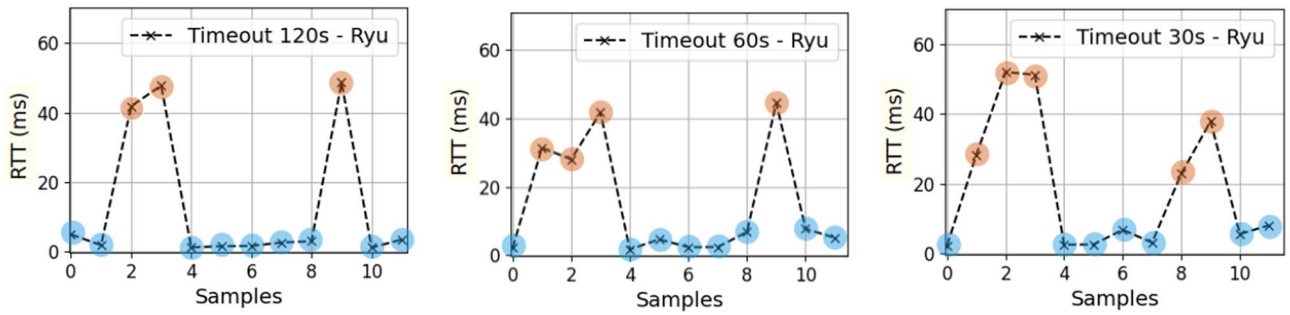
The results presented in Fig. 13a indicate that the proposal of prior installation by prediction allowed a reduction in IoT data stream RTT of about 65% compared to the traditional OpenFlow reactive mode (No Prediction). This characteristic demonstrates that the model can predict the adequate time for installing the rules and, consequently, decrease the response time (see Sect. 4). According to Fig. 13a, the values indicate the timeout configuration with 120 s obtains the best results (lowest RTT). Such aspect occurs because the greater the timeout value inserted in the OpenFlow rules, the greater the possibility that a given IoT message will find the corresponding rule already installed. Another important aspect shown in Fig. 13a is the variations in the results when modifying the Controller version. Although both have similar behavior to the results described above, it is noteworthy that Ryu Controller has lower RTT values when compared to the same configuration using the POX Controller. This factor indicates that Controllers implement OpenFlow policies and routing algorithms with different internal data structures and performance (although the similar behavior of the results). However, in experiments with both Controllers, the IoT data stream prediction strategy decreased the RTT value and consequently fulfilled the delivery of IoT messages in a shorter time.

Figure 13b shows the results of the percentage of hits obtained by the proposed machine learning model according to the variation in the OpenFlow rules timeout. In this case, are considered hits events that the IoT data stream occurs with the OpenFlow rules previously installed in the switches (by prediction). The results presented in Fig. 13b have a similar explanation to the data indicated in Fig. 13a, i.e., the longer the rule remains in the switches, the greater the probability that new IoT data stream will find its corresponding rule. Thus, the configuration with 120s of timeout had the highest hit rate (70.3%), followed by 68.4% and 64.7% for the 60 s and 30 s timeouts scenarios. Such hit values reinforce the reason for the decrease in RTT presented in this section. To highlight the hits and misses of the proposed model, a specific fragment of the results is presented in Fig. 14. This figure contains, for a specific sensor, the variation in hits and misses according to the OpenFlow timeout value.

Figure 15 shows the values obtained in the experiments for the MAE and RMSE metrics. In these cases, the data is characterized between calculations (MAE and RMSE) only for hits (i.e., when the IoT message matches the previously OpenFlow rules installed) and calculations considering the misses and hits (considering all timeouts - hits and misses).

[2] POX Controller: https://github.com/noxrepo/pox.

[3] Ryu Controller: https://ryu-sdn.org/.

