Towards a Novel Congestion Notification Algorithm for a Software-Defined Data Center Networks

Hai-Anh TRAN*, Thi-Thanh-Tu NGUYEN*, Sami SOUIHI †, and Abdelhamid MELLOUK†
*School of Information and Communication Technology
Hanoi University of Science and Technology, Hanoi, Vietnam
Email: anhth@soict.hust.edu.vn, tu.nguyenthithanh@hust.edu.vn
†LISSI-TincNET Research Team University Paris-Est Creteil, France
Email: sami.souihi@u-pec.fr, mellouk@u-pec.fr

***Abstract*—Nowadays, besides the evident benefits of Data Centers (DC) in terms of information storage and processing, that posed also a few problems needed to be addressed. The computer network research community has been dedicated to resolve one of the severe problems of Data Center Networks (DCN): the congestion notification problem. The quality of connection links is the decisive factor for the success of a DC in providing higher bandwidth and low interconnection latency for high performance cluster computing. Therefore, the congestion detection and reaction mechanisms are vital to the performance of a DC. There are divers congestion notification algorithms, such as Backward Congestion Notification (BCN), Quantized Congestion Notification (QCN), Forward Explicit Congestion Notification (FECN,) and E-FECN. However, the common disadvantages of those algorithms is the lack of ability to predict congestion. In this paper, we propose an AI-based congestion notification algorithm in applying the Long short-term memory (LSTM) algorithm and its variants, such as GRU, Skip-LSTM, and Bi-Directional LSTM, into the original BCN algorithm, called IBCN, in order to increase the capacity of predicting link congestion of the original one. We also implement IBCN in a Software-defined DCN that is a combination of the DCN and the Software-defined Network (SDN) architecture. The obtained experimental results showed that the IBCN outperforms the original BCN at any network load levels (from 10% to 90%) with the average improvement of 50% of throughput. In addition, the proposed solution gives also better results than other traditional linear prediction algorithms.**

***Index Terms*—Data Center Networks, Software-Defined Network, congestion notification protocol, LSTM, backward congestion notification**

I. Introduction

The past two decades have witnessed an explosive development in Data Centers (DC) that mainly host a huge number of applications and services, such as social networks, real-time multimedia services, entertainment services, online gaming, etc. Today, modern DC contains millions servers that are connected via routers, switches, and high-speed links. The main goal of a DC is to store and process the data/information in maximizing efficiency while maintaining a low cost. To ensure that the goal is achieved, extensive research efforts have been enforced to resolve divers challenges of Data Center Networks (DCN), such as network architecture design, TCP Incast, power consumption, and congestion notification algorithms [1]. Among these problems, we focus on the congestion notification problem in the Layer 2.

Being a essential protocol for node-to-node DCN interconnection network, Ethernet gives various benefits such as ease of management, low cost, and ubiquitous connectivity [2]. However, the Ethernet's policy, *best-effort* transmissions, is its weakness because it drop packets when the switch is overloaded. Today, new Ethernet protocols are being developed to help DCN to support low latency connection links for high performance computing [3] . The main goal of the Ethernet improvements is to give congestion notification in order to reinforce the transport accuracy while increasing the efficiency of the transport protocol (e.g. TCP). Hence, congestion notification mechanism is considered as a traffic management solution in the Layer 2 that observes the process for queuing the data frames of switches. Then, it detects the congestion and signals the source node so that the source can react in a consistent manner with this warning to avoid frame losses. Consequently, the congestion control mechanism of the reliable transport protocol, TCP, can benefit from this Layer 2 congestion notification protocol because it has the ability to react quickly to link congestion. In addition, thanks to the congestion notification protocol, the bandwidth usage is greatly improved with the new high rate Ethernet link of 10Gpbs.

There are various congestion notification algorithms for DCN, such as FECN [4], enhanced FECN (E-FECN) [5], QCN [6], and BCN [7]. Besides the proven advantages, the mentioned proposals still have a common disadvantage: the lack of a predicting congestion mechanism. That motivates us to apply a Artificial Intelligence (AI) technique to predict congestion in order to address this problem. Fortunately, AI has been advanced very robustly, thanks to the evolution of computer technology, such as GPU and TPU [8]. It presents the convenience to apply AI to the network control mechanisms, supporting effective prediction techniques. This means every switches in a DCN will be an AI node, which is able to make decision automatically. Among the aforementioned algorithms, we decide to base on the queue-based and reactive signaling algorithm BCN due to its advantages [1]: fast congestion regulation, large throughput oscillation, link disconnection supported, and fast start supported.

However, it is impractical to apply AI to a traditional DCN because each switch has only a local view and local control. Hence, a centralized control architecture, e.g. Software-defined Network (SDN) [9], is a promising candidate. The ultimate idea of a SDN architecture is to centralize all the control plane of all nodes in a network into a controller (called SDN controller). Therefore, it provides a global view of the whole system and makes it easier to integrate AI into the DCN. In fact, the trend of combining the SDN and DCN architecture has been in research for the last few years in different research works [10]–[13].

To sum up, the main contribution of this paper is to propose an AI-based congestion notification algorithm in basing on the BCN algorithm for a Software-Defined Data Center Network (SDDCN). Especially, the AI techniques we used are the Long short-term memory (LSTM) and its variants, such as Gated Recurrent Unit (GRU) [14], Skip-LSTM [15], and Bi-Directional LSTM [16]. They are artificial recurrent neural network (RNN) architectures used in the field of deep learning.

