Towards a Cognitive Routing Engine for Software Defined Networks

Frederic Francois and Erol Gelenbe
Intelligent Systems and Networks
Department of Electrical and Electronic Engineering
Imperial College, London SW7 2BT, UK
Email: {f.francois and e.gelenbe}@imperial.ac.uk

Abstract—Most Software Defined Networks (SDN) traffic engineering applications use excessive and frequent global monitoring in order to find the optimal Quality-of-Service (QoS) paths for the current state of the network. In this work, we present the motivations, architecture and initial evaluation of a SDN application called Cognitive Routing Engine (CRE) which is able to find near-optimal paths for a user-specified QoS while using a very small monitoring overhead compared to global monitoring which is required to guarantee that optimal paths are found. Smaller monitoring overheads bring the advantage of smaller response time for the SDN controllers and switches. The initial evaluation of CRE on a SDN representation of the GEANT academic network shows that it is possible to find near-optimal paths with a small optimality gap of 1.65% while using 9.5 times less monitoring.

I. INTRODUCTION

The ossification of computer networks due to the use of non-standardized and closed source protocols between the control and data plane of commercial Network Forwarding Elements (NFEs) has led researchers to develop Software Defined Networks (SDNs) [1] where open and standardized protocols, such as OpenFlow (OF) [2], are used to program the data plane of NFEs. In order to route traffic, SDNs usually make use of routing applications which runs on top of the SDN controller. These routing applications make routing decisions based on network policies and the state of the network but unfortunately, gathering the state of the network is an expensive activity both in terms of processing overhead at the controller and NFEs and control traffic [3]. With this in mind, we develop a new routing application called Cognitive Routing Engine (CRE) which significantly increases the efficiency of the network state gathering process while obtaining enough information about the network to calculate the best paths that meet the Quality of Service (QoS) requirements of the host applications which use them. The design objectives of CRE are:

- the state of the network should be gathered in an efficient way, i.e. a minimum number of OF messages should be exchanged between the SDN controller and the NFEs during the gathering process since each OF message has a processing overhead and occupies bandwidth on the control network of the SDN;
- the optimization of the network monitoring process should not lead to worse paths being found by the CRE SDN application;
- the use of CRE should not lead to frequent changes in the path chosen by CRE for a given traffic flow, i.e. no frequent oscillations of the path for a given flow by CRE which can increase the control plane processing and bandwidth overhead inside SDNs;
- updates to the flow tables of the NFEs must not have an adverse impact on the traffic being transported by the NFEs. For e.g., no packet loss and forwarding loops when flow tables are updated; and
- the implementation of CRE should be compatible with existing SDN-enabled NFEs so as to enable more network operators to adopt CRE without purchasing new hardware and software, and avoid lengthy standardization processes.

II. RELATED WORK

In current literature, most traffic engineering techniques [4] used in SDN takes the view that SDN provides a global view of the current network state and topology in a logically centralized controller and therefore, optimization can be performed on the network by running different types of global traffic engineering algorithms such as constrained shortest path first. In this paper, we take the view that networks can be large and therefore, it is inefficient to obtain accurate and updated state information for the whole network at a frequency required for effective traffic optimization. This view is supported by the large amount of work that is currently undertaken to reduce the excessive overhead linked with accurate and frequent global monitoring the network [3]. Existing work on network monitoring in SDN can be divided into 2 categories: active and passive monitoring [3]. Active monitoring involves either the actual probing of the network by sending special packets [5] and/or polling the state of the network through OF mechanism, e.g. retrieving the value of OF counters [2], while passive monitoring either only observes the existing SDN behaviour to infer the state of the network, e.g. Packet-In and Flow Removed OF control messages [6], or calculates the network state based on the collected sampled packets [7].

The Cognitive Routing Engine developed in this paper is similar to the routing algorithm that is used in Cognitive Packet Networks (CPNs) [8]–[11] but with the significant and
important differences that in CPN, each router runs its own routing and learning algorithm for every source-destination pair and QoS requirement and, the routers collect the network state through the use of smart packets and their associated acknowledgment packets. This is in contrast with CRE which runs in a logically centralized manner on top of the SDN controller and uses exclusively OF mechanisms to gather network state. Hence, CRE can be viewed as a SDN-compatible version of CPN and can be used with the large number of SDN-enabled switches that has already been deployed. CPN has been used successful in various settings [12], for e.g., traffic engineering [13], routing in wireless [14] and sensor [15] networks and defence against Denial-of-Service (DoS) attacks [16].