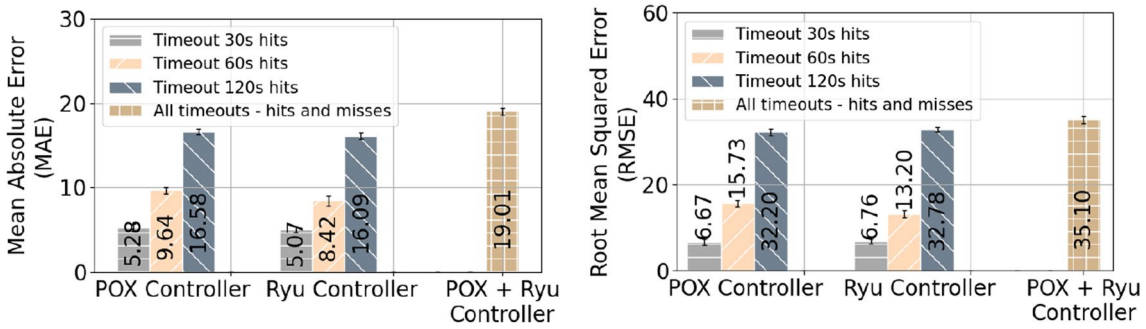


(a) Samples from sensor ID 2 (see Figure 12) with 120 seconds timeout.

(b) Samples from sensor ID 2 (see Figure 12) with 60 seconds timeout.

(c) Samples from sensor ID 2 (see Figure 12) with 30 seconds timeout.

Fig. 14 Impact of hit predictions on RTT metric results. The results show samples collected in three sensors included in the set of devices of the experiments



(a) MAE metric

(b) RMSE metric

Fig. 15 Analysis of MAE and RMSE metrics based on variations: Controller version, and OpenFlow rules timeout

This approach aims to perform calculations for the global scenario (misses and hits) and contexts of only hits, allowing behavior visualization of the MAE and RMSE metrics for both scenarios.

In Fig. 15a (MAE data), the results show that the MAE values (hits) become smaller according to the decrease in the timeout value. Taking into account the MAE and the hits of the proposed model (timeout hits in Fig. 15a), the timeout 30s presents the best values for the MAE metric, indicating greater accuracy of the model for this configuration. It is worth mentioning that considering the time that the rule persists in the switches (timeout) for both versions of the Controllers, the results remained adequate in the four possible scenarios shown, including the global scenario (hits and misses). In the global scenario, the value of 19.01 was obtained directly by averaging all previous configurations and considering errors and hits. The values shown in Fig. 15a (especially All timeouts - hits and misses) match with the percentage of hits shown in Fig. 13b. In this case, the MAE values do not exceed the average time that a given OpenFlow rule will persist in the switch, thus causing more hits than misses.

The results for the RMSE metric (see Fig. 15b) present a behavior similar to that obtained for the MAE metric. For hit scenarios, the results show that lower rule timeouts cause lower values for the RMSE. Furthermore, considering all scenarios, the highest RMSE was obtained by the global setting (All timeouts - hits and misses) with a value of 35.10. Such results corroborate the percentage of hits described in Fig. 13b, as the time that OpenFlow rules remain in the switches can be adjusted to values greater than the results obtained in the global RMSE - allowing a greater hits probability.

7 Conclusion and future works

In this paper, we present an approach to implement data stream prediction among IoT devices to install OpenFlow rules in an SDN network. Using such a solution, we reduced the response time (by RTT) in IoT data traffic in an SDN/OpenFlow network operating in reactive mode. The proposed machine learning algorithm employs an LSTM network that receives temperature data and concept drift time as inputs and outputs the time of the following IoT communication. The results presented indicated that the generated model produced adequate predictions for the evaluated IoT scenario considering the variation in timeout in the OpenFlow rules. Therefore, based on the results, it is considered that the approach proposed in this work led to an improvement in the performance of the experiments considering the evaluated RTT metric.

In future work, we intend to expand the scenario by increasing the number of sensors. This approach will enable us to evaluate the impact of the number of devices on the effectiveness of the solution and comprehensively assess the scalability aspects of the proposal. Additionally, we plan to test and adapt the proposed solution for other datasets and types of sensors applicable to the Internet of Things context. This approach will allow us to evaluate the proposal across a broader range of scenarios and situations within IoT environments.