The paper is structured as follows. In section II, we survey some related research works. Section III presents the fundamentals of the original BCN algorithm, the LSTM RNN, and its variants. The proposed IBCN is introduced in section IV where we explain how we integrate the LSTM and its variants to the original BCN, called IBCN, and how we implement IBCN in a SDDCN. Section V will show some obtained experimental results.

II. Related works

Jiang et al. [4] proposed the algorithm Forward Explicit Congestion Notification (FECN) that uses a mechanism to control the feedback rate. The source node begins by sending with full rate and uses periodically a probing mechanism to check the congestion of the routing path. In the case where, the available bandwidth of a switch in the path is not enough, the rate controller adjusts the sending rate in basing on the modified rate field of the received probe message. In an extension version of FECN, E-FECN [5], the switches under intense condition are able to send back directly to the source node. However, the FECN and E-FECN have a slow congestion regulation and a small throughput oscillation. In addition, FECN and E-FECN don't support the fast start and the link disconnection.

Alizadeh et al. [6] proposed the QCN algorithm that consists of two sub-algorithms: Congestion Point Algorithm and Reaction Point Algorithm. The Congestion point algorithm applies a sampling method to help a switch to detect a congestion situation. This switch then informs the congestion level to the source node. For the Reaction point algorithm, the rate limiter of a source will decrease the sending rate after receiving the feed back of the congestion point.

Bergasamo et al. [7] introduced the BCN algorithm with three modules: Congestion Detection, Backward Signaling, and Source Reaction. In the congestion detection module, a switch samples periodically a packet and determines the instantaneous queue-size. The latter is used in the Backward Signaling module in comparing with two thresholds: Q_{eq} (equilibrium queue length) and Q_{sc} (the severe congestion queue length). For the Source Reaction phase, the source will reacts in depending on the received message from the rate regulator. Based on its advantages, such as fast congestion regulation, large throughput oscillation, link disconnection supported, and fast start supported, we decide to leverage BCN algorithm to propose an AI-based algorithm.

Gholami et al. [17] based on the Openflow protocol to control the congestion in Software-defined DCNs. Their main idea is to monitor the congestion of all network links at the SDN controller in using the port statistics of the Openflow-enabled switches. All data flows that reside in congested links are rerouted to other links with more resources.

Concerning the idea of applying Machine Learning (ML) technique to the Layer 2 congestion notification mechanism in a DCN, to the best of our knowledge, there is only one research work [18] that addressed that problem. Most of the research works attempting to apply ML to the congestion control mechanism today focus on the layer 4 for the transport protocol (TCP).

Majidi et al. [18] proposed a ML-based dynamic threshold control scheme for Explicit Congestion Notification (ECN) marking in DCN, called DC-ECN (Data Center-Explicit Congestion Notification). They use a ML-based classifier to separate the mice and elephant flows to place them in the desired queues. DC-ECN then adjusts the buffer space by keeping the same ECN thresholds instead of increasing a threshold to enhance the throughput that also increases the latency.

The main disadvantage of the aforementioned solutions is the lack of predictability. They always resolve the congestion problem in a DCN in using only instantaneous information. This issue will obviously affect the effectiveness of that congestion detection mechanism. That motivates us to propose a ML-based congestion notification algorithm with the ability to predict congestion.

III. Fundamentals of the original BCN algorithm and LSTM

In order to prepare the knowledge base for our proposals, in this section, we briefly present the fundamentals of the original BCN algorithm, the LSTM RNN and its variants including GRU, Skip-LSTM, and Bi-Directional LSTM.

A. Original BCN algorithm

As mentioned above, the BCN algorithm consists of three modules: Congestion Detection, Signaling, and Source Reaction.

In the first module, *Congestion Detection*, the incoming packets are sampled in every switch with the probability P_m. Two thresholds are chosen to express the congestion tolerance at a switch: Q_{eq} and Q_{sc} (Fig. 1). After sampling a packet, the congestion measure is determined in Eq. 1.

$$e_i = -(q(t) - Q_{eq} + \omega * (Q_a(t) - Q_d(t))) \qquad (1)$$

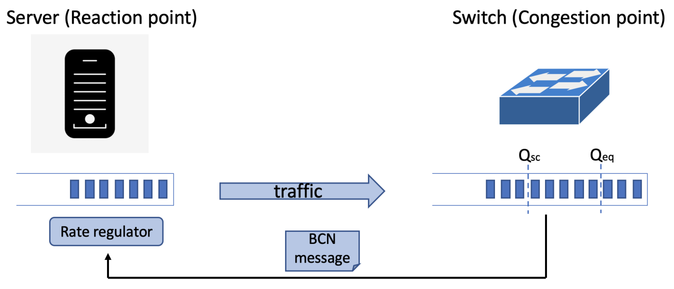

Fig. 1. BCN model

where $q(t)$ represents the queue-size at moment t. $Q_a(t)$ is the number of arrived packets and $Q_d(t)$ is the number departed packets at moment t. ω denotes the non-negative constant weight.

The value of the congestion measure e_i will be used in the Source Reaction phase.

The main purpose of the second phase, *Backward Signaling*, is to decide whether or not to send back the signal message to the source in basing on the comparison of $q(t)$ with the thresholds. After sampling a packet, the switch sends back the feedback to the source node in using the 802.1Q tag format [7]. The feedback process is is carried out as follows:

1) If $q(t) < Q_{eq}$:
   a) If the tag of the rate regulator doesn't exist: the switch doesn't send any feedback message.
   b) the tag of the rate regulator exists: the switch sends back a positive BCN message.
2) If $Q_{eq} < q(t) < Q_{sc}$: the switch sends back a BCN NORMAL message.
3) If $q(t) > Q_{sc}$: the switch sends back a BCN STOP message.

