An Efficient MPLS-Based Source Routing Scheme in Software-Defined Wide Area Networks(SD-WAN)

El Kamel Ali
Prince Lab, ISITCOM
University of Sousse
Sousse,Tunisia
Email : ali.el.kamel@hotmail.com

Majdoub Manel
Prince Lab, ISITCOM
University of Sousse
Sousse,Tunisia
Email : Manel.Majdoub@issatso.rnu.tn

Youssef Habib
Prince Lab, ISITCOM
University of Sousse
Sousse,Tunisia
Email : habib.youssef@fsm.rnu.tn

***Résumé*—Software Defined Networks (SDN) is a promising network paradigm that offers flexibility, efficiency and fine-grained control over forwarding Elements (FE) by decoupling control and data planes. The forwarding decision is often made by the controller by managing flow table entries in the switches. Certainly, flow table management raises a lot of concerns and various schemes have been proposed such as the traditional Hop-by-Hop Forwarding scheme. Studies have proved that this scheme may lead to performance degradation due to a huge control traffic and a massive flow table consumption. To reduce performance degradation, source routing was proposed. Unfortunately, this may also lead to significant bandwidth overhead, particularly in large-scale networks.**

This paper deals with the concept of source routing using MPLS labels. Basically, our scheme ensures flexibility and is suitable for application both in SD-LAN and SD-WAN by supporting various n-port switches ($n \geq 4$). Using a MaxHop clustering algorithm, the network is divided into several sections. In each section, a trade-off between control traffic overhead, bandwidth overhead and flow table usage is achieved using a linear weighted scalarization of the Multi-Objective optimization problem. Finally, we present the θ-MOBO algorithm to be run by the controller.

Simulation results show that the proposed scheme outperforms parallel solutions such as MPLS label-based forwarding [1] [2] and Hop-by-Hop forwarding [3] schemes in terms of the bandwidth overhead and flow rejection rate.

I. Introduction

Today, Software Defined Networks raises a lot of concerns due to its ability to offer resources sharing flexibility, adaptability, fine-grained control and network virtualization by decoupling the data plane and the control plane. Based on the OpenFlow protocol [17], a controller has a global overview on the network and can order updates to any switch directly as soon as required. When the first packet of a flow reaches an ingress switch, a request is forwarded to the controller, which establishes a path toward the destination and updates flow tables in switches belonging to that path with matching fields as well as associated actions. This is called the standard Hop-by-Hop Forwarding [3].

With the emergence of large-scale networks (such as Wide Area Networks, WAN), the traditional Hop-by-Hop forwarding scheme seems to be unsuitable for use since it may lead to performance degradation caused by a huge amount of control traffic and massive flow table entries. Therefore, Source routing [18] was proposed. It consists of defining the routing path information once at ingress switch. The routing information is embedded in a MPLS Label or VID tag and attached to the packet as a header. The header is inspected at each node along the path to keep the packet forwarding. Source routing is largely used in Software-defined WANs (SD-WAN).

In this paper, we suggest a source routing scheme that uses MPLS labels to carry routing information. Our scheme can be deployed both in SD-LAN or SD-WAN. Our scheme is able to support any n-port switch, where $n \geq 4$, by expressing a per-hop routing information as a pair of binary values : the class cl_j of the switch S_j and the forwarding port number p_j used in S_j. Based on the class of the switch, we can compute the total bit number required to encode all ports belonging to that switch.

For scalability purposes, we divide the network into several sections using a MaxHop clustering algorithm. In each section, a trade-off between the control traffic overhead, the bandwidth overhead and the flow table consumption is achieved. Thereby, we formulate the multi-objective optimization problem and we resolve it using a linear weighted scalarization. Finally, we describe the θ-MOBO algorithm to be run by the controller.

This paper suggests the following contributions : Firstly, we introduce the standard hop-by-hop forwarding scheme, MPLS-based forwarding scheme, JumpFlow [4] and AJSR [5], and analyze their benefits and drawbacks. Secondly, we model and analyze the control traffic overhead problem, the bandwidth overhead problem and the flow entry placement problem, respectively. Therefore, We formulate a Multi-objective optimization problem by combining the above problems. The resulting problem is resolved using a multi-objective programming based on a linear scalarization that depends on a set of weights. Finally, we simulate and evaluate the proposed scheme against the standard MPLS-based forwarding scheme and Hop-by-hop forwarding schemes. Experimental results show that our scheme can effectively outperform parallel solutions in SD-WANs.

2161-5330/17 $31.00 © 2017 IEEE
DOI 10.1109/AICCSA.2017.165

The rest of the paper is organized as follows. Section II discusses some related work. Section III introduces the problem background and discusses the motivation for this paper. Section IV models the problem and describes the proposed solution. Section V analyzes simulation results. Section VI concludes the paper.

II. Previous Work

For no later, more concern is given to the deployment of SDN in large-scale networks. The first example was developed by Google which integrates SDN into the private WAN connecting their private datacenters all around the world(the solution was known under the name B4) [6]. Therefore, it was proved that deploying SD-WAN is feasible, but it seems to be so hard and requires more work, compared to SD-LAN, before it can be widely deployed.

Indeed, many limitations postpone the emergency of SD-WAN. The most important one is the limited space of flow tables. Like Cohen et al. in [7], some researches present flow entry placement schemes as an optimization problem that may solve the flow table space limitation. based on an argument that the load of flow table is unbalance among the switches in the network, they suggest to efficiently use the TCAM resource on each switch by decomposing a large flow table into small pieces and distributing these small flow tables across the network, while preserving the overall SDN policy semantics [8]. As a result, the total number of flow entries was reduced since some table flows were shared with other connections.