The proposed model and experiments present some limitations we intend to evaluate in future work, such as training and evaluating the model in other datasets for communication prediction in the IoT data stream. In addition, the experiment with the LSTM model has the architecture for the univariate data stream; in future work, we intend to expand our prediction model to use the IoT data stream multivariate.

Author contributions E.B.: Conceptualization, methodology, implementation, testing, experiments, writing of the original draft, review. B.A.: Formal analysis, methodology, implementation, testing, experiments, writing of the original draft, review. E.S.: Implementation, testing, experiments, writing of the original draft. J.C.: Formal analysis, implementation. R. A. R.: Conceptualization, formal analysis, methodology, writing of the original draft, review. S.D.: Conceptualization, methodology, writing of the original draft, review. G.F.: Conceptualization, methodology, writing of the original draft, review. C.P.: Conceptualization, methodology, writing of the original draft, review, supervision.

Data availability The data that support the findings of this study are available from the corresponding author upon reasonable request.

Declarations

Competing interests The authors declare no competing interests.

References

1. Atzori L, Iera A, Morabito G. The internet of things: a survey. Comput Netw. 2010;54(15):2787–805.
2. Whitmore A, Agarwal A, Xu L. The internet of things-a survey of topics and trends. Inf Syst Front. 2015;17(2):261–74.
3. Sundmaeker H, Guillemin P, Friess P, Woelfflé S. Vision and challenges for realising the internet of things. EU Publications; 2010.
4. Yaqoob I, Ahmed E, Hashem IAT, Ahmed AIA, Gani A, Imran M, Guizani M. Internet of things architecture: recent advances, taxonomy, requirements, and open challenges. IEEE Wirel Commun. 2017;24(3):10–6. https://doi.org/10.1109/MWC.2017.1600421.
5. Delicato FC, Pires PF, Batista T, Cavalcante E, Costa B, Barros T. Towards an iot ecosystem. In: Proceedings of the First International Workshop on Software Engineering for Systems-of-Systems. SESoS '13, 2013; pp. 25–28. ACM, New York, NY, USA.
6. Bonomi F, Milito R, Zhu J, Addepalli S. Fog computing and its role in the internet of things. In: Proceedings of the First Workshop on Mobile Cloud Computing, 2012; pp. 13–16. ACM. https://doi.org/10.1145/2342509.2342513.
7. Chen K, Wang Y, Fei Z, Wang X. Power limited ultra-reliable and low-latency communication in uav-enabled iot networks. In: 2020 IEEE Wireless Communications and Networking Conference (WCNC), 2020; pp. 1–6. https://doi.org/10.1109/WCNC45663.2020.9120565.
8. Li Y, Zhang Y, Liu Y, Meng Q, Tian F. Fog node selection for low latency communication and anomaly detection in fog networks. In: 2019 International Conference on communications, information system and computer engineering (CISCE), 2019; pp. 276–279. https://doi.org/10.1109/CISCE.2019.00069.