For the third phase, *Source Reaction*, the souce node will react differently with two types of received feedback message: the BCN STOP message and the BCN NORMAL message:

- After receiving a BCN STOP message, the rate regulator halts the sending process for a random period. It will restore the sending process with a ratio of C to K, where K represents the flow density in the DCN, and C is the capacity of the congested link.
- After receiving a BCN NORMAL message, the rate regulator regulates the rate as shown in the Eq. 2.

$$r_i = \begin{cases} r_i + G_i e_i R_u & \text{if } e_i > 0 \\ r_i(1 + G_d e_i) & \text{if } e_i < 0 \end{cases} \tag{2}$$

where R_u denotes the increase rate unit parameter. G_i is the the additive increase parameter, and G_d is the multiplicative decrease gain parameter. The congestion measure e_i is calculated by using Eq. 1 and extracted from the received BCN message.

B. Long Short Term Memory Neural Networks

LSTM is considered as a solution for the problem of long-term dependencies of the original Recurrent Neural Network (RNN). The main advantage of the LSTM is that it possesses a constant error flow by using a set of memory blocks, which can store the temporal state of the network. LSTM provides also special multiplicative units called gates that control the information flow. These advantages are appropriate for the congestion prediction in a DCN. That motivated us to apply LSTM solution to predict the queue size information of switches.

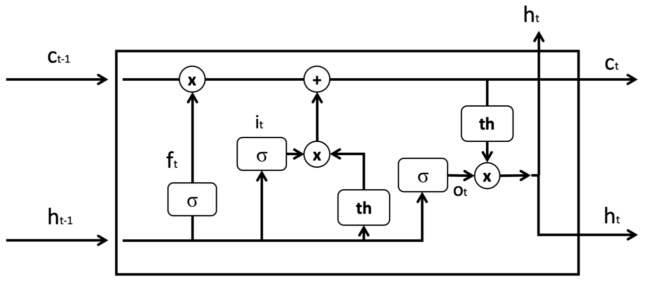

Fig. 2. LSTM cell

Fig. 2 has shown a graphical representation of our implemented LSTM cell that consists of three gates, three input variables, and two output variables as follows:

- input gate: i_t
- output gate: o_t
- forget gate: f_t
- current input: x_t
- previous output: h_{t-1}
- previous cell state: c_{t-1}
- current output: h_t
- current cell state: c_t

The implemented LSTM cell is defined as a neural network where the input vector, $x = (x(1), x(2), \ldots x(t))$, is equivalent to the vector of queue-size of a switch from moment 1 to t, $p = (p(1), p(2), \ldots p(t))$. The output vector is $h = (h(t+1), h(t+2), \ldots h(t+k))$, which is equivalent to the queue-size vector $p = (p(t+1), p(t+2), \ldots p(t+k))$. In other words, k next values of the value $p(t)$ are predicted. The output vector is calculated through the following layers:

- the forget gate sigmoid layer at moment t: f_t is determined by Eq. 3.

$$f_t = \sigma(W_f \bullet [h_{t-1}, x_t] + b_f) \tag{3}$$

  where σ is the sigmoid function. W_f is the matrix of weights from the forget layer. h_{t-1} and x_t are the previous output and the input vector, respectively. b_f is the correspondent bias.
- the input gate sigmoid layer at moment t: i_t is calculated by Eq. 4.

$$i_t = \sigma(W_i \bullet [h_{t-1}, x_t] + b_i) \tag{4}$$

  where W_i is the matrix of weights from the input layer and b_i is the correspondent bias.

- the cell state at moment t: C_t is calculated by the Eq. 5.

$$C_t = f_t \otimes C_{t-1} + i_t \otimes \tanh(W_c \bullet [h_{t-1}, x_t] + b_c) \quad (5)$$

  where i_t is the input gate. W_c and b_c are the weight matrix for the cell and the corresponded bias, respectively.
- the output gate sigmoid layer at moment t: o_t is calculated by Eq. 6.

$$o_t = \sigma(W_o \bullet [h_{t-1}, x_t] + b_o) \quad (6)$$

  where W_o and b_o are the weight matrix from the output layer and the corresponded bias, respectively.
- The final stage is used for calculating the output h_t that is determined by the multiplication operation $\otimes$ between output gate layer and $\tanh$ layer of the current cell state C_t (Eq. 7).

$$h_t = o_t \otimes \tanh(C_t) \quad (7)$$

The obtained output value h_t will be pushed to the network as the previous state for the next LSTM cell.

C. Variants of LSTM algorithm

As aforementioned, some variants of LSTM algorithm are implemented to compare its different impacts on the proposed IBCN algorithm. Three permutations of LSTM algorithm have been chosen: GRU, Skip-LSTM, and Bi-directional LSTM.

1) Gated Recurrent Unit (GRU): Cho et al. introduced a variant of the original LSTM algorithm in 2014, called GRU [14], [19], [20]. The advantage of GRU over original LSTM is that it has shorter training time and less computation to adjust the hidden layers. In GRU architecture, there are two gates: 1) the update gate determines the significance of the previous memory cell; 2) the reset gate establishes the combination of the new the previous memory cell with the input cell. Similar to other RNNs, for each time step, GRU creates an output that is used to train the network in using gradient descent.

