Deep-Learning-Based SDN Model for Internet of Things: An Incremental Tensor Train Approach

Amritpal Singh, Gagangeet Singh Aujla, *Senior Member, IEEE*, Sahil Garg, *Member, IEEE*, Georges Kaddoum, *Member, IEEE*, and Gurpreet Singh

***Abstract*—The Internet of Things (IoT) has emerged as a revolution for the design of smart applications like intelligent transportation systems, smart grid, healthcare 4.0, Industry 4.0, and many more. These smart applications are dependent on the faster delivery of data which can be used to extract their inherent patterns for further decision making. However, the enormous data generated by IoT devices are sufficient to choke the entire underlying network infrastructure. Most of the data attributes present little or no relevance to the prospective relationships and associations with the projected benefits foreseen. Therefore, order-based generalization mechanisms, known as tensors, can be used to represent these multidimensional data, thereby minimizing the flow table (FT) lookup time and reducing the storage occupancy. So, a novel IoT-train-deep approach for intelligent software-defined networking is designed in this article. The proposed approach works in four phases: 1) tensor representation; 2) deep Boltzmann machine-based classification; 3) subtensor-based flow matching process; and 4) incremental tensor train network for FT synchronization. The proposed model has been extensively tested, and it illustrates significant improvements with respect to delay, throughput, storage space, and accuracy.**

***Index Terms*—Deep Boltzmann machine, network intelligence, software defined networking (SDN), tensor train decomposition.**

I. Introduction

SMART cities generate huge amounts of data from numerous connected devices (sensors, vehicles, actuators, etc) and applications (traffic management, healthcare system, stock exchange data, e-shopping, etc.) which are transmitted to geolocated sites for processing or storage [1]. For example, in the healthcare sector, a large amount of medical images and data are generated and stored for use in online medical health records. Similarly, at the consumer end, the content (images, videos, etc.) is generated via social networks for creating a highly connected society. Another example is that of smart meters, deployed at each household, capture the energy usage data continuously thereby generating a multithousand fold data. These kinds of multihundred or multithousand petabyte data sets are creating a new horizon of opportunities in diverse application domains like human genomics, healthcare systems, finance and banking, oil and natural gas exploration, and many more. These data being huge in amount (volume), variable in type and structure (velocity), and generated at different rates or times (variety) have to be analyzed using sophisticated algorithms or big data solutions for identifying hidden patterns which can further lead to several benefits related to decision making in smart ecosystems [2]. In simple words, big data technologies and architectures are designed to extract economical value from a large amount of variable data through high-velocity capture, discover, or analyze process [3].

The evolution of the Internet of Things (IoT) has converted the entire technological paradigm into a data-driven plethora of smart and connected ecosystems. The data generated from this revolutionary ecosystem needs to be transmitted at a high data rate and thereafter processed or analyzed in real time at a remote cloud data center or at the edge of the network using resource-constrained devices. The success of big data solutions completely relies on the intelligence of the underlying architecture driving the network infrastructure. For quite some time, the data centers have been the hub of providing big data solutions in the IoT environment using the cloud computing platform. However, as the need for processing data closer to the IoT devices or data sets arises, the big data solutions should be provisioned at the network edge using the fog or edge computing technology [4]. For this reason, a reevaluation of the existing network infrastructure (specifically for IoT ecosystem) is essential, keeping in view the following prerequisites: 1) the capability to handle mixed data sets (structured and unstructured) generated from multiple heterogeneous sources and 2) the content generated can be of unpredictable nature with no standard schema and structure.

A. Motivation

A flexible network architecture that could integrate multidimensional and multifunctional big data generated by IoT devices at a varied scale at all times is the most essential requirement. In a typical network layout, the expected

Manuscript received September 12, 2019; revised October 30, 2019; accepted November 6, 2019. Date of publication November 14, 2019; date of current version July 10, 2020. This work was supported by the Tier 2 Canada Research Chair on the Next Generations of Wireless IoT Networks. *(Corresponding author: Sahil Garg.)*

A. Singh and G. Singh are with the Computer Science and Engineering Department, Chandigarh University, Mohali 147001, India (e-mail: amritpal_bcet@yahoo.co.in; ait.cse.gurpreet@gmail.com).

G. S. Aujla is with the School of Computing, Newcastle University, Newcastle Upon Tyne NE1 7RU, U.K., and also with the Computer Science and Engineering Department, Chandigarh University, Mohali 147001, India (e-mail: gagi_aujla82@yahoo.com).

S. Garg and G. Kaddoum are with the Electrical Engineering Department, École de Technologie Supérieure, Université du Québec, Montreal, QC H3C 1K3, Canada (e-mail: sahil.garg@ieee.org; georges.kaddoum@etsmtl.ca).

Digital Object Identifier 10.1109/JIOT.2019.2953537

services linked to the incoming traffic are not always the same. Therefore, routing the incoming traffic generated by IoT devices using a standard and fixed approach is not effective. The traffic flow can be classified on the basis of payload, port number, priority, and many more characteristics. For example, video applications, VOIP services, and chat require lower delay while, in contrast, multimedia, Web or interactive applications require a higher bandwidth for a reliable flow [13]. However, in a conventional network, such an adaptive or autonomous network management is not possible. Therefore, software-defined networking (SDN) is one of the popular choice for the network administrator as it pushes the intelligence into the central controller which in turn provide the network control in a very dynamic and agile manner.

In SDN architecture, the flow table (FT) is one of the most important entities where the incoming traffic packet header values are matched to find a suitable path [14]. But, in some cases, the exact match is not available in the FTs and the control is passed to the SDN controller. The controller inspects the packet and rebuilds a new flow entry and adds it to the FTs of the concerned switches. In this way, the size of the FT keeps on increasing and thereby the lookup time for matching a flow entry also increases. At some specific point of time, the storage space of FTs crosses the threshold limit of storage and in such cases, the request is forwarded to the controller. This increases the generation control traffic in the network thereby creating congestion. Although this process of FT management is quite effective, however, there are still unwanted elements in the traffic packets which are not used for any processing or forwarding. These unwanted elements not only consume additional memory but also hinder the overall performance [9], [15]. One of the possible solution for the above issue is to convert the data packets and the FTs into tensorized format. This can help to minimize the complexity of incoming traffic and reduce the size of the FTs thereby minimizing the lookup and flow matching time which in turn ends up in the reduced delay.

B. Contributions of the Article

Keeping in view the above challenges, a deep learning-based intelligent SDN model is designed for the IoT environment, which works in the following phases.

1) In the first phase, all the unwanted fields are removed from the incoming packets using the tensorization approach to reduce them to a lower rank tensors.
2) In the second phase, the incoming traffic (which is in tensor form) is classified into different tensor classes using the deep Boltzmann machine-based approach. The behavior of the traffic packets can be extracted using deep learning and thereafter classified.
3) In the third phase, the entries in the FT are represented in different subtables (in line with the classes generated by deep learning approach) using tensor representation called subtensor tables. These tensor represented FTs help to utilize the restricted storage space. Here, the tensorized data packages are matched with the associated subtensor FT which reduces the lookup time.
4) However, in some cases, the exact match is not available and the controller rebuilds the flow entry and installs in the concerned subtensor FT. In this phase, in order to reduce redundancy in FT entries, the incremental tensor-train decomposition approach is used to synchronize the subtensor FTs.

C. Organization

The rest of this article is organized as follows. Section II provides the related work. The system model is presented in Section III. In Section IV, the proposed scheme is presented. The proposed scheme is evaluated in Section V followed by the concluding remarks in Section VI.

