Dynamic Resource Discovery Protocol for Software Defined Networks

Yury Jiménez, Cristina Cervelló-Pastor, *Member, IEEE*, and Aurelio García

***Abstract*—In Software Defined Networking (SDN), the network management is logically centralized but physically distributed among the controllers in order to improve the scalability compared with a completely centralized model. As an alternative to managing the network state in SDN in a distributed way, this work proposes a distributed protocol called SDN Resource Discovery Protocol (SDN-RDP). This protocol divides and distributes the network management among the controllers. In essence, each controller discovers a portion of the network topology creating a minimum-latency tree rooted at each controller, thus creating the control layer. Through the delay-constrained shortest paths, henceforth called control channels, the controllers collect network state information from nodes, and decide and distribute the forwarding decisions to them. This process is asynchronous since there is no global initialization process to activate the execution of the protocol, and knowledge about the network is not required. Simulation results show that the proposed protocol works efficiently on large networks in terms of time and load.**

***Index Terms*—Software Defined Networking (SDN), controllers, Resource Discovery Protocol (RDP), control layer, tree.**

I. Introduction

In Software Defined Networking (SDN) the network intelligence is delegated to a set of centralized servers called controllers, which abstract the information of the underlying network topology. It has been demonstrated that the controller scalability is affected by the number of requests that they manage and the latency between them and the nodes [1] [2]. Most of the approaches assume that the centralized controller use Link-Layer Discovery Protocol (LLDP) for detecting changes on network topology and make available physical topology information for controllers. However, this protocol does not distribute the load among controllers by itself, it just keeps updated the network state on them. Despite the critical role of topology information in enhancing the manageability of SDN networks, to the best of our knowledge, none of the proposed network management platforms offer a general-purpose tool for distributing both the discovery and management of the physical network among the controllers [3]. In this work, we propose the SDN Resource Discovery Protocol (SDN-RDP), a protocol that allows to the controllers discover the network topology and distribute its management by themselves.

The remainder of this letter is organized as follows. Section II describes the notation needed to formulate the proposed protocol. Section III describes the SDN Resource Discovery Protocol and its complexity. Section IV is devoted to the analysis of the protocol through simulations. Finally, Section V presents the conclusions.

The authors are with the Department of Telematics Engineering, Polytechnic University of Catalonia, Esteve Terradas, 7, 08860, Castelldefels, Spain, e-mails: {yury.jimenez;cristina;aurelioj.garcia}@entel.upc.edu

This work has been supported by the Ministerio de Economía y Competitividad of the Spanish Government under project TEC2013-47960-C4-1-P.

II. Formulation

Consider a physical network that is modeled using a graph denoted by the tuple $G = (V, E)$, where V is partitioned into the set of nodes n and controllers C, and E is the set of links. Then, $C = \{C_1, \ldots, C_k\}$ is the set of distributed controllers that defines the control layer. The set of adjacent neighbours of each node $i \in V$ is defined by N_i. The control layer created by each controller C_i is defined as $G_{\mathrm{C}_i} = (n_{\mathrm{C}_i}, e_{\mathrm{C}_i})$, where $n_{\mathrm{C}_i} \subseteq n$ is the set of nodes managed by the controller $C_i \subseteq C$, and $e_{\mathrm{C}_i} \subseteq E$ is the set of links that defines the control channels. The global control layer is defined by the set of control layers built by the controllers, $G_{\mathrm{C}} = \{G_{\mathrm{C}_1}, G_{\mathrm{C}_2}, ..., G_{\mathrm{C}_k}\}$.

III. SDN Resource Discovery Protocol

The process of creating the control layer is executed in two phases, the *forwarding* phase where the controllers announce their presence and the nodes that start the creation of the control channels (leaf nodes) are discovered, and the *backward* phase where the nodes decide which node to join in direction of a controller, thus creating the control layer. Below, the in-band control messages used by SDN-RDP are described.