The main contribution of this paper is a SDN-compatible CPN application which reduces the amount of network monitoring while maintaining the ability to discover paths which meet the QoS requirements of host applications.

III. OVERALL ARCHITECTURE

Fig. 1 shows where the CRE application is located in the overall SDN architecture and how it interacts with the various other components of a typical SDN deployment.

A brief description of each component of the overall SDN architecture is provided as follows:

- **Network Forwarding Elements (NFE)—are SDN-enabled packet switches and use OpenFlow v1.3 [2] as the communication protocol between the control and data plane of the NFEs in this particular instance.**

- **SDN Controller—a.k.a Network Operating System, is responsible to send and receive OF messages from the NFEs. In addition, the SDN controller typically parses the received OF messages into data structures which can be understood by the different applications running on top of the controller.**

- **Applications—are pieces of software running on top of the controller which provides specialized network functions such as Network Address Transation (NAT) and firewall. In this particular instance, there are 2 applications:**
  - Link Discovery application—which discovers the data plane topology of the network by using the standardized Link Layer Discovery Protocol (LLDP) [17]. A link is defined in this paper as unidirectional.
  - CRE application—which efficiently finds and installs new suitable paths in the network as requested by other SDN applications. CRE is made of 3 main modules:
    - the Cognitive Routing Algorithm Module (CRAM)—which uses Random Neural Networks (RNNs) with Reinforcement Learning (RL) to find network paths which maximize a customizable objective function and therefore, meet the QoS requirements of host applications,
    - a Network Monitoring Module (NMM)—which efficiently either uses past network measurements and/or probes and/or poll the network to get the necessary network state information to update the RNNs in the CRAM,
    - a Path-to-OF Translator Module (PTM)—which is able to convert the paths found by the CRAM into the appropriate set of OF messages so that paths are either created or updated with minimum inconsistency in the network.

IV. MESSAGE EXCHANGE SEQUENCE BETWEEN DIFFERENT COMPONENTS OF SDN ARCHITECTURE

Fig. 2 provides an example of the sequence of message exchange between the different components of the SDN architecture when a new flow arrives at the network and paths need to be set up to route the flow through the SDN according to the network policies set by the network operator. A flow in the electrical domain is considered by OF as a collection of packets where the value of a subset of the layer 2 to 4 fields are the same and a new flow is one where there is no rule which matches the flow inside the flow table(s) of the first encountered NFE.