II. Related Work

The existing state-of-the-art proposals have utilized various approaches to handle the incoming data traffic in a dynamic manner. For example, Aujla *et al.* [5] proposed a decision tree-based classification approach which adopts queuing theory to reduce the waiting time of data traffic and schedule them in a dynamic manner in SDN. Tajiki *et al.* [6] proposed a resource allocation architecture for the SDN environment based on the energy-aware service function chaining. The authors considered the problem of the virtual network functions in the form of delay, server utilization, and link utilization and designed a heuristic function for optimal solution. However, the above discussed proposals have not addressed the limited rule-space issue of switches in the SDN architecture. To overcome these challenges, Chuang *et al.* [7] proposed a routing and forwarding algorithm to handle the scalability problem in SDN. The authors classified the incoming traffic and optimized the FT entry storage space to resolve the limited space capacity problem of the ternary content addressable memory. However, the above proposal handles the entire data traffic in a similar manner regardless of the traffic characteristics.

To resolve the above issues, Hayes *et al.* [8] designed a full traffic classification technique which used the system-based approach to improve the performance and enhance the scalability of the SDN-based network. Similarly, Moa *et al.* [9] proposed a smart packet routing technique using tensor-based deep belief architecture for intelligent flow of the packets in the network. The proposed scheme categorized the parameters of the incoming traffic using the deep learning approach which was trained by assigning weights and biases according to traffic characteristics. The authors highlighted that the tensor network can help to remove the unwanted dimensions of the data traffic which can provide multifold benefits. In this direction, Kaur *et al.* [10] introduced a tensor-based big data management scheme for reduction in the size of parameters of incoming traffic in the elephant traffic flows. The authors applied Frobenius norms to reduce the reconstruction error of high dimensional tensors. Even more, they designed an empirical probability-based control approach using SDN for selection of best path for data forwarding. However, the above proposal has not used complete SDN architecture but used SDN and non-SDN switches to provide dynamic flow forwarding.

TABLE I
SUMMARIZATION OF DIFFERENT TECHNIQUES/TECHNOLOGIES FOR EFFICIENT FLOW FORWARDING

Author	Technique used	Tensor	Flow entries	SDN	Incremental Tensor Train	Deep Learning	IoT
Aujla *et al.* [5]	Adaptive Flow Forwarding	Yes	Heterogeneous	✓	No	×	×
Tajiki *et al.* [6]	Heuristic function based approach	No	Heterogeneous	✓	No	×	×
Chuang *et al.* [7]	Routing and forwarding approach	No	Heterogeneous	✓	No	×	×
Hayes *et al.* [8]	System based approach	No	Heterogeneous	✓	No	×	×
Moa *et al.* [9]	Tensor-based Deep Belief Architecture	Yes	Homogeneous	×	No	✓	✓
Kaur *et al.* [10]	Tensor-based big data management	Yes	Heterogeneous	✓	No	×	×
Kuang *et al.* [11]	Forwarding tensor model	Yes	Heterogeneous	✓	No	×	×
Maity *et al.* [12]	Tensor-based Rule-space Management	Yes	Heterogeneous	✓	No	×	×
Proposed Scheme	Tensor-based Rule-space Management	Yes	Heterogeneous	✓	Yes	✓	✓

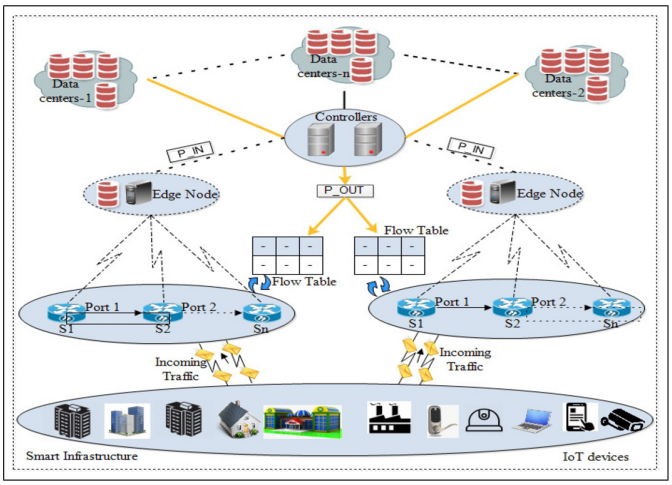

Fig. 1. IoT-edge-cloud network.

Kuang *et al.* [11] proposed a tensor-based model to sustain the quality-of-service in the SDN environment. The designed technique identifies the best suited path for packet transmission at the control plane. The authors introduced a transition tensor model for effectively prediction of incoming traffic to improve the service quality at the control plane. Similarly, Maity *et al.* [12] proposed a tensor-based rule space management system which optimizes the limited space of the switches. The authors used the tensor decomposition concept to control the heterogeneous entries in FT of SDN-based switches. They also incorporated the caching mechanism to increase the throughput of the network. Table I summarizes the existing proposals related to the addressed problem and compares them with the proposed scheme.

III. SYSTEM MODEL FOR IOT-EDGE-CLOUD NETWORK

Fig. 1 shows the SDN architecture in multi IoT-edge-cloud environments which consists of many horizontal as well as vertical communication patterns: 1) server-to-server; 2) server-to-storage; 3) IoT-to-edge; 4) edge-to-cloud; and 5) IoT-to-SDN control plane. The data center traffic flow can be managed more efficiently using SDN and network function virtualization compared to the present day situation. The system model is divided into multilayered architecture which is discussed as follows.

1) *IoT Plane:* This plane consists of multiple IoT devices deployed at different smart communities like smart homes, smart factories, and healthcare environment.
2) *Data Plane:* All the OpenFlow forwarding devices (routers and switches) are deployed at this layer for transmitting the data packets received from IoT devices.
3) *Control Plane:* The logically centralized SDN controller is located at this plane. This controller builds the flow rules at each FT installed in the forwarding devices.
4) *Edge Plane:* This plane comprises of edge servers or nodes which are deployed geo-distributively to process or analyze data generated by IoT devices.
5) *Cloud Plane:* Multiple geodistributed cloud data centers managed by different cloud services providers are deployed to handle the high computing workload.

IV. INTELLIGENT IOT-TRAIN-DEEP APPROACH

In this section, an intelligent network management approach is designed for the IoT environment. Fig. 2 depicts the different phases of the proposed IoT-Train-Deep approach in a software-defined IoT-edge-cloud environment. This overall scheme embeds network intelligence which helps to convert the traffic packets into tensors, then efficiently classifies them, matches these packets in subtensor tables synchronized using the incremental tensor train approach. The detailed explanation of the different phases of the proposed approach is provided in the subsequent sections.

A. Phase I: IoT-Tensor Network Formation

In this section, the tensor representation of data generated by IoT devices is explained. This representation helps remove unwanted dimensions of data which in turn leads to enhanced performance of the underlying network infrastructure. Here, the matrices are represented by M, vectors are represented by v, tensor is represented by X, rank of the tensor is mentioned by r, the dimensions of the matrix are $(I_1, I_2, I_3, \ldots, I_N)$. To represent all the elements of a tensor in a given order, a colon (:) is used as $(X_1(I_1, :, :), X_2(:, I_2, :), \ldots, X_N(:, :, I_N))$ [16]. The comprehensive elaboration of tensors, the related operation, and tensor decomposition techniques are presented.