- *Announcement messages* (AN): Controllers and nodes announce events through these messages. The controllers announce their presence to the network (message Type 0), and the nodes announce network changes (e.g., broken links, node failures, etc) to the controllers (message Type 1).
- *Response messages* (RES): These messages are used to create the control channels. Nodes send a join message (RES_JOIN) to join to a parent node, and a leave message (RES_LEAVE) to announce to their neighbours that the node has already been joined to a node.
- *Improved Announcement messages* (IAN): By sending these messages the nodes can announce to their neighbours that a nearer path to a controller has been discovered.

A. Forwarding Phase (FP)

To build the control layer each controller advertises its presence by forwarding an AN message to all their N_i adjacent nodes. This message contains information concerning the controller identifier, the sender node identifier and its latency to the controller. We consider that nodes discover their adjacent neighbours and the latency to them by exchanging Hello messages. Upon receiving the first AN message, the nodes update the packet fields and forward the message through every outgoing interface except that which the message was received. This process is repeated for each node, consequently the AN messages are distributed all over the network.

The nodes that receive at least one AN message through all their neighbours stop forwarding this message. Consequently, these nodes discover the latency to a controller through each one of their (N_i) neighbours, and they become a *discovered node*. Let $D_n = \{D_1, D_2 \dots\}$ be the set of discovered nodes in G. These nodes based on the known information can find out if the latency of any N_i node can be improved via another neighbour. When a node $i \in D_n$ discovers that through node $k \in N_i$ the known latency of node $j \in N_i$ to a controller can be reduced, it forwards an IAN message to node j announcing that a better path to a controller has been found. This message is retransmitted back hop-to-hop as long as it encounters a node that can not improve its latency through this path. Upon receiving this message, the nodes update the delay and sender identifier fields in the message. The *discovered nodes* that cannot improve the latency of any neighbour to a controller are called *leaf nodes*. The set of leaf nodes is defined as $L_n = \{L_1, L_2, \dots\}$, where $L_n \subseteq D_n$. These nodes determine the minimum latency possible to a controller $C_k \in C$, and initiate the control layer construction process (backward phase).

On the other hand, nodes that have not received one AN message through all their neighbours wait for the answer to the AN messages sent through each interface. This answer can be an IAN message or a RES message. The nodes that become *discovered nodes* can make a decision as explained below in the backward phase. In this context, the IAN message has 2 purposes, i) discover the leaf nodes, and ii) discover the nearest controller from each node. Given that the controllers are not synchronized, we consider that the controllers that have not started the network discovery process initiate the FP when receiving one AN message. In this way, the protocol guarantees that the node management is distributed among all the controllers, besides simplifying the SDN-RDP resolution.

B. Backward Phase (BP)

In this phase the nodes decide to which node to join, leading backwards in the direction of a controller, and as a result, a control layer G_{C_i} on top of each controller $C_i \subseteq C$ is created. This process starts from the *leaf nodes* which send a RES_JOIN message to their neighbour node that is nearest to a controller and a RES_LEAVE message to the rest of the adjacent nodes.

In order to ensure the discovered control channels are the shortest ones and also that all nodes are managed, only the nodes that i) have received a response message to the sent AN messages, and ii) can not improve the latency of any node $k \in N_i$, can decide which node to join in the direction of a controller. Therefore, the nodes that satisfy these conditions have to update their identifier into the received RES_JOIN message and forward it to their best neighbour. Consequently, the RES_JOIN messages converge on the controllers, which receive (at most) a join message from each N_i adjacent node.