2) Skip-LSTM: As a variant of LSTM, Skip-LSTM is introduced by Campos et al. [15]. The main idea of Skip-LSTM is to skip state updates in the computation graph in order to reduce the number of sequential operations. The advantage of the Skip-LSTM is that in a complex models or long sequences. Besides, the optimization task is simpler with fewer update steps for gradient backpropagation process.

3) Bi-directional LSTM: Mike et al. introduced Bi-directional LSTM [16] with the purpose of separating the hidden layers into two components: forward state and backward state sequence. Therefore, the Bi-directional LSTM is able to approach the preceeding and succeeding contexts. It is implemented with the equations 8, 9, and 10.

$$\overrightarrow{h_t} = \sigma(W_{x\overrightarrow{h}} x_t + W_{\overrightarrow{h}\overrightarrow{h}} \overrightarrow{h}_{t-1} + b_{\overrightarrow{h}}) \quad (8)$$

$$\overleftarrow{h_t} = \sigma(W_{x\overleftarrow{h}} x_t + W_{\overleftarrow{h}\overleftarrow{h}} \overleftarrow{h}_{t-1} + b_{\overleftarrow{h}}) \quad (9)$$

$$y_t = W_{\overrightarrow{h}y} \overrightarrow{h}_t + W_{\overleftarrow{h}y} \overleftarrow{h}_t + b_y \quad (10)$$

where $\overrightarrow{h}$ and $\overleftarrow{h}$ are forward state sequence and backward state sequence, respectively.

IV. Proposed IBCN algorithm in a Software-Defined Data Center Network

As shown in the Equations 1 and 2, the congestion detection process and the reaction process are both based on the value $p(t)$, which represents the instantaneous queue-size at moment t. Hence, the original algorithm BCN has revealed a lack of ability to predict congestion. That motivates us to propose the IBCN (Intelligent BCN), which is able to predict the queue size, thereby predicting also the potential congestion risk. Concretely, we apply the Long Short-Term Memory (LSTM) algorithm and its variants for congestion prediction. However, the congestion prediction module does not work efficiently if it is executed separately in each switch. This is explained by the fact that the data transmission in a DCN makes the queue size information of switches related to each other. Hence, we implemented a Software-Defined DCN (SDDCN) so that the SDN controller is able to collect all the queue-size information of all the DCN OpenFlow-enabled switches.

As shown in Fig. 3, a conventional three-tier DCN topology that consists of edge tier (top-of-rack switches), the aggregation tier, and the core layer switches. Each rack is comprised of 20-40 servers. Two aggregation switches are connected to a server for backup. The aggregation switches give divers operations, such as firewalling, server load balancing, etc. The most important operation of the DCN playing the role of a switching backbone for incoming and outgoing data flows of the DCN is implemented by the core tier switches.

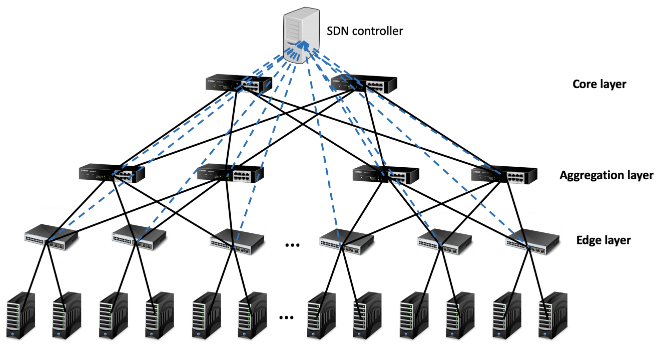

Fig. 3. Proposed three-tier Software-Defined DCN

Each switch in these three tiers is connected to a SDN controller. The latter performs the proposed alogirhtm, IBCN, that is comprised of 4 modules: Data collection, Training LSTM RNN, Congestion prediction, and Congestion reaction (Fig. 4).

The remainder of this section presents in details these four modules.

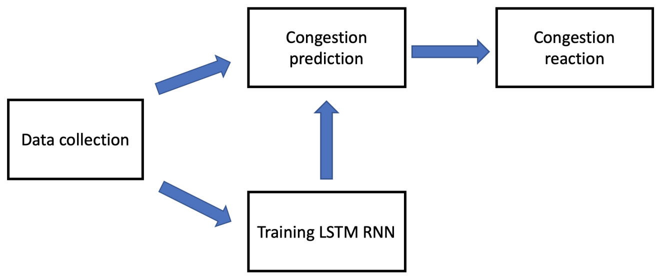

Fig. 4. Proposed IBCN protocol

A. Data collection module

The main goal of the *Data collection module* is to query, gather, and store all the statistics, especially the queue size information from all Openflow switches of the DCN. Concretely, this module periodically queries the statistics, including the queue size information, from all the switches with a fixed interval of 1s. After receiving the STATS_REQUEST message from the controller, the switch then replies with a STATS_REPLY message. The overhead generated by this module is not worth to consider because the size of the request and reply messages are 8 bytes and 104 bytes, respectively. The collected queue size information are then used by the *Training LSTM RNN module* and the *Congestion prediction module*.

B. Training LSTM RNN module

As aforementioned, the core idea of IBCN is to predict the queue size of switches in a SDDCN, thereby improving the accuracy of the congestion prediction of BCN algorithm. Concretely, the *Training LSTM RNN module* is performed by using the backpropagation through time algorithm (aka. a deep learning method) to learn the queue size characteristics from historical data flows that passes through the switches and predict its future queue size. Since the training method of LSTM and all of its variants is similar, only the general training method of LSTM is presented in this sub-section.