Tensor Definition: A tensor is an order-based generalization mechanism for the representation of multidimensional array. In a tensor network, the rank defines the number of indices/dimensions used collectively in the network [10]. So, zero-rank defines scalar data (s), first-rank represents the vector (v_i) and second-rank represents the matrix ($M_{i,j}$). A tensor is denoted as

$$X \sum \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}. \quad (1)$$

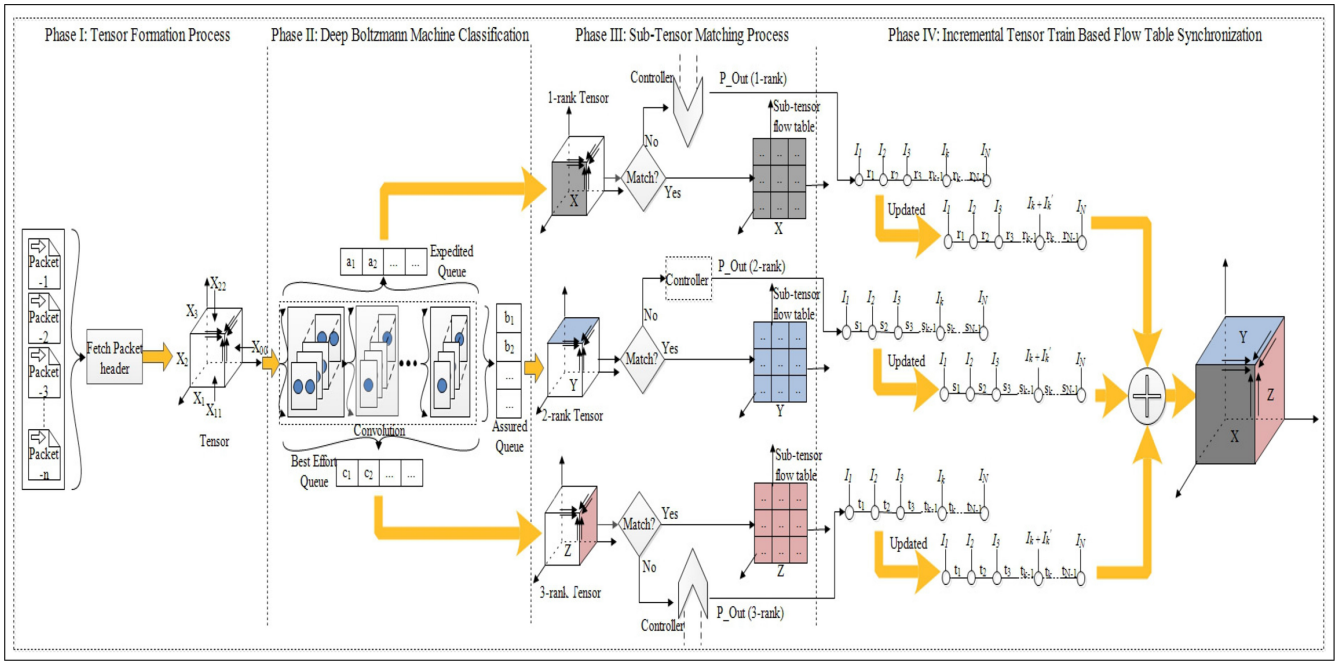

Fig. 2. Phases of proposed scheme.

Tensor Contractions: In tensor contractions, the indices of an array are reduced by pairing the dimensional vectors. It is expressed as the sum of products of scalar components of the tensor caused due to the application of summation convention to the dummy indices which are bound to each other in an expression. In each vector ν, there is a dual vector space in which different linear functions (f) are defined. Precisely, if the column vectors are represented by ν, then the row vectors are represented as f and they are depicted by a dot product. In the dot product, the row vector is positioned at the left and the column vector is positioned at the right of the matrix. The contraction in ν is represented as

$$\tilde{f}\left(\vec{\nu}\right) = f_i \nu^i. \tag{2}$$

The above representation can be further expanded as

$$f_i \nu^i = f_1 \nu^1 + f_2 \nu^2 + \cdots + f_N \nu^N \tag{3}$$

where, ν^i are the components of ν. By using the contraction, a new matrix is produced with two indices.

Scalar Product: The scalar product of two vectors ν_1 and ν_2 is represented by the vector ν with zero-rank as

$$\nu = \sum_{s=1}^{D} \nu_{1(s)} . \nu_{2(s)} \tag{4}$$

where, D denotes the different component values.

In a tensor network, the fourth rank tensor is defined as the dot product of four different tensors X^1, X^2, X^3, and X^4. Here, the indices are given as I_1, I_2, I_3, I_4, I_5, I_6, and I_7. The tensor with rank 4 is given as

$$f = \sum_{I_1, I_2, \ldots, I_6, I_7}^{D} X^1_{I_1, I_2, I_3, I_4} . X^2_{I_2, I_3, I_4, I_5} . X^3_{I_3, I_4, I_5, I_6} . X^4_{I_4, I_5, I_6, I_7}. \tag{5}$$

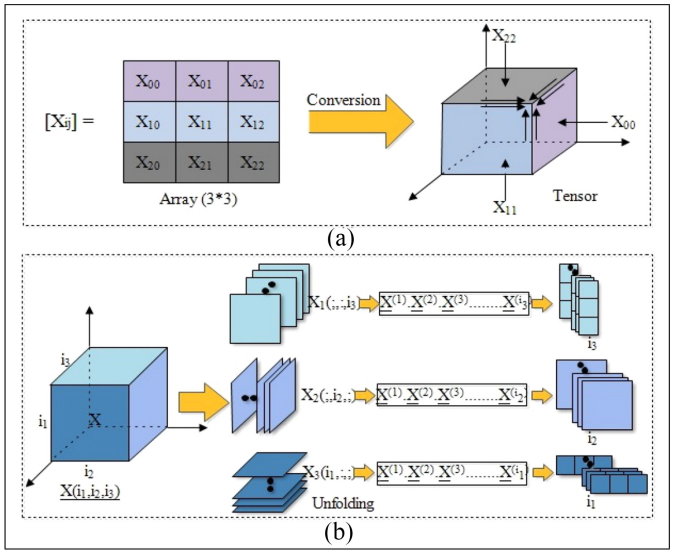

Fig. 3. Tensor formation and unfolding process.

A tensor is represented by shapes and indices and the tensor network is defined by the lines connecting different shapes. The connected lines are referred to as the contracted indices and the lines which are not connected between shapes are called open indices. The connected lines are the repeated indices in the network that help to reduce the components in the network. The scalar data (s) is represented as ($s \sum \mathbb{R}^{r \to 0}$), vector ($\nu$) is represented as ($\nu \sum \mathbb{R}^i$), matrix ($M$) is represented as ($M \sum \mathbb{R}^{i \times j}$), tensor ($X$) is represented as ($X \sum \mathbb{X}^{r \times \mathbb{I} \times r}$), the matrix product (f) is represented as ($M \sum \mathbb{R}^{i \times j}$), the vector product (g) is represented as ($\nu \sum \mathbb{R}^i$), and the matrix-to-matrix product is depicted as ($M_1 \sum \mathbb{R}^{i \times r} . M_2 \sum \mathbb{R}^{r \times j}$) [17].

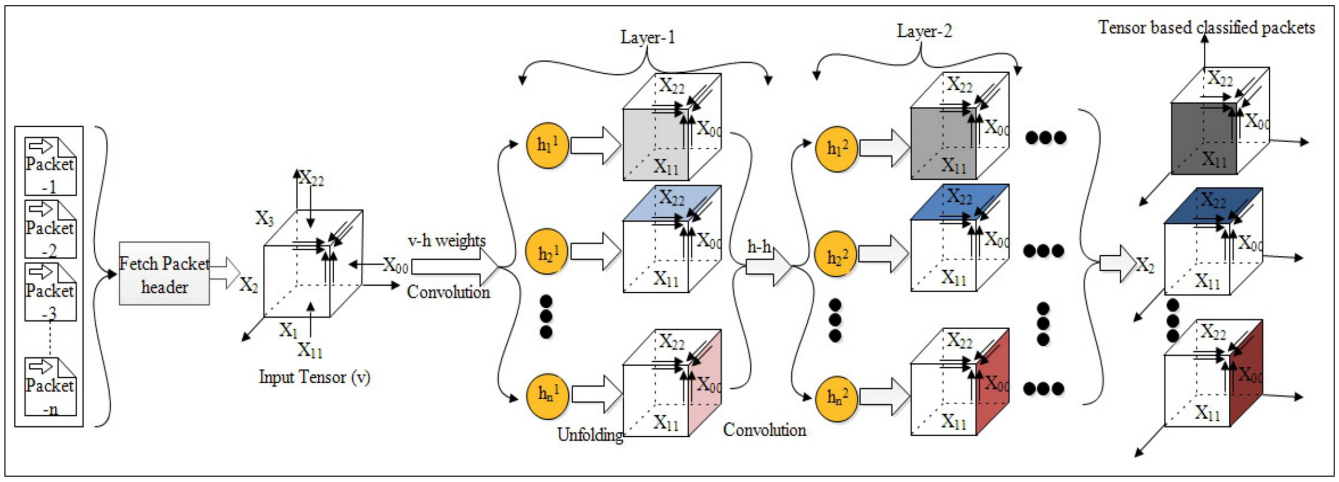

Fig. 4. Tensor formation and traffic classification using deep Boltzmann machine scheme.