The different steps in Fig. 2 are described below:
- Step 1: A new flow $F_{1,3}$ arrives at the NFE $N_1$ which needs to travel to NFE $N_3$.
- Step 2: NFE $N_1$ analyzes the first packet of the flow $F_{1,3}$ and finds that there is no rule inside its flow table(s) which matches the packet. Therefore, $N_1$ encapsulates the packet into an OF Packet-In message and sends the message to its master SDN controller since the default action for a table-miss event was set to forward the packet to the master controller during the configuration of the NFE.
- Step 3: The master SDN controller receives the OF Packet-In message and parses it into a data structure more suitable for processing by different applications running on top of the SDN controller. The SDN controller notifies and sends the data structure to all SDN applications which have registered with it to receive the OF Packet-In messages.
- Step 4: Notified applications decide if they are interested to perform actions based on the data structure that is sent by the controller. In this particular scenario, the CRE application is the only relevant application which will perform actions due to the new flow $F_{1,3}$.
- Step 5 & 6: CRE first installs a path for flow $F_{1,3}$ by calculating the shortest path based on hop count between NFE $N_1$ and $N_2$ by using the network topology discovered by the Link Discovery application. This allows packets to be routed in the network fast without waiting for CRE to collect network measurements and find a path based on RNN which can lead to packet loss. The PTM module of CRE is responsible to implement the new path for flow $F_{1,3}$ in the NFEs by installing the appropriate OF rules as will be described in Section VII. Next, CRAM finds the most suitable links for a given path request by using RNNs with RL. In order for CRAM to operate, it needs to gather information about the network state, which is the responsibility of the NMM in CRE. The detailed description of how CRAM operates will be given in the following Section V.
- Step 6, 7 & 8: The NMM receives requests from the CRAM for certain characteristics of links in the network so that CRAM can calculate the numeric value of the objective function of the whole path that it has chosen and also updates its RNNs so that it chooses better paths in the future. Currently, NMM can obtain utilization, quality (based on packet loss and frame errors) and delay of the links of the network. NMM can retrieve and calculate link utilization and quality of a given link by using the OF Port Stats Request messages. On the other hand, OF doesn’t have any native ability to obtain the necessary information from the network in order to calculate the packet delay on a link but the SDN controller can send probe packets to obtain a good estimate of packet delays [5]. In addition, NMM is intelligent enough that it does not always query the network for new data but it can use its database of recently gathered data to reply to the requests made by CRAM. This increases the efficiency of the network monitoring process for the whole CRE.
- Step 9: CRAM updates its RNNs through reinforcement learning based on the data received from the NMM and will output the best paths found. If paths need to be updated, they are then passed to the PTM.
- Step 10: The PTM is responsible to implement the paths in the NFEs by modifying the flow table(s) of the NFEs in a way which will not lead to packet loss and forwarding loops in the network. This is explained in detail in Section VII.

V. CRAM: COGNITIVE ROUTING ALGORITHM MODULE

CRAM is a routing algorithm which is based on Random Neural Network (RNN) with Reinforcement Learning (RL) [8]–[11]. For a given flow, a new RNN is created for each NFE along the path initially found through shortest path routing. A RNN consists of a network of interconnected neurons where each neuron represents one active port of the NFE. The decision of the RNN on which output to use as the next hop can be either exploratory or exploitation. When the RNN is in exploratory mode, it chooses the output port randomly while when the RNN is in exploitation mode, it chooses the neuron which has the highest potential as will be described in the next paragraph. Exploratory mode is chosen with a probability $X\%$, e.g. 5% [12], of the time and exploitation mode is chosen for the remaining $(1-X)\%$ of the time. If the next hop NFE does not currently have a representative RNN for this particular flow, a new RNN is created for it.

Each neuron has a property called potential and the neuron with the highest potential is selected as the output when the RNN is in exploitation mode and therefore, the most excited neuron determines the next hop NFE of the path. The potential, $q_i$, of neuron $i$ can be calculated by using:

$$q_i = \frac{\lambda^+_i}{r_i + \lambda^-_i}$$

(1)

where $\lambda^+_i$ and $\lambda^-_i$ are respectively the total positive and negative potential that are transferred from other neurons connected to neuron $i$ in the RNN. These total potentials are dependent on both the potential of the neurons themselves and the weights of the links and can be calculated as:

$$\lambda^+_i = \sum_{j \in N} q_j w^+_{j,i} + \Lambda^+_i \quad \text{where } j \neq i$$

(2)

$$\lambda^-_i = \sum_{j \in N} q_j w^-_{j,i} + \Lambda^-_i \quad \text{where } j \neq i$$

(3)

where $w^+_{j,i}$ and $w^-_{j,i}$ are respectively the positive and negative weight of the link from the other neurons $j$ in the RNN to the neuron $i$ and $N$ is the set of all neurons in the RNN. $\Lambda^+_i$ and $\Lambda^-_i$ are the fixed rate at which external positive and negative spikes arrives at the neuron respectively. $r_i$ is a normalization factor (firing rate in [12]) which is calculated as:

$$r_i = \sum_{j \in N} [w^+_{i,j} + w^-_{i,j}] \quad \text{where } j \neq i$$

(4)
The positive \( w_{i,j}^+ \) and negative \( w_{i,j}^- \) weight of each link are determined with the help of reinforcement learning. The current exponential averaged value of the objective function \( O_{sd}[t] \) of a path with source \( s \) and destination \( d \) is calculated as follows:

\[
O_{sd}[t] = \alpha O_{sd}[t-1] + (1 - \alpha)(o_{sd}[t]) \quad \text{where } 0 < \alpha < 1 \quad (4)
\]

where \( O_{sd}[t-1] \) is the previous value of the objective function, \( o_{sd}[t] \) is the current value of the objective function and \( \alpha \) is the exponential average parameter which determines the relative importance of the current value of the objective function compared to the previous values. If the objective function is to be minimized, the reward \( R_{sd}[t] \) of the path is calculated as \( R_{sd}[t] = (O_{sd}[t])^{-1} \). The effect of \( R_{sd}[t] \) on the link weights in all RNNs along the found path depends on whether \( R_{sd}[t] \) is greater or less than a threshold \( \gamma \) with \( \gamma_{sd}[t] \) calculated as follows:

\[
\gamma_{sd}[t] = \beta \gamma_{sd}[t-1] + (1 - \beta) R_{sd}[t] \quad \text{where } 0 < \beta < 1 \quad (5)
\]

where \( \beta \) is the exponential average parameter.

If \( R_{sd}[t] \geq \gamma_{sd}[t] \), i.e. the reward of the new path is greater than the threshold and therefore the RNNs have made the right decision, the link weights in each RNN is updated as follows:

\[
w_{j,i}^+ = w_{j,i}^+ + R_{sd}[t] \quad \text{and} \quad w_{j,k}^- = w_{j,k}^- - \frac{R_{sd}[t]}{\gamma_{sd}[t]} \quad \text{where } j \in N \text{ and } k \neq i.
\]

In the first link weight update operation, the positive link weights from other neurons \( j \) to the chosen neuron \( i \) are increased so that the potential \( q_i \) of neuron \( i \) is increased and the port associated with neuron \( i \) is chosen again as output for the next search if the RNN is in exploitation mode. In the second link weight update operation, the negative link weights from neurons \( j \) in the RNN to other neurons other than \( i \) are then increased so that their potential is decreased with as end result the reduction of the probability that their associated ports will be chosen as output if the RNN is in exploitation mode.

On the other hand, if \( R_{sd}[t] \leq \gamma_{sd}[t] \), i.e. the reward of the new path is less than the threshold and therefore the RNNs have made the wrong decision, the link weights in each RNN is updated as follows:

\[
w_{j,i}^+ = w_{j,i}^+ + R_{sd}[t] \quad \text{and} \quad w_{j,k}^- = w_{j,k}^- + \frac{R_{sd}[t]}{\gamma_{sd}[t]} \quad \text{where } k \neq i.
\]

In the first case, the positive link weights from neurons \( j \) in the RNN to other neurons other than \( i \) are increased so that their potential is increased and it is more likely that one of them will have its associated port chosen as the output during the next search rather than the previously chosen associated port of neuron \( i \). In the second case, the negative link weights from other neurons \( j \) to the chosen neuron \( i \) are increased so that the potential \( q_i \) of neuron \( i \) is decreased with as end result the reduction of the probability that the port associated with \( i \) will be chosen as output during the next search.

In order to prevent the link weights \( W \) to increase indefinitely, they are re-normalized by first calculating the new \( r_i^+ \) by using the updated \( W \) values in eq. (3) and updating the link weights as follows:

\[
W = W NMM \quad (3)
\]

Furthermore, CRAM keeps a record of the best \( Z \) recent paths found for each path request and the best path out of the \( Z \) paths is used for the actual routing of the traffic flow if the existing path has been active for longer than minimum time period to avoid frequent path fluctuations.

VI. NMM: Network Monitoring Module

The Network Monitoring Module (NMM) is responsible for providing to the CRAM and external SDN applications the characteristics of the links that they request. The link characteristics that can currently be monitored and calculated by NMM are: utilization, quality (based on packet loss and frame errors) and delay. It should be noted that NMM cannot directly measure these characteristics but can obtain them after performing operations on measurement information provided by the NFEs and sending probe packets if necessary. The NMM is made of the following components:

- **Northbound API**—allows the NMM to interact with the CRAM and external SDN applications. The characteristics of links can be requested to the NMM and the NMM can also receive information from other SDN applications, for e.g., the links which are active in the network can be received from the Link Discovery application [17].