Consider a switch S, let $X = q(0), q(1), \ldots q(T)$ be the vector that represents the queue size of S at the moment t with $t = [0..T]$. The queue size prediction problem is defined as solving the predictor $\hat{Y} = q(T+1), q(T+2), \ldots q(T+k)$ via a series of historical and measured data set X, where k is the number of value of queue size we want to predict.

In order to forcefully feed the LSTM network, we don't predict one item $q(i+T+1)$ at a time by feeding the LSTM one item $q(i)$ at a time because the assumption that each item is independent from each other was proven to be wrong [21]. Therefore, in order to perform real-time prediction, the LSTM network needs to be fed and learned continuously. In addition, the total number of $q(i)$ increases obviously over time. To address this problem, we used the learning window mechanism. The latter fixes the window W for the number of $q(i)$. Concretely, we put together W values of queue size $(q(T), q(T-1), \ldots q(T-W-1))$ at a time.

To evaluate the performance and accuracy of our LSTM model, the Mean Square Error (MSE) is used to estimate the prediction accuracy. The scale dependent metric MSE is determined by the the difference between the estimated values and the actual ones. It is calculated by the the average sum of squared errors (Eq. 11).

$$MSE = \frac{1}{k}\sum_{i=1}^{k}(q(i) - \widehat{q(i)})^2 \tag{11}$$

where $q(i)$ and $\widehat{q(i)}$ are the observed value and the predicted value, respectively. k is the total number of predictions.

The output of this module is a trained LSTM RNN, which is used by the *Congestion prediction module*.

C. Congestion prediction module

The proposed IBCN predicts k number of queue size prediction samples represented by the vector $K = \{q(t+1), q(t+2), \ldots q(t+k)\}$. In basing on the original BCN algorithm, two thresholds are also fixed: Q_{eq} and Q_{sc}. Let n_a is the number of elements of K that are less than Q_{eq} (Eq. 12).

$$n_a = \{|K_a| \mid K_a \subset K \wedge j < Q_{eq} \forall j \in K_a\} \tag{12}$$

Let n_b is the number of elements of K that its value is between 2 thresholds (Eq. 13).

$$n_b = \{|K_b| \mid K_b \subset K \wedge Q_{eq} < j < Q_{sc} \forall j \in K_b\} \tag{13}$$

Let n_c is the number of elements of K that are greater than Q_{sc} (Eq. 14).

$$n_c = \{|K_c| \mid K_c \subset K \wedge j > Q_{sc} \forall j \in K_c\} \tag{14}$$

Basing on the original BCN algorithm, the feedback process of IBCN is implemented as follows:

1) If $n_a = \max(n_a, n_b, n_c)$:
    a) If the tag of the rate regulator doesn't exist: the switch doesn't send any feedback message.
    b) If the tag of the rate regulator exists: the switch sends back a positive BCN message.
2) If $n_b = \max(n_a, n_b, n_c)$: the switch sends back a BCN NORMAL message.
3) If $n_c = \max(n_a, n_b, n_c)$: the switch sends back a BCN STOP message.

We attach the value of the congestion measure e_{IBCN}, that is calculated by the Eq. 15, to every feedback message, so that *the Congestion reaction module* can use it to react appropriately.

$$e_{IBCN} = -(\frac{1}{k}\sum_{i=t+1}^{t+k} q(i) - Q_{eq} + \omega * (Q_a(t) - Q_d(t))) \tag{15}$$

D. Congestion reaction

This module is performed at the source servers to react to the received feedback messages. If it receives a STOP message, it stop sending packets for a for a random period from 200ms-800ms. If the source server receives a NORMAL message, it adjusts the sending rate as described in Eq. 16. In fact, we just replace the e_i in Eq. 2 with e_{IBCN} in Eq. 16.

$$r_i = \begin{cases} r_i + G_i e_{IBCN} R_u & \text{if } e_{IBCN} > 0 \\ r_i(1 + G_d e_{IBCN}) & \text{if } e_{IBCN} < 0 \end{cases} \qquad (16)$$

V. Experiments

For the experiments, *mininet* [22] is used to emulate the network with the conventional three-tier DCN topology (Fig. 3) that consists of 50 hosts, 5 top-of-rank (edge) switches, 2 aggregation switches, 1 core switches. We assign 10 servers per rack, each server in a rack is connected to a edge switch. Each edge switch is connected to 2 aggregation switches. The two aggregation switches, in its turn, are connected to the core switch. The 50 host nodes represent source servers and destination servers. In our topology, 2 hosts are chosen to be destination servers that receives packet flows from 48 other source servers. All links have the capacity of 1Gbs except the links connected to the core switch have the capacity of 10Gbs. The packet size is fixed to 64 bytes. The lightweight POX [23] is used as the SDN Controller. All the switches in the topology are connected to the POX controller. The IBCN parameters are fixed as shown in the Tab. I.

Parameter	R_u	G_i	G_d	W	Q_{sc}	Q_{eq}	k
Value	3Mbps	4	0.02	2	90	20	5

TABLE I
IBCN PARAMETERS

In order to generate the flows with any data bit-rate, we used the open source *Tcpreplay* tool [24]. The latter is usually used to edit and replay a previous captured network traffic with a fixed sending bit-rate. So, this tool helps us to just generate a few initial data flow and replay it after as many as we want.