Using the above concept, the heterogeneous data packets are forwarded to the tensor network, to remove the unwanted information from the packets. Only the required parameters in the header of the packet is stored in the tensor network. Initially, the header of the incoming traffic is fetched to understand the behavior of the traffic. Afterward, the incoming traffic packets are converted into tensor format as shown in Fig. 3(a). If a multivariant array considered as header of the packet, stores the meaningful information of the packet, then $f(x_1, x_2, \ldots, x_N)$ is an I variant function of array. The unfolding of X [as visible in Fig. 3(b)] is shown as

$$X(I_1, I_2, \ldots, I_n) = f(x_1(I_1), \ldots, x_n(I_n)). \tag{6}$$

B. Phase II: Deep Boltzmann Machine-Based Classification

In this phase, the tensorized data traffic is classified into multiple queues using the deep Boltzmann machine-based approach. The deep Boltzmann machine works on the concept of stochastic values. There are three layers in the deep Boltzmann machine approach, i.e., visible/input (v) layer, hidden layer (h) and output layer. In the hidden layers, random weights are assigned to the training data. Thereafter, the weights are assigned for the input according to their behavior. The weights in the visible layer are $v \in \{0, 1\}^D$, where D is the number of units in the visible layer. In case of the hidden layer, the weights are $h \in \{0, 1\}^{F_1}$, where F_1 are the units in the hidden layer and so on. After this, the $v - h$ weights are mapped with each other and the output of this process becomes the input for the next hidden layer and the same process is repeated till the required output is reached [18]. Using this approach, the incoming data packets ($\mathbb{D}_{TR}$) which have been converted into tensor format are classified into multiple queues on the basis of data characteristics or desired quality parameters. The different steps involved in the proposed classification scheme are listed as follows.

1) In the first step, the weights of low-rank tensors are mapped with the hidden layer units weights. The process is known as convolution of the tensor.
2) Then, the mapped units are unfolded into a low-rank tensor.
3) The resultant low-rank tensor undergoes a convolution process wherein the inference of the sample is improved. To verify the inference level, the following equation is used to check energy (E) of the previous and current stages:

$$\mathbf{E}(\mathbf{v}, \mathbf{h}, \theta) = -\sum_{\mathbf{i}=\mathbf{1}}^{\mathbf{D}} \sum_{\mathbf{j}=\mathbf{1}}^{\mathbf{F}} \mathbf{W_{i,j} v_i h_j} - \sum_{\mathbf{i}=\mathbf{1}}^{\mathbf{D}} \mathbf{a_i v_i} - \sum_{\mathbf{j}=\mathbf{1}}^{\mathbf{F}} \mathbf{b_j h_j}. \tag{7}$$

   This helps to extract the information about the incoming traffic for classification.
4) The classified tensors are categorized into three queues, as explained below.
   a) Most prioritized traffic is stored in an expedited queue (X_{EX}) for further processing.
   b) The traffic requiring guaranteed processing is forwarded toward an assured queue (X_{GR}).
   c) Rest is added to the best effort queue (X_{BE}).
5) Fig. 4 shows the three-mode tensor representation for the deep Boltzmann machine-based tensor formation and classification from input to output layers. The summarization of this entire process is provided in Algorithm 1.

C. Phase III: Rule-Space Management Using SubTensor Tables

In this phase, the FTs installed at the OpenFlow stitches are converted into tensorized form. This is done as there are various attributed in the FTs which are not at all used during the entire flow matching and data forwarding process. Such unwanted attributes or entries in the FT are ignored to reduce the storage space. Another new task performed in this phase is to divide the entire FT into subtensor FTs on the basis of different priority levels of the flow entries available. FTs at each switch in the network are converted into subtensor FTs (X_θ, $\theta ==$ EX, GR, BE) as follows.

1) Flow entries stored in network X are converted into expedited subtensor FT (ST_{FT}^{EX}) for faster traffic forwarding.

Algorithm 1 Classification Algorithm

Input: $\mathbb{D}_{TR}$, Data from network
Output: X_{EX},X_{GR},X_{BE} (Subtensor classification)

Convert $\mathbb{D}_{TR}$ to X tensor using Eq. 6
Forward the X input to DBM
Unfold the tensor X using

$$X(i_1, i_2, ..., i_n) = X_1(i_1, :, :)X_2(:, i_2, :)...X_n(:, :, i_n)$$

Initialize θ^0 for each layer in *DBM*
for h=1 : H **do**
  for α Iterations **do**
    for i=1 : F **do**
      if h==H **then**
        $\mathbf{v_i^h} = \sigma\left(\sum_{\mathbf{i=1}}^{\mathbf{F}} \mathbf{R_{ih}v_i}\right)$
      else
        $\mathbf{v_j^h} = \sigma\left(\sum_{\mathbf{j=1}}^{\mathbf{F}} \mathbf{2R_{jh}v_j}\right)$
      end if
    end for
    Iterate α till $\alpha \rightarrow \mu$ (Desired result)
  end for
  Store $\alpha\rightarrow$ (X_{EX} or X_{GR} or X_{BE})
end for
Add X_{EX}, X_{GR}, and X_{BE} into respective queues

Algorithm 2 Flow Matching and Synchronization Algorithm

Input: X_θ, θ == EX, GR, BE
Output: Flow path, Unified tensor Z

if $\theta == EX$ **then**
  Match $X_{EX} \rightarrow ST_{FT}^{EX}$
else if $\theta == GR$ **then**
  Match $X_{GR} \rightarrow ST_{FT}^{GR}$
else if $\theta == BE$ **then**
  Match $X_{BE} \rightarrow ST_{FT}^{BE}$
else
  MISMATCH
  Function *Packet_In*(X_θ) **do**
  Send $X_\theta \rightarrow OF$ controller
  OF controller $\rightarrow$ PACKET_OUT
  Return FE_{FT}^{θ}
  for $(n = 1; n \geq 0; n++)$ **do**
    Synchronize FT
    $ST_{FT}^{\theta} = ST_{FT}^{\theta(1)} \otimes ST_{FT}^{\theta(2)} \otimes ST_{FT}^{\theta(3)} \otimes \cdots \otimes ST_{FT}^{\theta(N)}$
    End Function
  end for
end if

2) Flow entries stored in network Y are converted into assured subtensor FT (ST_{FT}^{GR}) for guaranteed flow forwarding.
3) The remaining flow entries are converted into best effort subtensor FT (ST_{FT}^{BE}).

Now, the tensor data packets are matched in a correlated subtensor FT which consists of similar flow path characteristics. For example, if a video application data packet requires a flow path, then it is matched with a subtensor FT which contains flow entries which support lower delay paths only. This approach reduces the lookup time required to find the matching entry in a small and less complex subtensor FT. If the requested information is not found in the subtensor FTs, then the controller is called to rebuilds new flow entries specific to the incoming packets (Packet_In). The controller generates Packet_Out messages to update FT entries. Upon rebuilding the specific entry, the controller forwards the Packet_Out message to the concerned subtensor FT only. Thus, this approach utilizes the storage space of FT effectively and avoids unnecessary flow entries in FTs, thereby reducing the congestion.