- **NFEs Database**—stores information about the fixed characteristics of NFEs. During the initial contact between the NFEs and the SDN controller, the NFEs send information about their identity and the features that they support by sending OF Features Reply messages [2] (it should be noted that the port description in OF v1.3 [2] is now obtained through the use of OF Port Description Request/Reply messages). The following information are stored: NFE Data Plane Identifier (DPID), and the port number, MAC address and speed (as negotiated between 2 adjacent NFEs) of the different ports of the NFEs.

- **Links Database**—stores information about the characteristics of each unidirectional link in the SDN. There are 2 types of characteristics stored about the links: fixed and non-fixed. Fixed characteristics of the links are: source NFE DPID \( s \), destination NFE DPID \( d \), source port number at \( s \), destination port number at \( d \) and capacity \( b \). The non-fixed characteristics are: operational status, utilization, quality and delay. In addition, each non-fixed characteristic of the link has an associated timestamp field which stores the time when the characteristic was last updated.

- **Link Delay Monitoring**—sends and receives probe packets in the network in order to measure the delay of a link in a similar way as in [18]. The detailed procedure will be described below.

- **OF Statistics Gathering**—send (resp. receive) OF Port Stats Requests (resp. Replies) to calculate the utilization and quality of a link. The detailed procedure will be described below.

- **Southbound API**—allows the NMM to interact with the SDN controller in order to send and receive OF statistics messages as well as probing packets for delay measurements.
A. Link Delay Monitoring

A Link Delay Monitoring mechanism based on using only the features that OF [2] provides so that CRE is compatible with OF-enabled NFEs which are already deployed in the field. Since the OF protocol as of [2] does not specify that an OF-compatible NFE can measure and store link delay by itself, it is necessary to make use of probe packets in order to measure and calculate the delay of the link between 2 NFEs. The Link Delay Monitoring mechanism follows the following steps to calculate the link delay between 2 NFEs $N_s$ and $N_d$ with source and destination port $P_s$ and $P_d$ respectively:

- **Step 1:** An OF Echo Request message (with a specified transaction id $xid$) along with an OF Barrier Request message is sent to NFE $N_s$ and the time $T_{N_s}$ it takes for $N_s$ to reply back with a corresponding OF Echo Reply (identified by the $xid$ value) is measured. The same is done for $N_d$.
- **Step 2:** The SDN controller is instructed to send an OF Packet Out message to NFE $N_s$ where the source and destination MAC of the packet is set to the MAC address of the port $P_s$ and port $P_d$ respectively. The MAC addresses are obtained by using an OF Port Description Request message in OF v1.3 [2] during the population of the NFEs Database. The Packet Out message also contains the single action of outputing the packet on the port $P_s$. The Ethernet protocol type of the packet is set to an arbitrary value, e.g. 0x07c3, so that the SDN controller can identify the packet as a probe packet. The time $T_i$ that it takes for the SDN controller to receive the probe packet from $N_d$ is measured. The delay $D_{sd}$ between $N_s$ and $N_d$ is calculated as $D_{sd} = T_i - \frac{T_{N_s}}{2} - \frac{T_{N_d}}{2}$.
- **Step 3:** The NMM will instruct the SDN controller to periodically send probe packets as in step 2 above if the link continues to be selected by any RNN for probing. $T_{N_s}$ and $T_{N_d}$ are periodically measured also since these may change during the network operation.

B. OF Statistics Gathering

SDN-enabled NFEs have counters [2] which can count different port characteristics such as sent and received packet and number of received packet with errors per port. This information is retrieved on either a per NFE-basis or port-basis by the SDN controller by sending an OF Port Stats Request message to the relevant NFE with the port number specified if retrieving information on a per-port basis.