Algorithm 1 Forwarding Phase for node i

```
Require: At least one controller has started the $FP$, and node $i$ knows its adjacent neighbours and the delay to them, $d(i,j)$
  $d(k, C_k) = \emptyset \leftarrow$ delay from each node $k \in N_i$ to a controller $C_k \in C$
  if it has received one $AN$ or a $RES$ message then
    update local delay information, $d(k, C_k)$
    if this is one $AN$ message and this is the first $AN$ received then
      forward the message through every outgoing interface except those which:
      i) the message was received and,
      ii) connect to neighbours whose identifiers were included in this message
    else
      if it has received at least one $AN$ or one $RES$ message from all its neighbours then
        change state to discovered node
      end if
    end if
  else
    if it has received an $IAN$ message or state is discovered node then
      for each node $k \in N_i$ from which an $AN$ or $IAN$ message was received do
        if $d(k, C_l) + d(i, k) < d(j, Cm) + d(i, j)\ \forall j \in N_i, j \neq k$ where $j$ are the nodes from which an $AN$ message was received then
          send an $IAN$ message to node $j$
          update local delay information, $d(j, C_i)$
        else
          change state to leaf nodes
        end if
      end for
    end if
  end if
```

There are two situations that prevent condition (i) can be fulfilled. These are: 1) forwarding conflicts and 2) network failures. These cases, as well as the proposed solution, are explained below.

1) Forwarding Conflicts: As nodes work asynchronously, during the forwarding phase some nodes i and j can receive and forward over the same link (i, j) an AN message. This fact causes that the involved nodes may wait indefinitely for a response from each other to make a decision in the BP. Consequently, these nodes can not take any decision, stopping the control channel building. In order to solve this problem, conditions (1) and (2) have been defined. These conditions are evaluated independently by each node. By evaluating (1) each node i and j discovers if the latency to the known controller can be improved through its neighbour.

$$d(i, C_m) \leq d(j, C_l) + d(i, j), \qquad (1)$$

where $C_m, C_l \in C$ and $d(i, j)$ is the latency in link (i, j). Thus, if condition (1) is true for node i, it will decide that the path to the controller C_m is better than the path to the controller C_l through the neighbour j. In this case, the AN message sent by the neighbour j is unanswered and becomes an implicit RES_LEAVE message. Thus, node i does not need to send the RES_LEAVE message. If condition (1) is false for node i, it will decide that the path to controller C_l

through node j is better than the path to controller C_m. In this case, it will not wait for a response from j to make a decision. By evaluating condition (2) node i knows the decision made by node j without any dialogue between them. If condition (2) is true for node i, it will not wait for a response because it knows that node j will join another node. On the other hand, if condition (2) is false for node i, it will wait for a response from node j.

$$d(j, C_l) \leq d(i, C_m) + d(i, j) \tag{2}$$

These conditions are valid if both controllers are the same or even if they are different. By evaluating these conditions, each node discovers which is its best neighbour leading to the nearest controller. In this way, the local conflicts are resolved implicitly without generating any additional traffic and in a minimum time. These conditions also ensure that no cycles exist in the control layer.

2) Failures: Network failures can prevent the nodes receiving the response messages and, as a result, being unable to make a decision. In order to avoid this situation, a timer is activated after the first response message is received by a node. Therefore, if after a time t the node has not received a response from an adjacent neighbour, it assumes that the latency to a controller through this neighbour is infinite. After that the node can decide to which parent node to join.

Each RES_JOIN message received by the controller contains the sequence of node identifiers in the control channel. This information has a general tree structure, which is converted to a binary string and forwarded in a linear sequence. In this way, each control layer G_{C_k} is built hop-to-hop from the leaf nodes to a controller. After receiving each RES_JOIN message, the controllers forward a $HELLO$ message to each node contained in the message in order to negotiate the secure connection setup, which is answered with the same message from each switch. As a result, the control channel between each node and a controller is established, this channel is used by the nodes to announce their information about node connectivity (N_i) and the controller to which its neighbours have been joined. By aggregating this information the controllers can discover their partial network topology and the control channels among them.

C. Protocol complexity

The complexity of SDN-RDP is defined in terms of time and number of messages to create the control layer G_C. Given a network of N nodes and k controllers, the network discovery time is defined by the mean number of nodes assigned to each controller, $N_c = \lceil \frac{N-k}{k} \rceil$, the mean number of neighbours per node, D, and the average time (t_t) defined as the sum of propagation delay (t_p) and transmission time (t_{tx}). Thus, the maximum time to create G_C is defined by $2 \times N_c/D \times t_t$, assuming that the node processing time is negligible. Therefore, the time complexity is $\mathcal{O}\left(\frac{N}{kD}\right)$. On the other hand, the upper number of messages per controller is bounded by $k \times D + (N-k)\left[2 \times (D-1) + 1\right]$, considering that just one AN message is sent by each controller through each one of its output interfaces, the nodes forward an AN message just one time, and the nodes respond to all of the received AN messages. Therefore, the message complexity is $\mathcal{O}(ND)$.