Whenever there is an update in the subtensor FT entries, the concerned subtensor is converted into a Tensor_Train format. After this, the incremental tensor train decomposition approach is used to synchronize the reduced subtensor FTs into a unified tensor FT to maintain consistency and reduce redundancy. Here, the padding process is executed to synchronize the dimensions of the subtensor FTs. This process is executed at different tensor train formats. Both original and incremental FTs undergo the padding process and the flow entries in tensors ST_{FT}^{EX}, ST_{FT}^{GR}, and ST_{FT}^{BE} are synchronized as

$$ST_{\mathrm{FT}}^{\theta} = ST_{\mathrm{FT}}^{\theta(1)} \otimes ST_{\mathrm{FT}}^{\theta(2)} \otimes ST_{\mathrm{FT}}^{\theta(3)} \otimes \cdots \otimes ST_{\mathrm{FT}}^{\theta(N)}. \quad (8)$$

Algorithm 2 shows the control mechanism of the flow matching and synchronization phases. A unified tensor network is formed by using the incremental tensor train and original FT entries from all subtensors to remove redundancy and improve the overall performance. The detailed process and operations of the incremental tensor train decomposition model are discussed in the next section.

D. Phase IV: Incremental Tensor Train Decomposition

The Incremental tensor train decomposition process used in the previous phase is elaborated in detail in this section. The tensor train decomposition concept is similar to the "matrix state multiplication" in discrete problems. The prescribed X tensor is mapped into a tensor train format as

$$X^{'} = X^{'(1)} \otimes X^{'(2)} \otimes X^{'(3)} \otimes \cdots \otimes X^{'(N)} \quad (9)$$

where, $\otimes$ is the contraction operation in the X tensor.

The Nth-order tensor in tensor train format is shown in Fig. 5(a) and represented as

$$X^{'(n)} \sum \mathbb{R}^{r_{n-1} \times I_n \times r_n} \quad (10)$$

where, $n = 1, 2, \ldots, N$ and r_N is the rank of the tensor.

Now, the incremental tensor-train decomposition works in different steps as shown in Fig. 5 (b). These decomposition steps are explained as follows and are based on [19] and [20].

1) The initial tensor ($X \sum \mathbb{R}^{I_1 \times I_2 \times I_3 \times \cdots \times I_n \times \cdots \times I_N}$) with rank r is converted into tensor-train format.
2) Then, the incremental tensor ($Y \sum \mathbb{R}^{I_1 \times I_2 \times I_3 \times \cdots \times I_n \times \cdots \times I_N}$) with rank ($s$) is converted into the tensor-train format.
3) Zero's are added to the initial tensor (X) and incremental tensor ($X^{'}$) using the zero-padding technique by taking into consideration the indices of the initial tensors so as to balance the indices of the tensors.
4) Again, the zero-padded $\tilde{X}$ tensor ($\tilde{X} \sum \tilde{\mathbb{R}}^{I_1 \times I_2 \times I_3 \times \cdots \times I_{\tilde{n}} \times \cdots \times I_N}$) is converted into tensor train format with rank (r).
5) Similarly, the zero-padded $\tilde{Y}$ tensor ($\tilde{Y} = \tilde{\mathbb{R}}^{I_1 \times I_2 \times I_3 \times \cdots \times I_{\tilde{n}} \times \cdots \times I_N}$) is converted into tensor-train format using rank (s).

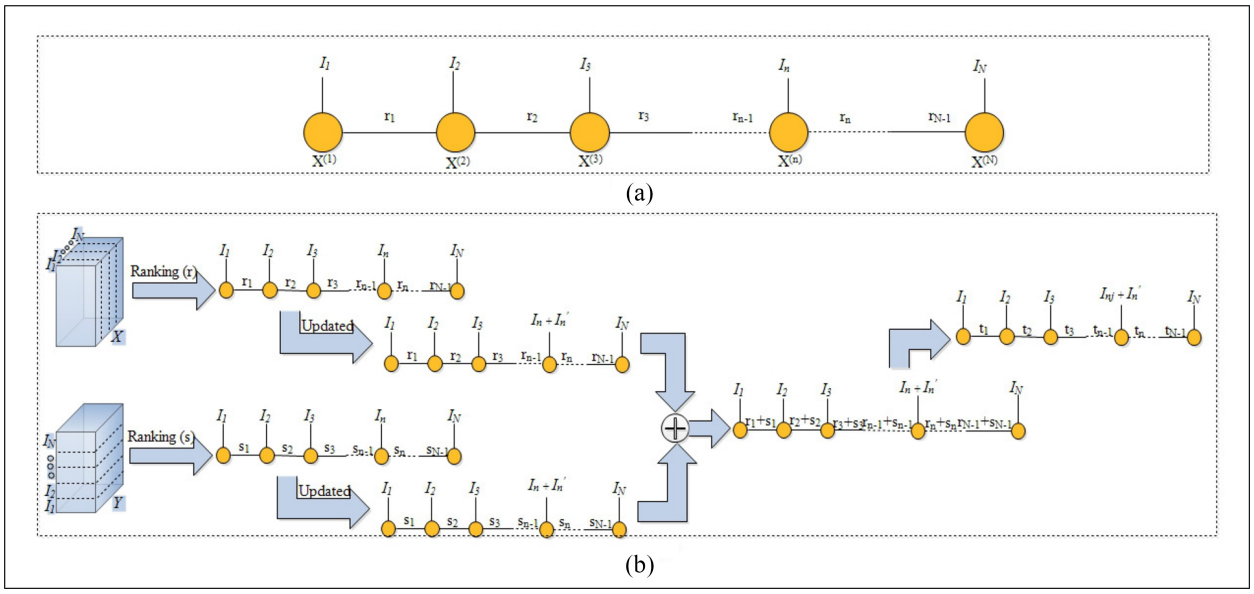

Fig. 5. Data flow in incremental tensor decomposition and synchronization.

6) A unified tensor Z is formed by adding incremental tensors $\tilde{X}$, $\tilde{Y}$ and initial tensors X,Y with rank t.

1) Tensor-Train Format of Zero Padding Tensor: Suppose that original tensor is $\underline{X'} \sum R^{I_1 \times I_2 \times \cdots I_k \times \cdots \times I_N}$ and incremental tensor train along k_{th} order is $\underline{Y'} \sum R^{I_1 \times I_2 \times \cdots I_k \times \cdots \times I_N}$, hence the updated tensor is represented as per [19] and [20], as follows:

$$\underline{Z'} \sum R^{I_1 \times I_2 \times \cdots I_k \times \cdots \times I_N}. \tag{11}$$

Therefore, all tensor-train cores of zero padding tensor $\underline{X'}$ can be achieved from tensor-train cores of the given tensor $\underline{X}$ without repeated computations. Here, all tensor-train cores remain unchanged except for the kth core. The tensor-train format of the zero padding tensor $\underline{X'}$ is represented as follows:

$$\underline{X'}(i_1, i_2, \ldots, i_N) = \underline{X'}^{(1)}(i_1)\underline{X'}^{(2)}(i_2) \cdots \underline{X'}^{(n)}(i_n) \cdots \underline{X'}^{(N)}(i_N)$$

Now, the rules of yielding all new tensor-train cores $\underline{X'}^{(n)}(i_n)$ $(n = 1, \ldots, N)$ of a zero padding tensor $\underline{X'}$ are defined according to [19] and [20], as follows:

$$\underline{X'}^{(n)}(i_n) = \begin{cases} 0, & n = k \text{ and } i_n > I_k \\ \underline{X}^{(n)}(i_n), & \text{otherwise.} \end{cases}$$

2) Addition of Two Tensors in Tensor-Train Format: Tensor-train format of the original zero padding tensor and incremental zero padding tensor can be represented as follows:

$$\underline{X'}(i_1, i_2, \ldots, i_N) = \underline{X'}^{(1)}(i_1), \underline{X'}^{(2)}(i_2), \ldots, \underline{X'}^{(N)}(i_N) \tag{12}$$

$$\underline{Y'}(i_1, i_2, \ldots, i_N) = \underline{Y'}^{(1)}(i_1), \underline{Y'}^{(2)}(i_2), \ldots, \underline{Y'}^{(N)}(i_N). \tag{13}$$