To retrieve the following characteristic of a link:

- **Utilization**—is calculated by first obtaining the number of transmitted bytes $A$ being sent over a link by sending to the source NFE of the link an OF Port Stats Request message specifying the port number of the source port of the link so that only statistics about this port is retrieved and not for all the ports of the NFE [2]. The utilization $U$ of a link can be calculated as:

$$U = \frac{A[t] - A[t-1]}{\delta b} \quad (6)$$

where $\delta$ is the time period between the 2 times at which the number of transmitted bytes $A$ is polled from the NFE. $\delta$ can be adjusted according to the accuracy and overhead required, a smaller $\delta$ will lead to a more accurate $U$ but more overhead in terms of processing and bandwidth used by the control messages.

- **Quality**—(based on packet loss and frame errors) is calculated by first obtaining the statistics of the source $(s, i)$ and destination $(d, i)$ ports of the link in the same manner as above for utilization. The link quality $Q$ can be calculated by using:

$$Q = (A_{s,i} - A_{\text{Dropped}} - A_{\text{Error}} - R_{d,i}) + (R_{d,i} + D_{d,i})$$

where $A_{s,i}$, $A_{\text{Dropped}}$ and $A_{\text{Error}}$ are respectively the total no. of packets transmitted, dropped and containing errors on the transmission pipeline of port $i$ of source NFE $s$. $R_{d,i}$, $D_{d,i}$ and $R_{d,i}$ are respectively the total no. of packets received, dropped and containing errors on the reception pipeline of port $i$ of destination NFE $d$.

C. Optimizing the Network Monitoring Process

The network monitoring process can be optimized through the following:

- the monitoring information is shared between the different RNNs by re-using the monitoring information already in the monitoring databases if it is recent. This is different to traditional CPN [11] where each NFE will send smart packets for each of its RNNs to collect network state. This allows CRE to have a worst-case/upper bound on its network monitoring activity that is the same as when the current global SDN traffic engineering applications [4] poll every NFEs in the network. Moreover, CRE runs a global and optimal routing algorithm when it finds itself in a scenario where it is monitoring the whole network.
- the NMM will not repeat the network monitoring if last monitoring was done less than threshold time $t_e$ ago. $t_e$ can be seen as a trade-off between having updated network state and overhead associated with network monitoring. In addition, $t_e$ is an adjustable parameter that can be optimized for e.g., depending on the rate of change of the link characteristic. If the link characteristic is stable, then a longer $t_e$ can be used [19].
- OF statistics are stored in the database and reused if appropriate and needed. For e.g., 1 RNN may request the NMM to provide the quality of a link and shortly after, another RNN may request for the utilization of the same link. Since OF returns all the port counters after an OF Port Stats Request message [2], the NMM may have the updated information already in its database without having to poll the involved NFE(s) again.

VII. PTM: PATH-TO-OF TRANSLATOR MODULE

The main objective of the Path-to-OF Translator Module (PTM) is to install and/or update OF flow rules in the network
so that the traffic flow follows the path chosen by CRAM without any packet loss, forwarding loops and unnecessary Packet-In messages being sent to the SDN controller. PTM operates in the following manner if the path is made of 2 or more NFES:

- New path—OF rules are inserted at the NFES by starting with the last NF of the path to the first NF so that Packet-In are not triggered by the NFES for flows for which CRAM has already calculated a path. For the last NF, a rule is added which matches the original flow with the addition of VLAN ID 1. The actions of the rule is to remove the VLAN tag and output the packet to the destination port. Next, the rule with the same match as the previous rule is added to all intermediate NFES except the first NF of the path. The action of the rule is to output to the relevant port as selected by CRAM. Finally, for the first NF of the path, the rule is to match the orginal flow only with actions being tag the flow packets with VLAN ID 1 with action of forwarding the flow to the output port connected to the next NF of the path.

- Change existing path—OF rules are inserted for all NFES except the first NF as the previous case with the difference that the VLAN ID used is 2 if 1 was previously used for the flow and 1 if 2 was used. The use of VLAN tags is used to avoid conflict between the old and new rules for the same flow. The OF rule for the flow in the first switch is modified so that now it tags the flow packets with the new VLAN ID and outputs the packet to the port connected to the next NF of the new path. Finally, the old rules in the intermediate and last NF of the old path are deleted.

It should be noted that if the source and destination of the flow is directly connected to the same NF, CRE does not need to calculate a path, it just inserts a rule which matches the flow with the single action of outputing the flow packets to the port connected to the destination NF.