Algorithm 2 Backward Phase for node i

```
Require: Node knows its state; $d(k, C_k) \leftarrow$ delay from each $k \in N_i$ to a controller $C_k \in C$
  if state is discovered node then
    if it has sent and received through the same interface an $AN$ message then
      evaluate $d(i, C_m) \leq d(j, C_l) + d(i, j)$
      evaluate $d(j, C_l) \leq d(i, C_m) + d(i, j)$
      decide the (implicit or not) response
    end if
  end if
  if state is leaf node or discovered node and it has received a response message or an implicit response message then
    forward a $RES\_JOIN$ message to the nearest adjacent node to a controller and a $RES\_LEAVE$ to the rest of the nodes (if it is not implicit)
  end if
```

IV. Simulation and Results

We have implemented our complete solution from scratch in OMNET++. In order to show the scalability and efficiency of our protocol, we have evaluated the time and number of messages required to create the control layer over a set of randomly generated graphs of different sizes while varying the number of controllers. In all these cases we assume a link capacity between 100 Mbps to 10 Gbps and the distances between any pair of nodes are selected randomly in a range of 1 to 15 Km. The placement of the controllers was selected by using $k - Critical$ approach [4]. This approach selects the controllers to ensure that the management of the whole network is guaranteed while satisfying a maximum delay between each node and its controller. We present the simulation results for the SDN-RDP together with their respective 95% confidence intervals based on Student-t distribution. The adjacency matrices for these graphs are generated by using the Gephi software. It generates networks with a given number of nodes n, which are randomly connected by undirected edges. For a random pair of nodes, there is a probability p (wiring probability), where $0 < p < 1$, that there exist a link connecting them. This implies that the degree of random graphs generated using a fixed value of p increases when n increments. Table I shows the main characteristics of the generated networks, all of them were generated using a wiring probability of 0.05.

Size	Avg. Number of links	Avg. Node degree
50	65	3
100	253	5
200	994	10
500	6243	25

TABLE I
Simulated networks.

A. Results

In order to show the basic operation of our protocol and evaluate its performance in a controlled scenario, the controllers run the SDN-RDP protocol simultaneously. Fig. 1(a) and 1(b) show that networks with high connectivity (or density) process a higher number of messages per each node than networks with low connectivity, and also reduce the control layer creation time with respect to networks with low connectivity. In networks with high connectivity, the number of forwarding hops of the messages decreases because the network diameter is small. Thus, the AN messages are quickly spread, discovering the leaf nodes in few hops. Networks with low connectivity reduce the message conflicts in the network, but increase the construction time of the control layer since the diameter is large. That is, the convergence time of the response messages to the controller is limited by the longest delay time of all the shortest paths found. In this scenario, for each network size, the average number of messages on each node remains constant even if the number of controllers is low (Fig.1(a)). This is because, i) conflicts are solved using implicit response messages and ii) IAN messages are not required as nodes receive the AN message through their shortest path. We have also evaluated the implications of the controller selection in the SDN-RDP performance. Fig. 2 shows the control layer creation time for different number of controllers selected randomly (dashed lines) and using $k - Critical$ (solid lines). As can be seen in Fig. 2, the control layer creation time required by controllers selected randomly was significantly higher than the time spent by controller selected using k-Critical in all cases. As controllers selected randomly may be geographically close or far way among them, some controllers may have more load than others, increasing the creation time of the control layer and therefore affecting the SDN-RDP performance. The average number of messages on each node, for each network size, is slightly higher when the control layer is created by controllers selected randomly, but in both cases (controller selected randomly and using k-Critical) the average number of messages are remained constant on the nodes. SDN-RDP has been designed to be scalable over a wide range of network sizes. Presented results indicate that, for a given network, the average number of required messages to create the control layer is invariant with respect to the number of controllers for networks of any density. With respect to the control layer computation time, improvement in the case of networks with low connectivity is only observed when multiple controllers are used. However, the control layer creation time is reasonable even when using just 1 controller. According to the evaluation, we can deduce that the resulting control layers deal with real network requirements. For instance, the data layer fault recovery may be achieved in a scalable way within 50 ms, the time required in transport networks. This topic will be further explored in the future work.