Suppose the tensor-train format of the updated tensor is represented according to [19] and [20], as follows:

$$\underline{Z'}(i_1, i_2, \ldots, i_N) = \underline{Z'}^{(1)}(i_1)\underline{Z'}^{(2)}(i_2) \cdots \underline{Z'}^{(N)}(i_N) \tag{14}$$

where $\underline{Z'}^{(n)}(i_n) \sum R^{t_{n-1} \times t_n} (n = 1, \ldots, N; t_0, t_N = 1)$ Then, the computation of tensor-train format of updated tensor can be converted to merge the original and incremental tensor-train cores. According to addition operation, the generation rule of updated tensor-train cores is defined according to [19] and [20], as follows:

$$\underline{Z'}^{(i_n)} = \begin{cases} \big(\underline{X'}(i_1)\big)\big(\underline{Y'}(i_1)\big), & n = 1 \\ \begin{pmatrix} \underline{X'}(i_n) & 0 \\ 0 & \underline{Y'}(i_n) \end{pmatrix}, & n = 2, \ldots, N-1 \\ \begin{pmatrix} \underline{X'}^{(N)}(i_N) \\ \underline{Y'}(N)(i_N) \end{pmatrix}, & n = N. \end{cases} \tag{15}$$

Now, using (13) and (13), (14) can be represented as

$$\underline{Z'}(i_1, i_2, \ldots, i_N) = \underline{Z'}^{(1)}(i_1), \underline{Z'}^{(2)}(i_2), \ldots, \underline{Z'}^{(N)}(i_N) \Rightarrow \underline{X'}(i_1, i_2, \ldots, i_N) + \underline{Y'}(i_1, i_2, \ldots, i_N). \tag{16}$$

It can be seen that any entry of tensor $\underline{Z'}$ is the sum of the corresponding entries of tensors $\underline{X'}$ and $\underline{Y'}$. Also, tensor-train rank of updated tensor $\underline{Z'}$ are the sum of corresponding tensor-train ranks of the original and incremental zero-padding tensors $\underline{X'}$ and $\underline{Y'}$.

3) Orthogonalization and Compression of Updated Tensor-Train Cores: Let T_{itt}, T_{ttzero}, T_{ttadd} and T_{ttorth} denote the times consumed for execution of incremental tensor-train, zero padding, addition, and orthogonalization, respectively. Then, the time consumed for incremental tensor-train decomposition process is given according to [19], and [20], as follows:

$$T_{\text{ITTD}} = T_{\text{itt}} + T_{\text{ttzero}} + T_{\text{ttadd}} + T_{\text{ttorth}}. \tag{17}$$

Now, given an Nth order tensor $\underline{X} \sum R^{I_1 \times I_2 \times \cdots \times I_N}$, if incremental dimensionality along the specific order is I', then the maximum dimensionally of original incremental and updated tensors is I, I' and $I + I'$, respectively. Now, to compute the tensor-train decomposition of incremental tensor, $N-1$ successive singular value decompositions (SVDs) must be executed. If we consider the first $N/2$ SVDs, then we can calculate the time consumed according to the size of unfolding matrices according to [19], and [20], as follows:

$$T_{\text{itt}} = O\Big(I^{\frac{N}{2}+2} r_{\frac{N}{2}-1^2}\Big). \tag{18}$$

TABLE II
EXPERIMENTAL SETTINGS

Parameter	Setting
SDN controller	OpenDaylight (ODL)
Traffic controller	POX controller (1.5 GHz, 2GB RAM, 20 GB HDD)
Raspberry Pi	1.2 GHz CPU, 1 GB RAM, WiFi, 1 built-in Ethernet port
Packet Size	512 bytes
Packet Send Interval	10-20 s
Traffic Rate	100 Mbps
Ethernet Port speed	100 Mbps
Service Rate	Variable

Hence, total time of orthogonalization and compression of tensor-train cores is given according to [19], and [20], as follows:

$$T_{\text{ttorth}} = O\left(32NTr^4\right). \tag{19}$$

Now, the total time consumed is given according to [19], and [20], as follows:

$$T_{\text{ITTD}} = \begin{cases} O\left(I^{N-1}I'^2 + NIr^4\right), r \leq I^{\frac{N-2}{4}} \\ O\left(I^{\frac{N}{2}+2}r^2 + NIr^4\right), r > I^{\frac{N-2}{4}}. \end{cases} \tag{20}$$

For details about the incremental tensor train decomposition and the above described formulations, refer to [19] and [20].

V. EVALUATION AND VALIDATION: CASE STUDY OF SMART UNIVERSITY CAMPUS

For the validation of the proposed scheme, an evaluation has been performed using Raspberry Pi 3 (1.2 GHz CPU, 1 GB RAM, WiFi, 1 built-in Ethernet port and up to 4 USB ports) nodes for experimentation. Three nodes are dedicated to emulated IoT sensors and two nodes act as edge devices. The network traffic flow is collected from the deployed testbed in a smart University campus. The SDN-IoT testbed comprises of sensors, Raspberry Pi 3 (1.2 GHz CPU, 1 GB RAM, WiFi, 1 built-in Ethernet port and up to 4 USB ports), OF switches (Open vSwitch), bluetooth devices, OpenDaylight (ODL), and POX controllers (installed on Ubuntu-based servers). Table II shows the details about the experimental setup.

The SDN network topology has been created using a Mininet emulator which receives heterogeneous network traffic from IoT sensors. A virtual POX controller (traffic controller) has been deployed as a deep Boltzmann machine for flow analysis and classification. This traffic controller is deployed in a dockerized environment in an Intel core processor (1.5 GHz, 2 GB RAM, 20 GB HDD) which terrorizes and classifies the incoming IoT traffic. Once the traffic has been classified, the ODL controller collects the flow statistics and builds the flow rules accordingly. Fig. 6 shows the layout of the IoT-based network setup for the proposed case study.

The obtained results for the proposed framework were compared with the standard SDN architecture which follows traditional flow matching and building statistics. Figs. 7 and 8 depict the delay and throughput plots for the proposed framework in contrast with the standard SDN architecture. The variation of delay and throughput with respect to and increase in the number of traffic flows clearly outperforms its existing counterpart with a significant difference. This is due to the

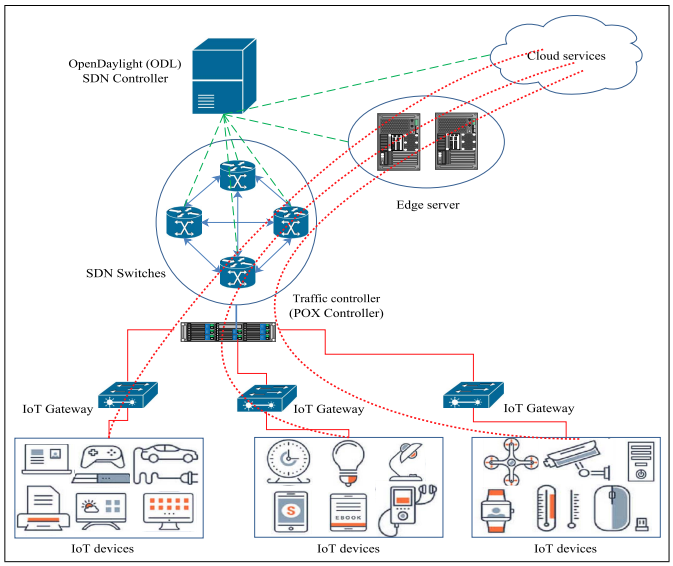

Fig. 6. IoT network case study.

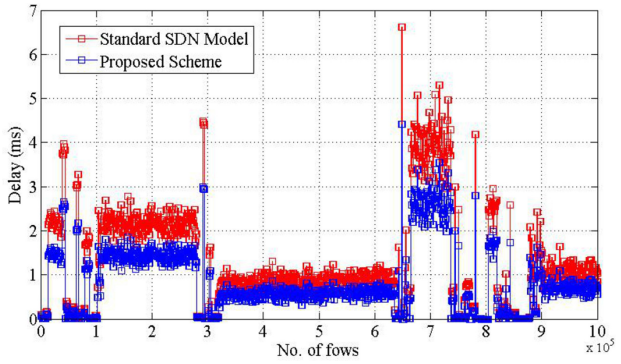

Fig. 7. Delay.