In order to automatically remove rules for flows with no traffic for the last $t_1$ seconds, the *idle_timeout* is set to $t_1$ for all the rules inserted in the NF. The *hard_timeout* is set to 0 so that the network always has a path for a current active flow even if the controller becomes disconnected.

### VIII. Evaluation

The following evaluations scenarios where carried out using the SDN emulator Mininet [20] where custom topology can be deployed with custom delay, bandwidth, packet loss and queue length for each link.

#### A. Scenario 1: Delay Detection and Path Switching in illustrative network

The aim of scenario 1 is to demonstrate CRE finding, monitoring and switching paths in a network where the delay on links can vary. Scenario 1 uses the topology in Fig. 1 where the initial link delays are shown in the figure. When Ping #1 enters the network with source 1 and destination 2, CRE first installs 2 paths $G_{1,2} = H_1 \rightarrow N_1 \rightarrow N_2 \rightarrow H_2$ and $G_{2,1} = H_2 \rightarrow N_2 \rightarrow N_1 \rightarrow H_1$ based on shortest path based on hop count which also coincidently gives the paths with the shortest delay with the Ping Round-Trip Time (RTT) of 120ms, the higher RTT for Ping #1 compared to subsequent Pings are due to the overhead of notifying the controller of new flows, calculating paths and inserting rules inside the NFES.

Then CRE starts to monitor the network based on RNN with RL. At time $9s$ in the experiment, the links from/to $N_1$ to/from $N_2$ $L_{1,2}$ and $L_{2,1}$ suffers from increased delay from 25ms to 200ms resulting in the Ping RTT to increase to 420ms. CRE detects this increase in path delays through its monitoring of the network, the first path that is changed is $G_{2,1}$ where an alternative path $G_{2,1} = H_2 \rightarrow N_2 \rightarrow N_3 \rightarrow N_1 \rightarrow H_1$ is found which reduces the Ping RTT to 280ms at Ping #18. Furthermore, the path $G_{1,2}$ is changed to the $G_{1,2} = H_1 \rightarrow N_1 \rightarrow N_3 \rightarrow N_2 \rightarrow H_2$ with the RTT becoming 140ms at Ping #20. These two new paths found by CRE are the optimal one for the new network conditions.

#### B. Scenario 2: CRE convergence and monitoring reduction in the GEANT operational network

The network topology used is GEANT—a European academic network which is made up of 23 Point of Presence (PoP), represented as NFES in this experiment, and 74 unidirectional links, where the propagation delay of given link is calculated using the line-of-sight distance between the source and destination PoP of the link. The experiment objective is to measure the amount of time and monitoring probes that CRE takes to discover the best path in the network for a given Source-Destination (SD) pair compared to an optimal routing algorithm which probes all the 74 links each time that it probes the network. The experiments are run for 90s with the network being loaded around 20s into the experiment. The monitoring frequency of both CRE and the optimal algorithm is set to 5s.
Fig. 4 shows the increase in Ping RTT around 20s when the network is loaded and the decrease in RTT when CRE finds iteratively better paths to route the flow.

Table I shows that CRE can reduce by up to 9.5 times the amount of monitoring required to find a best path which is iteratively better paths to route the flow.

**IX. CONCLUSIONS AND FUTURE WORK**

In this work, we show that CRE is able to find paths which are very close to the optimal ones without incurring a large monitoring overhead. CRE is able to achieve this performance without requiring any modification to already-deployed SDNs which run OpenFlow.

It is our intention to further improve the evaluation section of this work by carrying more sophisticated experiments with different CRE parameters, network topologies, traffic conditions, QoS metrics and network events such as link and node failures.

**REFERENCES**

[1] N. Feamster, J. Rexford, and E. Zegura, “The road to sdn,” *Queue*, vol. 11, no. 12, pp. 20:20–20:40, Dec. 2013.

[2] (2015, March) Openflow switch specification, version 1.3.5. Open Networking Foundation. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/ onf-specifications/openflow/openflow-switch-v1.3.5.pdf.