For the training phase of LSTM and its variants, we run the *Tcpreplay* tool in each source node with a random data bit-rate that varies from 10Mbs to 1Gbs. The collecting dataset process takes 1 hour. Since the collection data module fixes the interval to 500ms, the obtained dataset matrix D has the size of (7200×8) elements. We split D into the matrix D_{train} and the matrix D_{test} of sizes (5760×8) and (1440×8). The D_{train} is used to train the RNN of LSTM and its variants, and the D_{test} is used to validate and test the accuracy.

First, the impact of the number of hidden layer of the RNN of LSTM and its variants on the MSE and the training time is evaluated. As shown in Fig. 5, generally, the deeper the network, the more accurate the prediction is. With the values of number of hidden layer from 1-4, the Classic LSTM and GRU outperform Skip-LSTM and Bi-Directional LSTM. Especially, the classic LSTM improves the average MSE of 39% of the Skip-LSTM. Besides, Classic LSTM also slightly improves 9% the MSE value of GRU. With the number of hidden layers from 6-8, the difference between algorithms is not clear.

As expected, the deeper the RNN, the longer it will take to train (Fig. 6). The obtained results show that, generally, the Classic LSTM takes the longest time for the training phase. The Skip-LSTM attains the best result because it skipped state updates in the computation graph. Its training time is only 49% of that of Classic LSTM at 1 hidden layer, and 63% at 8 hidden layers.

Basing on the experimental results of Figures 5 and 6, we set 4 hidden layers for the LSTM RNN to balance between the MSE and the training time.

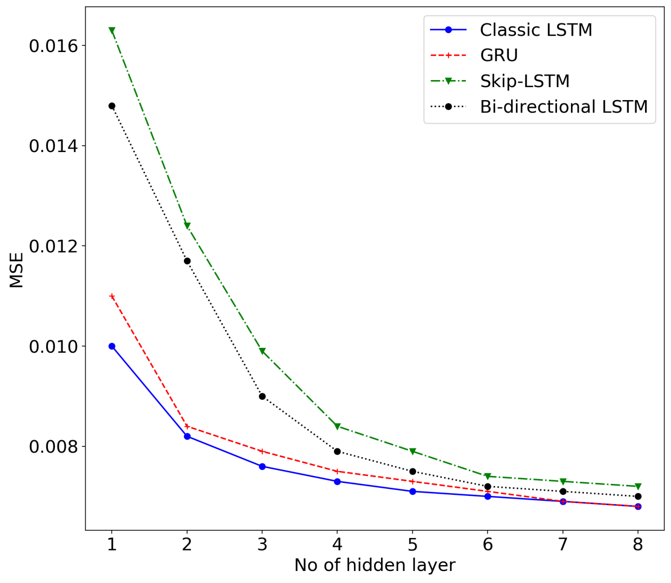

Fig. 5. MSE over the number of hidden layer

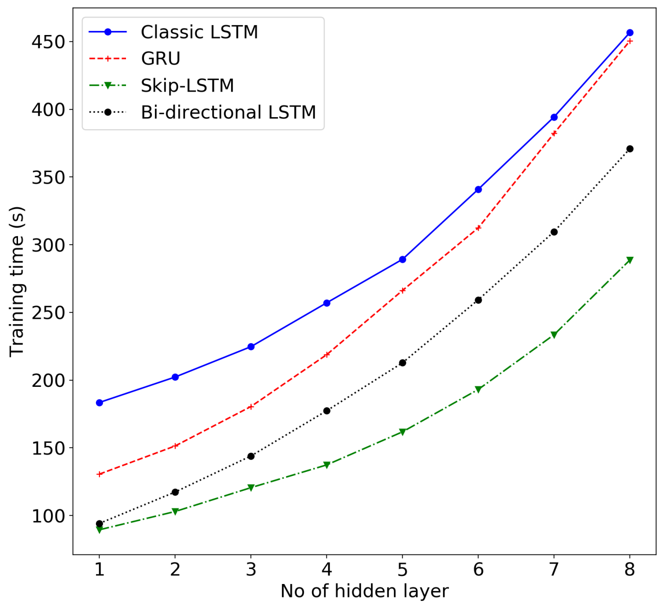

Fig. 6. Training time over the number of hidden layer

In order to prove the improvement of IBCN over the original BCN, two scenarios are created:

- Scenario 1 (Lightweight network): Each source nodes send out 10kB data and it stops for a random period (a few miliseconds), and it continue to transmit.
- Scenario 2 (Heavy network): Each source node sends data continuously to two destination nodes.

For the Scenario 1, we based on three parameters: the throughput in both transactions per second (Tps) and gigabit per second (Gbps), and the delay (μs) from a source node to one of two destination nodes. The obtained results in Tab. II showed that the BCN has well enhanced all these three parameters compared to results without any congestion management method (CM). The proposed IBCN-LSTM, as expected, give a better result than the original BCN also in three parameters. Concretely, it allows 1093 more transactions per second than BCN and significantly improves 15.5% in terms of throughput. The IBCN's delay is reduced from 215 to 196 μs. This improvement is explained by the capacity of predicting congestion of IBCN-LSTM, that helped the source nodes to react early to the congestion in the DCN. However, the IBCN-SkipLSTM and IBCN-BiLSTM have no significant improvement compared to the original BCN.