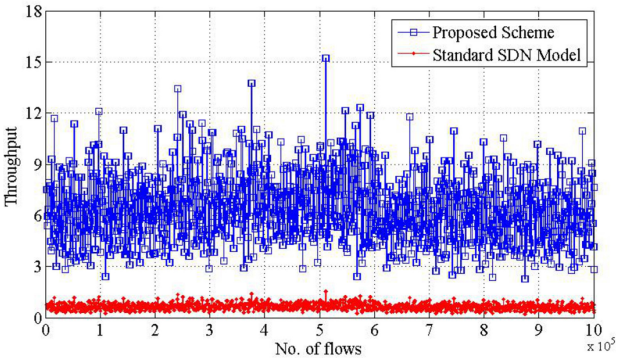

Fig. 8. Throughput.

flow classification and optimal rule-space management using the proposed architecture.

Fig. 9 shows the throughput obtained for each insertion and lookup operations performed in the FTs. Fig. 9(a) depicts the variation of throughput for insertion operation. It is visible that initially, the throughput decreases with an increase in the table occupancy index but as it reaches a table occupancy of index (0.6), it slowly stabilizes. But, the throughput for insertion operation is significantly higher than the conventional counterpart. Fig. 9(b) illustrates the throughput for lookup

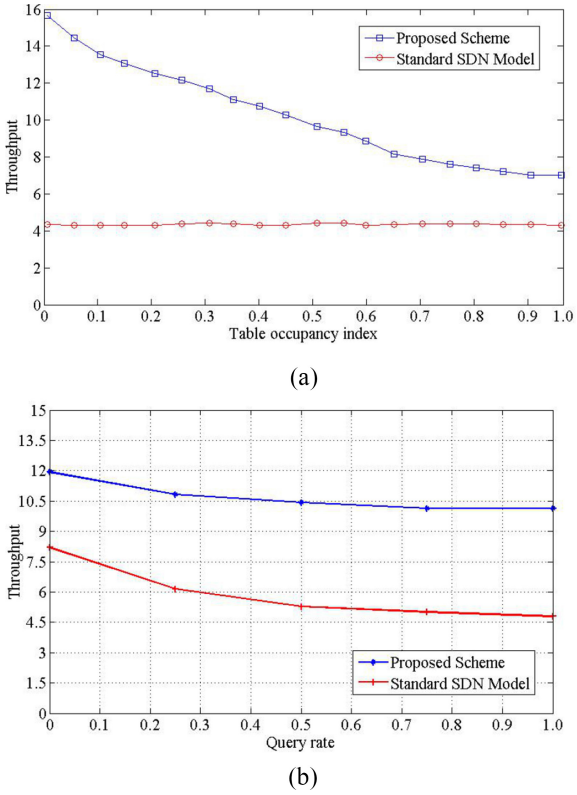

Fig. 9. Results obtained for (a) insertion and (b) lookup operations.

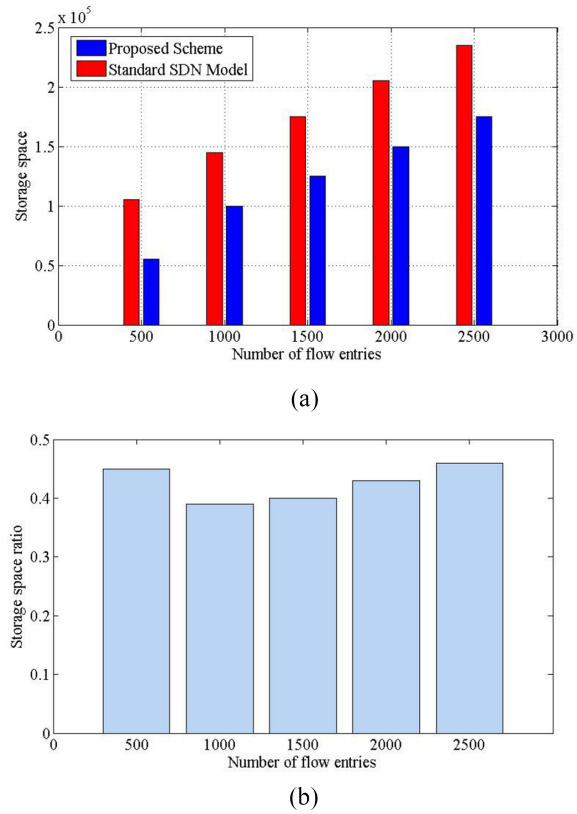

Fig. 10. Storage space analysis. (a) Storage space. (b) Storage space ratio.

operation in the FTs with respect to an increase in query rate. It is evident from the obtained results that the throughput achieved for the proposed scheme is significantly higher and stable than its existing variant. Fig. 10 shows the storage space analysis using the proposed scheme. Fig. 10(a) shows the growth of storage space with an increase in the FT entries. The storage space consumed is much lesser as compared to the conventional counterpart. Fig. 10(b) further supports this fact as the storage space ratio shows almost 40%–45% less space consumption. Even more, the deep computational model was also evaluated with respect to different performance metrics such as correctly classified instances, true positive rate, false positive rate, precision, recall, and F-score. Table III shows the obtained values for these metrics with respect to the simple DBF model and Naive Bayes classifier. The proposed model clearly outperforms the comparative variants with respect to all the considered performance metrics.

TABLE III
EVALUATION RESULTS

Algorithm	CCI(%)	TP rate	FP rate	Precision	Recall	F-Score
Naive Bayes	72.26	0.723	0.267	0.751	0.723	0.709
DBF	82.88	0.829	0.164	0.807	0.829	0.81
Proposed	88.59	0.886	0.166	0.83	0.886	0.84

VI. CONCLUSION

The standard structure of the incoming traffic consists of many unwanted dimensions or attributes which are not necessary for transmission, decision making, processing, or storage, and therefore these may be ignored. For this reason, a tensor representation or tensor network format has been used to tackle this situation. For achieving this goal, we tried to embed network intelligence into the flow forwarding architecture of SDN through the deep Boltzmann machine and the incremental tensor train decomposition model. The evaluations performed have been tested on the basis of different performance metrics like throughput, delay, and storage space with respect to the variation in the number of traffic flows, table occupancy index, query rate, and the number of flow entries. From the analysis, the proposed architecture seems promising and outperforms its convention counterpart.

REFERENCES

[1] K. Kaur, S. Garg, G. S. Aujla, N. Kumar, J. J. P. C. Rodrigues, and M. Guizani, "Edge computing in the industrial Internet of Things environment: Software-defined-networks-based edge-cloud interplay," *IEEE Commun. Mag.*, vol. 56, no. 2, pp. 44–51, Feb. 2018.

[2] G. S. Aujla, R. Chaudhary, N. Kumar, A. K. Das, and J. J. P. C. Rodrigues, "SecSVA: Secure storage, verification, and auditing of big data in the cloud environment," *IEEE Commun. Mag.*, vol. 56, no. 1, pp. 78–85, Jan. 2018.

[3] R. Chaudhary, G. S. Aujla, N. Kumar, and J. J. P. C. Rodrigues, "Optimized big data management across multi-cloud data centers: Software-defined-network-based analysis," *IEEE Commun. Mag.*, vol. 56, no. 2, pp. 118–126, Feb. 2018.

[4] G. S. Aujla, N. Kumar, A. Y. Zomaya, and R. Ranjan, "Optimal decision making for big data processing at edge-cloud environment: An SDN perspective," *IEEE Trans. Ind. Informat.*, vol. 14, no. 2, pp. 778–789, Feb. 2018.