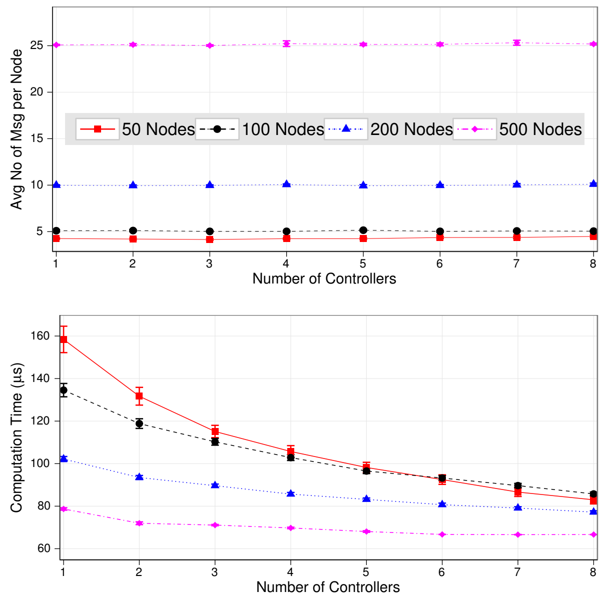

Fig. 1. Networks with different sizes, varying the number of k-Critical controllers.

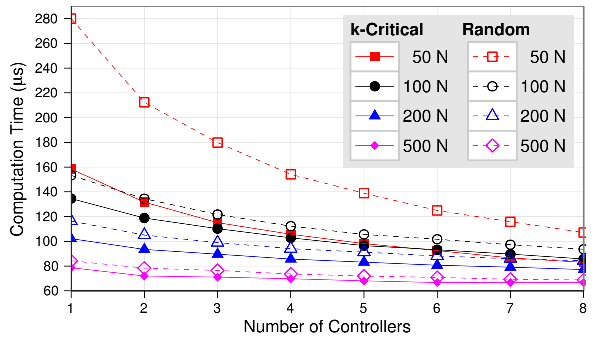

Fig. 2. Control layer computation time; CI are omitted to improve readability. (Different number of nodes, k-Critical and randomly selected controllers.)

V. Conclusions

We have presented the SDN-RDP protocol that distributes and divides the network management among controllers creating a control layer on top of any physical topology. The resulting control layer has a tree topology, through the shortest paths or control channels the network can be discovered, monitored and time-efficiently managed since the switches are managed by their nearest controller. Although this work only takes into account the delay other parameters may be considered when building the control layer.

References

[1] S. Hassas, A. Tootoonchian, Y. Ganjali. "On Scalability of Software-Define Networking," *IEEE Commun. Mag.*, vol. 51, no. 2, pp. 136–141, Feb. 2013.

[2] M. Reitblatt, N. Forter, J. Rexford and et al., "Consistent updates for software-defined networks: change you can believe in!," *ACM SIGCOMM Workshop HotNets*, pp. 1–6, Nov. 2011.

[3] B. Nunes, M. Mendonca, X. Nguyen, and T. Turletti, "A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks," *IEEE Communications Surveys & Tuts.*, vol. 16, no.3, pp. 1–18, 2014.

[4] Y. Jiménez, C. Cervelló-Pastor, and A. García, "On the controller placement for designing a distributed SDN control layer," *in Proc. Conf. IFIP Netw*, Trondheim, Norway, Jun. 2014.