	TPS	Throughput (Gbps)	Delay (μs)
without CM	726	0.084	2038.49
BCN	7124	0.462	215.82
IBCN-LSTM	**8217**	**0.534**	**196.38**
IBCN-GRU	8193	0.507	205.73
IBCN-SkipLSTM	7131	0.468	211.93
IBCN-BiLSTM	7045	0.455	231.94

TABLE II
PERFORMANCE COMPARISON FOR SCENARIO 1 (LIGHTWEIGHT NETWORK)

For the Scenario 2 (Heavy network), we monitor a congested link (its link utilization is 99.9%) that connects the core switch to an aggregation one. As shown in Tab. III, even the BCN cannot give a better average throughput, the IBCN-LSTM slightly improve it 4.7%. In this scenario, we actively make the DCN heavier. Consequently, it caused large variance in the throughput of many bulk source nodes. Both the BCN and IBCN-LSTM are able to reduce this variance significantly. Especially, the IBCN-LSTM and IBCN-GRU reduce it down to 0.8% while BCN is 1.4%. The obtained results of IBCN-SkipLSTM and IBCN-BiLSTM are similar to the original BCN.

	Average Throughput	Standard deviation (%)
without CM	3.126	18.5
BCN	2.984	1.4
IBCN-LSTM	**3.273**	**0.8**
IBCN-GRU	3.058	**0.8**
IBCN-SkipLSTM	2.945	0.95
IBCN-BiLSTM	2.829	1.1

TABLE III
PERFORMANCE COMPARISON FOR SCENARIO 2 (HEAVY NETWORK)

We also try to evaluate the effectiveness of the congestion solution on different network load, which is varied from 10% to 90%. We used the Iperf tool [25] to vary the network load and collect the throughput results. The obtained results in the Fig. 7 showed that, with the load value from 10% to 60%, BCN and IBCN-LSTM and its variants give better throughput than the solution without any congestion management method. Especially, the IBCN-LSTM attained in average 33% better throughput than BCN. However, with the load value from 70% to 90%, the BCN, IBCN-SkipLSTM, and IBCN-BiLSTM do not have any improvement compared to the solution of using no CM. That shows the inefficiency of BCN, IBCN-SkipLSTM, and IBCN-BiLSTM in a high network load condition. Contrary to BCN, the IBCN-LSTM attained the enhancement of more than 50% in this high load condition.

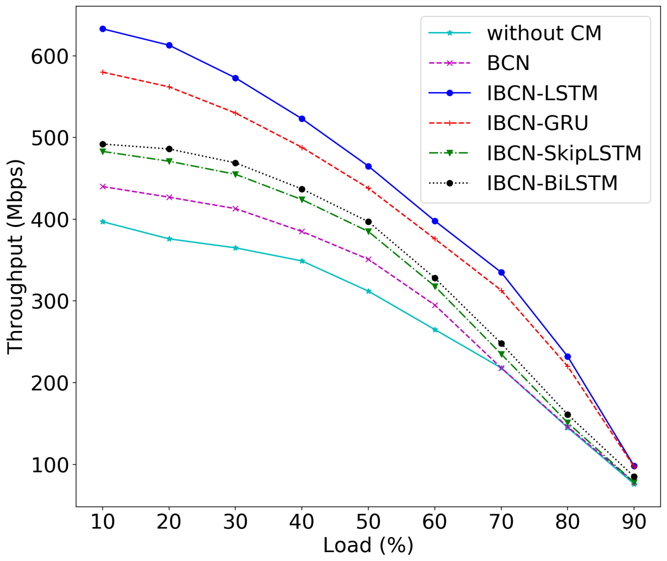

Fig. 7. Throughput at different load

In order to prove the advantage of using LSTM RNN for predicting congestion, we also compare the prediction error determined by the average difference $\frac{(q(i)-\widehat{q(i)})}{q(i)}$ in Eq. 11 of our proposed LSTM and its variants solutions with three traditional linear prediction algorithms: ARMA model [26], ARAR algorithm [27], [28], HoltWinters algorithm [29]. These three linear algorithms are also implemented in the original BCN algorithm. The Tab. IV shows that our LSTM approach outperforms the others. In general, the LSTM-based algorithms give better results than the traditional linear prediction algorithms.

Algorithms	Prediction error (%)
ARMA	41.3
ARAR	29.6
HotWinters	26.5
LSTM	**3.7**
GRU	4.1
SkipLSTM	11.7
BiLSTM	8.5

TABLE IV
PREDICTION ERROR

VI. Conclusion

In this paper, we address the problem of lack of ability to prediction congestion of Layer 2 congestion notification algorithms, such as BCN, FECN, E-FECN, and QCN, for a DCN. In order to resolve this issue, we proposed the IBCN algorithm in basing on the original BCN. Concretely, we have applied the LSTM RNN and its variants (GRU, Skip-LSTM, and Bi-Directional LSTM) to train the prediction model. To facilitate such application, we implemented it in a centralized control system in combining the SDN and DCN to construct a Software-defined DCN. That helps IBCN to predict queue size information of all switches in the implemented SDDCN. The obtained experimental results showed that the IBCN outperforms the original BCN at any network load levels of the SDDCN. Besides, the LSTM-based solutions give also better results than other traditional linear prediction algorithms. We believe these positive results have opened up a new trend in the application of ML to Layer 2 congestion notification algorithms to improve its performance and accuracy.

Acknowledgment

This research is funded by Vietnam National Foundation for Science and Technology Development (NAFOSTED) under grant number 102.02-2019.314

References

[1] Y. Zhang and N. Ansari, "On architecture design, congestion notification, tcp incast and power consumption in data centers," *IEEE Communications Surveys & Tutorials*, vol. 15, no. 1, pp. 39–64, 2012.