9. Chour H, Kouicem DE, Fotouhi A, Mabrouk MB. Toward an autonomous smart home: a three-layer edge-fog-cloud architecture with latency analysis. In: 2021 IEEE 22nd International Conference on high performance switching and routing (HPSR), 2021; pp. 1–7. https://doi.org/10.1109/HPSR52026.2021.9481821.
10. Ghosh S, Das J, Ghosh SK, Buyya R. Clawer: context-aware cloud-fog based workflow management framework for health emergency services. In: 2020 20th IEEE/ACM International Symposium on cluster, cloud and internet computing (CCGRID), 2020; pp. 810–817. https://doi.org/10.1109/CCGrid49817.2020.000-5.
11. Kreutz D, Ramos FMV, Veríssimo PE, Rothenberg CE, Azodolmolky S, Uhlig S. Software-defined networking: a comprehensive survey. Proc IEEE. 2015;103(1):14–76. https://doi.org/10.1109/JPROC.2014.2371999.
12. OpenFlow: OpenFlow Product Certification. https://opennetworking.org/product-certification/. 2021. Online. Accessed 8 June 2021.
13. Nishtha, Sood M. Software defined network architectures. In: International Conference on parallel, distributed and grid computing, 2014; pp. 451–456. https://doi.org/10.1109/PDGC.2014.7030788.
14. Awan II, Shah N, Imran M, Shoaib M, Saeed N. An improved mechanism for flow rule installation in-band sdn. J Syst Arch. 2019;96:1–19. https://doi.org/10.1016/j.sysarc.2019.01.016.
15. Sanabria-Russo L, Alonso-Zarate J, Verikoukis C. Sdn-based pro-active flow installation mechanism for delay reduction in iot. In: 2018 IEEE Global Communications Conference (GLOBECOM), 2018; pp. 1–6. https://doi.org/10.1109/GLOCOM.2018.8647382.
16. Cai Y-Z, Wang Y-T, Tsai M-H. Dynamic adjustment for proactive flow installation mechanism in sdn-based iot. Comput Netw. 2021;194: 108167. https://doi.org/10.1016/j.comnet.2021.108167.
17. Isravel DP, Silas S, Rajsingh EB. Long-term traffic flow prediction using multivariate ssa forecasting in sdn based networks. Pervasive Mob Comput. 2022. https://doi.org/10.1016/j.pmcj.2022.101590.
18. Zafar S, Zafar B, Hu X, Zaydi NH, Ibrar M, Erbad A. Pbclr: prediction-based control-plane load reduction in a software-defined iot network. Internet of Things. 2023;24: 100934. https://doi.org/10.1016/j.iot.2023.100934.
19. Ghanbari H, Khayyambashi MR, Movahedinia N. Improving fog computing scalability in software defined network using critical requests prediction in iot. In: 2021 12th International Conference on information and knowledge technology (IKT), 2021; pp. 6–10. https://doi.org/10.1109/IKT54664.2021.9685070. IEEE.
20. Volkov A, Proshutinskiy K, Adam AB, Ateya AA, Muthanna A, Koucheryavy A. Sdn load prediction algorithm based on artificial intelligence. In: International Conference on distributed computer and communication networks, 2019; pp. 27–40. Springer.
21. Youssef S, Rysavy O. Toward migration to sdn: generating sdn forwarding rules by decision tree. In: 2023 26th Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN), 2023; pp. 16–20. https://doi.org/10.1109/ICIN56760.2023.10073500.
22. Kumar KP, Sivanesan P, Sathyaprakash P. Qos-enhancement adaptive openflow rule integration in software defined wsn for iot applications. In: 2022 International Conference on augmented intelligence and sustainable systems (ICAISS), 2022; pp. 1466–1472. https://doi.org/10.1109/ICAISS55157.2022.10011102.
23. Isyaku B, Kamat MB, Abu Bakar Kb, Mohd Zahid MS, Ghaleb FA. Ihta: dynamic idle-hard timeout allocation algorithm based openflow switch. In: 2020 IEEE 10th Symposium on Computer Applications and Industrial Electronics (ISCAIE), 2020; pp. 170–175. https://doi.org/10.1109/ISCAIE47305.2020.9108803
24. Kuźniar M, Perešíni P, Kostić D, Canini M. Methodology, measurement and analysis of flow table update characteristics in hardware openflow switches. Comput Netw. 2018;136:22–36.