[5] G. S. Aujla, R. Chaudhary, N. Kumar, R. Kumar, and J. J. P. C. Rodrigues, "An ensembled scheme for QoS-aware traffic flow management in software defined networks," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2018, pp. 1–7.

[6] M. M. Tajiki, S. Salsano, L. Chiaraviglio, M. Shojafar, and B. Akbari, "Joint energy efficient and Qos-aware path allocation and VNF placement for service function chaining," *IEEE Trans. Netw. Serv. Manag.*, vol. 16, no. 1, pp. 374–388, Mar. 2019.

[7] C.-C. Chuang, Y.-J. Yu, and A.-C. Pang, "Flow-aware routing and forwarding for SDN scalability in wireless data centers," *IEEE Trans. Netw. Serv. Manag.*, vol. 15, no. 4, pp. 1676–1691, Dec. 2018.
[8] M. Hayes, B. Ng, A. Pekar, and W. K. G. Seah, "Scalable architecture for SDN traffic classification," *IEEE Syst. J.*, vol. 12, no. 4, pp. 3203–3214, Dec. 2018.
[9] B. Mao *et al.*, "A tensor based deep learning technique for intelligent packet routing," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2017, pp. 1–6.
[10] D. Kaur, G. S. Aujla, N. Kumar, A. Y. Zomaya, C. Perera, and R. Ranjan, "Tensor-based big data management scheme for dimensionality reduction problem in smart grid systems: SDN perspective," *IEEE Trans. Knowl. Data Eng.*, vol. 30, no. 10, pp. 1985–1998, Oct. 2018.
[11] L. Kuang, L. T. Yang, X. Wang, P. Wang, and Y. Zhao, "A tensor-based big data model for qos improvement in software defined networks," *IEEE Netw.*, vol. 30, no. 1, pp. 30–35, Jan./Feb. 2016.
[12] I. Maity, A. Mondal, S. Misra, and C. Mandal, "Tensor-based rule-space management system in SDN," *IEEE Syst. J.*, to be published.
[13] R. B. Basat, G. Einziger, R. Friedman, and Y. Kassner, "Optimal elephant flow detection," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, 2017, pp. 1–9.
[14] S. Garg, K. Kaur, N. Kumar, and J. J. P. C. Rodrigues, "Hybrid deep-learning-based anomaly detection scheme for suspicious flow detection in SDN: A social multimedia perspective," *IEEE Trans. Multimedia*, vol. 21, no. 3, pp. 566–578, Mar. 2019.
[15] S. Garg, K. Kaur, G. Kaddoum, S. H. Ahmed, and D. N. K. Jayakody, "SDN-based secure and privacy-preserving scheme for vehicular networks: A 5G perspective," *IEEE Trans. Veh. Technol.*, vol. 68, no. 9, pp. 8421–8434, Sep. 2019.
[16] P. Wang, L. T. Yang, Y. Peng, J. Li, and X. Xie, "m^2t^2: The multivariate multi-step transition tensor for user mobility pattern prediction," *IEEE Trans. Netw. Sci. Eng.*, to be published, doi: 10.1109/TNSE.2019.2913669.
[17] L. Kuang, L. T. Yang, Q. Zhu, and J. Chen, "Secure tensor decomposition for big data using transparent computing paradigm," *IEEE Trans. Comput.*, vol. 68, no. 4, pp. 585–596, Apr. 2019.
[18] S. Garg, K. Kaur, N. Kumar, G. Kaddoum, A. Y. Zomaya, and R. Ranjan, "A hybrid deep learning-based model for anomaly detection in cloud datacenter networks," *IEEE Trans. Netw. Serv. Manag.*, vol. 16, no. 3, pp. 924–935, Sep. 2019.
[19] H. Liu, L. T. Yang, Y. Guo, X. Xie, and J. Ma, "An incremental tensor-train decomposition for cyber-physical-social big data," *IEEE Trans. Big Data*, to be published.
[20] P. Wang, L. T. Yang, G. Qian, J. Li, and Z. Yan, "Ho-OTSVD: A novel tensor decomposition and its incremental decomposition for cyber-physical-social networks (CPSN)," *IEEE Trans. Netw. Sci. Eng.*, to be published.

Amritpal Singh received the B.Tech. and M.Tech. degrees in computer science and engineering from Punjab Technical University, Jalandhar, India, in 2003 and 2013, respectively. He is currently pursuing the Ph.D. degree with Chandigarh University, Mohali, India.

He is also working as an Assistant Professor with the Computer Science and Engineering Department, Chandigarh University. He has many research contributions in the area of cloud computing, edge computing, and software defined networks.

Gagangeet Singh Aujla (S'15–M'18–SM'19) received the B.Tech. and M.Tech. degrees in computer science and engineering from Punjab Technical University, Jalandhar, India, in 2003 and 2013, respectively, and the Ph.D. degree in computer science and engineering from the Thapar Institute of Engineering and Technology, Patiala, India, in 2018.

He is currently working as a Postdoctoral Research Associate with the School of Computing, Newcastle University, Newcastle Upon Tyne, U.K. He is also an Associate Professor with the Computer Science and Engineering Department, Chandigarh University, Mohali, India.

Dr. Aujla is a recipient of the 2018 IEEE TCSC outstanding dissertation award at Guangzhou China. He has been Workshop Chair for conferences including IEEE Infocom, IEEE Globecom, IEEE ICC, and IEEE PiCom. He is on the Editorial Board of the *Sensors Journal*. He has been a Guest Editor for special issues in the IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS, IEEE NETWORK, *Computer Communications* (Elsevier), *Software: Practice and Engineering* (Wiley), *Transactions on Emerging Telecommunications Technologies* (Wiley), and *Security and Privacy* (Wiley).

Sahil Garg (S'15–M'18) received the Ph.D. degree from the Thapar Institute of Engineering and Technology, Patiala, India, in 2018.

He is currently working as a Postdoctoral Research Fellow with the École de technologie supérieure, Université du Québec, Montreal, QC, Canada. He has many research contributions in the area of machine learning, big data analytics, security and privacy, Internet of Things, and cloud computing. He has over 50 publications in highly ranked journals and conferences, including over 20 IEEE TRANSACTIONS/Journal papers.

Dr. Garg was the recipient of prestigious Visvesvaraya Ph.D. Fellowship from the Ministry of Electronics and Information Technology under Government of India from 2016 to 2018. For his research, he has also been awarded the IEEE ICC Best Paper Award in 2018 at Kansas City, USA. He serves as an Associate Editor for the *International Journal of Communication Systems* (Wiley) and *Human-Centric Computing and Information Sciences* (Springer). He has been a Lead Guest Editor for a number of special issues in prestigious journals and magazines, such as the IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS, the IEEE TRANSACTIONS ON INTELLIGENT TRANSPORTATION SYSTEMS, IEEE INTERNET OF THINGS JOURNAL, IEEE NETWORK, and *Future Generation Computing Systems* (Elsevier). He also serves as the General and Publicity Chair for workshops, including IEEE Infocom, IEEE Globecom, and IEEE ICC.

Georges Kaddoum (M'11) received the bachelor's degree in electrical engineering from the École Nationale Supérieure de Techniques Avancés, Brest, France, the M.S. degree in telecommunications and signal processing from Telecom Bretagne (ENSTB), Brest, in 2005, and the Ph.D. degree in signal processing and telecommunications from the National Institute of Applied Sciences, Toulouse, France, in 2009.

He is currently an Associate Professor and the Tier 2 Canada Research Chair with the École de Technologie Supérieure, University of Québec, Montreal, QC, Canada. His recent research activities cover wireless communication networks, resource allocations, security and space communications, and navigation.

Gurpreet Singh received the B.Tech. and M.Tech. degrees in computer science and engineering from Punjab Technical University, Jalandhar, India, in 2010 and 2014, respectively. He is currently pursuing the Ph.D. degree with Chandigarh University, Mohali, India.

He is also working as an Assistant Professor with the Computer Science and Engineering Department, Chandigarh University. He has many research contributions in the area of computer networks and routing protocols.