[2] B. Stephens, A. Cox, W. Felter, C. Dixon, and J. Carter, "Past: Scalable ethernet for data centers," in *Proceedings of the 8th international conference on Emerging networking experiments and technologies*, 2012, pp. 49–60.

[3] S. Bradner, "The internet engineering task force (ietf)," *DiBona et al.[144]*, pp. 47–52, 1999.

[4] J. Jiang, R. Jain, and C. So-In, "An explicit rate control framework for lossless ethernet operation," in *2008 ieee international conference on communications*. IEEE, 2008, pp. 5914–5918.

[5] C. So-In, R. Jain, and J. Jiang, "Enhanced forward explicit congestion notification (e-fecn) scheme for datacenter ethernet networks," in *2008 international symposium on performance evaluation of computer and telecommunication systems*. IEEE, 2008, pp. 542–546.

[6] M. Alizadeh, B. Atikoglu, A. Kabbani, A. Lakshmikantha, R. Pan, B. Prabhakar, and M. Seaman, "Data center transport mechanisms: Congestion control theory and ieee standardization," in *2008 46th Annual Allerton Conference on Communication, Control, and Computing*. IEEE, 2008, pp. 1270–1277.

[7] D. Bergamasco and R. Pan, "Backward congestion notification version 2.0," in *IEEE 802.1 Meeting*, 2005.

[8] Y. Wang, Q. Wang, S. Shi, X. He, Z. Tang, K. Zhao, and X. Chu, "Benchmarking the performance and power of ai accelerators for ai training," *arXiv preprint arXiv:1909.06842*, 2019.

[9] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2014.

[10] A. Pages, F. Agraz, R. Montero, G. Landi, R. Monno, J. I. Aznar, A. Vines, C. Jackson, D. Simeonidou, and S. Spadaro, "Experimental assessment of vdc provisioning in sdn/openstack-based dc infrastructures with optical dcn," in *ECOC 2016; 42nd European Conference on Optical Communication*. VDE, 2016, pp. 1–3.

[11] J. Pang, G. Xu, and X. Fu, "Sdn-based data center networking with collaboration of multipath tcp and segment routing," *IEEE Access*, vol. 5, pp. 9764–9773, 2017.

[12] G. Xu, B. Dai, B. Huang, J. Yang, and S. Wen, "Bandwidth-aware energy efficient flow scheduling with sdn in data center networks," *Future Generation computer systems*, vol. 68, pp. 163–174, 2017.

[13] X. Xue, F. Wang, F. Agraz, A. Pagès, B. Pan, F. Yan, S. Spadaro, and N. Calabretta, "Experimental assessment of sdn-enabled reconfigurable opsquare data center networks with qos guarantees," in *2019 Optical Fiber Communications Conference and Exhibition (OFC)*. IEEE, 2019, pp. 1–3.

[14] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[15] V. Campos, B. Jou, X. Giró-i Nieto, J. Torres, and S.-F. Chang, "Skip rnn: Learning to skip state updates in recurrent neural networks," *arXiv preprint arXiv:1708.06834*, 2017.

[16] M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks," *IEEE transactions on Signal Processing*, vol. 45, no. 11, pp. 2673–2681, 1997.

[17] M. Gholami and B. Akbari, "Congestion control in software defined data center networks through flow rerouting," in *2015 23rd Iranian Conference on Electrical Engineering*. IEEE, 2015, pp. 654–657.

[18] A. Majidi, N. Jahanbakhsh, X. Gao, J. Zheng, and G. Chen, "Dc-ecn: A machine-learning based dynamic threshold control scheme for ecn marking in dcn," *Computer Communications*, vol. 150, pp. 334–345, 2020.

[19] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.

[20] ——, "Gated feedback recurrent neural networks," in *International conference on machine learning*, 2015, pp. 2067–2075.

[21] Y. Wen and G. Zhu, "Prediction for non-gaussian self-similar traffic with neural network," in *2006 6th World Congress on Intelligent Control and Automation*, vol. 1. IEEE, 2006, pp. 4224–4228.

[22] R. L. S. De Oliveira, C. M. Schweitzer, A. A. Shinoda, and L. R. Prete, "Using mininet for emulation and prototyping software-defined networks," in *2014 IEEE Colombian Conference on Communications and Computing (COLCOM)*. IEEE, 2014, pp. 1–6.

[23] S. Kaur, J. Singh, and N. S. Ghumman, "Network programmability using pox controller," in *International Conference on Communication, Computing & Systems (ICCCN'2014)*, 2014, pp. 134–138.

[24] F. Klassen, *TcpReplay*, 2009, https://tcpreplay.appneta.com.

[25] A. Tirumala, "Iperf: The tcp/udp bandwidth measurement tool," *http://dast. nlanr. net/Projects/Iperf/*, 1999.

[26] B. Choi, *ARMA model identification*. Springer Science & Business Media, 2012.

[27] F.-L. Chu, "Analyzing and forecasting tourism demand with arar algorithm," *Tourism Management*, vol. 29, no. 6, pp. 1185–1196, 2008.

[28] N. Miswan, N. Hussin, R. Said, K. Hamzah, and E. Z. Ahmad, "Arar algorithm in forecasting electricity load demand in malaysia," *Global Journal of Pure and Applied Mathematics*, vol. 12, no. 1, pp. 361–367, 2016.

[29] J. Ekberg, J. Ylinen, and P. Loula, "Network behaviour anomaly detection using holt-winters algorithm," in *2011 International Conference for Internet Technology and Secured Transactions*. IEEE, 2011, pp. 627–631.