25. Read J, Rios RA, Nogueira T, Mello RFd. Data streams are time series: challenging assumptions. In: Brazilian Conference on Intelligent Systems, 2020; pp. 529–543. Springer.
26. Batista E, Andrade L, Dias R, Andrade A, Figueiredo G, Prazeres C. Characterization and modeling of iot data traffic in the fog of things paradigm. In: 2018 IEEE 17th International Symposium on Network Computing and Applications (NCA), 2018; pp. 1–8. https://doi.org/10.1109/NCA.2018.8548340.
27. Gama J, Žliobaitė I, Bifet A, Pechenizkiy M, Bouchachia A. A survey on concept drift adaptation. ACM Comput Surv. 2014;46(4):44.
28. Lu J, Liu A, Dong F, Gu F, Gama J, Zhang G. Learning under concept drift: a review. IEEE Trans Knowl Data Eng. 2018. https://doi.org/10.1109/TKDE.2018.2876857.
29. Page ES. Continuous inspection schemes. Biometrika. 1954;41(1/2):100–15.
30. Mouss H, Mouss D, Mouss N, Sefouhi L. Test of page-Hinckley, an approach for fault detection in an agro-alimentary production system. In: 2004 5th Asian Control Conference (IEEE Cat. No.04EX904), 2004; vol. 2, pp. 815–8182.
31. Hochreiter S, Schmidhuber J. Long short-term memory. Neural Comput. 1997;9(8):1735–80.
32. Mininet: Mininet Network Emulator. http://mininet.org/overview/. 2022. Online. Accessed 18 Jan 2022.
33. Alencar BM, Rios RA, Santana C, Prazeres C. Fot-stream: a fog platform for data stream analytics in iot. Comput Commun. 2020;164:77–87.
34. Jadhav A. Clustering based data preprocessing technique to deal with imbalanced dataset problem in classification task. In: 2018 IEEE Punecon, 2018; pp. 1–7. https://doi.org/10.1109/PUNECON.2018.8745437.
35. Santos FJJD, Camargo HDA. Preprocessing in fuzzy time series to improve the forecasting accuracy. In: 2013 12th International Conference on machine learning and applications, 2013; vol. 2, pp. 170–173. https://doi.org/10.1109/ICMLA.2013.185.
36. Sapankevych NI, Sankar R. Time series prediction using support vector machines: a survey. IEEE Comput Intell Mag. 2009;4(2):24–38. https://doi.org/10.1109/MCI.2009.932254.
37. Khedka SP, Canessane RA, Najafi ML. Prediction of traffic generated by iot devices using statistical learning time series algorithms. Wirel Commun Mob Comput. 2021;2021:1–12.
38. Zhang J, Zhu Y, Zhang X, Ye M, Yang J. Developing a long short-term memory (lstm) based model for predicting water table depth in agricultural areas. J Hydrol. 2018;561:918–29.
39. Kong F, Li J, Lv Z. Construction of intelligent traffic information recommendation system based on long short-term memory. J Comput Sci. 2018;26:78–86.
40. Cheng Y, Xu C, Mashima D, Thing VL, Wu Y. Powerlstm: power demand forecasting using long short-term memory neural network. In: Advanced Data Mining and Applications: 13th International Conference, ADMA 2017, Singapore, November 5–6, 2017, Proceedings 13, 2017; pp. 727–740. Springer.

41. Saleem TJ, Chishti MA. Deep learning for internet of things data analytics. Proc Comput Sci. 2019;163:381–90.
42. Alencar B, Canário JP, Neto RL, Prazeres C, Bifet A, Rios R. Fog-deepstream: a new approach combining lstm and concept drift for data stream analytics on fog computing. Internet Things. 2023;22:1–18.
43. Hasan T, Adnan A, Giannetsos T, Malik J. Orchestrating sdn control plane towards enhanced iot security. In: 6th IEEE Conference on Network Softwarization (NetSoft), 2020; pp. 457–464. IEEE.
44. Li Y, Jin D, Wang B, Su X, Riekki J, Sun C, Wei H, Wang H, Han L. Predicting internet of things data traffic through lstm and autoregressive spectrum analysis. In: NOMS 2020 IEEE/IFIP Network Operations and Management Symposium, 2020; pp. 1–8. IEEE.
45. Madden S. Intel Lab Data. http://db.csail.mit.edu/labdata/labdata.html. 2004. Online. Accessed 10 Sep 2021.