[3] A. Yassine, H. Rahimi, and S. Shirmohammadi, “Software defined network traffic measurement: Current trends and challenges,” *Instrumentation Measurement Magazine, IEEE*, vol. 18, no. 2, pp. 42–50, April 2015.

[4] I. F. Akyildiz, A. Lee, P. Wang, M. Luo, and W. Chou, “A roadmap for traffic engineering in sdn-openflow networks,” *Computer Networks*, vol. 71, no. 0, pp. 1 – 30, 2014.

[5] K. Phemius and M. Bouet, “Monitoring latency with openflow,” in *Network and Service Management (CNSM), 2013 9th International Conference on*, Oct 2013, pp. 122–125.

[6] C. Yu, C. Lumezanu, Y. Zhang, V. Singh, G. Jiang, and H. Madhyastha, “Flowense: Monitoring network utilization with zero measurement cost,” in *Passive and Active Measurement*, ser. Lecture Notes in Computer Science, M. Roughan and R. Chang, Eds. Springer Berlin Heidelberg, 2013, vol. 7799, pp. 31–41.

[7] J. Suh, T. Kwon, C. Dixon, W. Felter, and J. Carter, “Opensample: A low-latency, sampling-based measurement platform for commodity sdn,” in *Distributed Computing Systems (ICDCS), 2014 IEEE 34th International Conference on*, June 2014, pp. 228–237.

[8] E. Gelenbe, Z. Xu, and E. Seref, “Cognitive packet networks,” in *Tools with Artificial Intelligence, 1999. Proceedings. 11th IEEE International Conference on*, 1999, pp. 47–54.

[9] E. Gelenbe, “Cognitive packet network,” US Patent 09/680,184, October 12, 2004.

[10] E. Gelenbe, R. Lent, and Z. Xu, “Design and performance of cognitive packet networks,” *Perform. Eval.*, vol. 46, no. 2-3, pp. 155–176, Oct. 2001.

[11] E. Gelenbe, R. Lent, A. Montuori, and Z. Xu, “Cognitive packet networks: Qos and performance,” in *Modeling, Analysis and Simulation of Computer and Telecommunications Systems, 2002. MASCOTS 2002. Proceedings. 10th IEEE International Symposium on*, 2002, pp. 3–9.

[12] G. Sakellari, “The cognitive packet network: A survey,” *The Computer Journal, vol. 53, no. 3*, pp. 268–279, 2010. [Online]. Available: http://comjnl.oxfordjournals.org/content/53/3/268.abstract.

[13] E. Gelenbe and A. Nunez, “Traffic engineering with cognitive packet networks,” in *Simulation Series*, vol. 35, 2003.

[14] E. Gelenbe and R. Lent, “Power-aware ad hoc cognitive packet networks,” *Ad Hoc Networks*, vol. 2, no. 3, pp. 205 – 216, 2004, quality of service in ad hoc networks.

[15] L. Hey, “Power aware smart routing in wireless sensor networks,” in *Next Generation Internet Networks, 2008. NGI 2008*, April 2008, pp. 195–202.

[16] E. Gelenbe, M. Gellman, and G. Loukas, “An autonomic approach to denial of service defense,” in *World of Wireless and Multimedia Networks, 2005. WoWMoM 2005. Sixth IEEE International Symposium on*, June 2005, pp. 537–541.

[17] “IEEE standard for local and metropolitan area networks– station and media access control connectivity discovery,” *IEEE Std 802.1AB-2009 (Revision of IEEE Std 802.1AB-2005)*, pp. 1–204, Sept 2009.

[18] A. Iyer, V. Mann, and N. Samineni, “Switchreduce: Reducing switch state and controller involvement in openflow networks,” in *IFIP Networking Conference*, 2013, May 2013, pp. 1–9.

[19] N. Van Adrichem, C. Doerr, and F. Kuipers, “Opennetmon: Network monitoring in openflow software-defined networks,” in *Network Operations and Management Symposium (NOMS), 2014 IEEE*, May 2014, pp. 1–8.

[20] B. Lantz, B. Heller, and N. McKeown, “A network in a laptop: Rapid prototyping for software-defined networks,” in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, ser. Hotnets-IX, New York, NY, USA: ACM, 2010, pp. 19:1–19:6.