However, these schemes led to an unwanted packet traveling inevitably, which may raise security problems. Moreover, decomposing the flow table is a hard problem [9].

As a remedy, source routing has gained attention since it can reduce the redundancy of flow table. Source Routing consists of encapsulating forwarding path in labels, such as MPLS and VLAN tag. However, more attention should be given to these labels since they may cause bandwidth overhead.

The problem of reducing or eliminating this overhead becomes a very interesting challenge. In [10], a system denoted SwitchReduce aims to reduce switch states and controller involvement in OpenFlow networks. Authors in [11] proposes a hierarchical Segment routing (H-SR) which is an implementation of segment routing for Carrier Ethernet networks to improve scalability of segment routed networks based on a hierarchical segment routing framework [12]. Indeed, the network is divided into clusters and specific swap nodes are selected within every cluster. The swap node works as an intermediate node, which breaks the complete route into sub-routes and limits the number of labels. However, swap nodes selection depends largely on the network topology. Moreover, JumpFlow [4] is proposed as a forwarding scheme that uses the VLAN identifier (VID) field in the packet header to carry routing information. Despite that JumpFlow eliminates the bandwidth overhead, it has many drawbacks and is unsuitable to be deployed in SD-WAN Moreover, it assumes that the network consists of single kind of switches, having 8 or 16 ports and all switches are treated similarly. in [5], authors propose

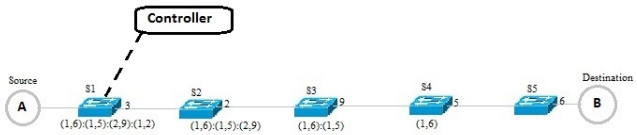

FIGURE 1. Flow entry placement scheme $F_P = \{1\}$

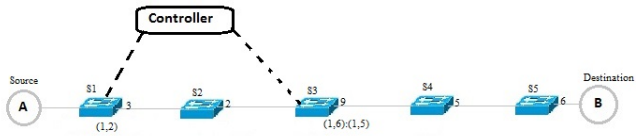

FIGURE 2. Flow entry placement scheme $F_P = \{1, 3\}$

an efficient MPLS-based forwarding scheme called Arbitrary Jump Source Routing (AJSR) which uses the source routing feature. it aims to achieve a trade-off between the control traffic overhead and the bandwidth overhead by dividing the complete routing path of a particular flow into arbitrary length sections and distributing these sections at different switches along the flow's routing path. Although AJSR can be deployed in SD-WAN, it looks to be not efficient since it can induce a significant bandwidth overhead, since each per-hop forwarding information is carried using one MPLS Label.

III. Proposed Mechanism

In this section, we will describe our scheme in detail. Firstly, we show an overview of the solution. Then, we formulate an analytic model using the multi-objective optimization problem. Finally, an heuristic based on the multi-objective programming is proposed to achieve a triple-axis trade-off.

A. Overview

Inspired both from AJSR and Jumpflow, the proposed scheme is based on network clustering and aims to achieve a trade-off between switch-to-switch communication overhead, controller-to-switch communication overhead and table flow consumption.

However, our scheme differs from AJSR and JumpFlow in the following aspects : (i) Jumpflow assumes that all switches in a routing path have the same number of ports which can be equal to 8 or 16, and, therefore, they are treated similarly. However,

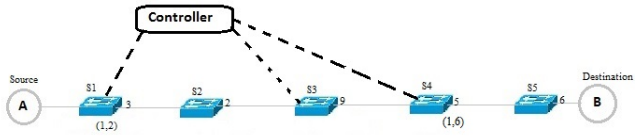

FIGURE 3. Flow entry placement scheme $F_P = \{1, 3, 4\}$

switches in large-scale networks may be heterogenous and the number of ports can reach over 128 ports [5]. Our scheme uses a pair of binary values (Switch class, Port number) which helps to express any n-port switch ($n \geq 4$) in the path. (ii) Jumpflow uses a limited 12-bits VID field. This allows to carry at maximum $[12/\log(n)]$ hops per packet [4], where n denotes the number of switches. As n goes large like in large-scale networks, JumpFlow seems to be unsuitable for application in SD-WAN. Our scheme uses MPLS labels to keep packet forwarding. This allows to carry over (n*8) hops per packet, where n denotes the used number of MPLS labels. (iii) AJSR uses one MPLS label to carry forwarding information of one hop from the routing path which may lead to bandwidth overhead as the number of hops goes on.

Carrying routing path in MPLS labels brings a lot of benefits. Indeed, MPLS labels are enough to encode any port of any switch. Each Switch should only perform pop, push and read operation on each packet without requirement to modify it. This will help to support heterogeneous switches with different number of ports. However, carrying routing path in MPLS labels may lead to bandwidth overhead introduced by appending multiple MPLS labels to the packets. Nevertheless, the bandwidth overhead can be significantly reduced by properly encoding forwarding ports with exactly the required number of bits. As an example, if the packet crosses the ports 2,10 and 31 during forwarding, the routing information requires only 2+4+5=11 bits to be forwarded, however, 3*32=96 bits are used if we consider the MPLS-based forwarding scheme.

B. Analytic model : general background

Generally, a network is defined as a directed graph G(S,L), where S and L denote the set of switches and the set of links, respectively. A routing path is a vector $P = (s_1, s_2, ..., s_N) \subseteq S^N$ between the source and the destination, where s_j (1≤j≤ N) is the j^{th} switch of the routing path and N is the length of the path. The current flow table usage relative to path P can be expressed as a vector $U_P = (u_1, u_2, ..., u_N)$ where u_j denotes the flow table usage of the switch s_j. Let $C_P = (c_1, c_2, ..., c_N)$ be the set of flow table capacities of the switches belonging to path P, where c_j represents the capacity of the flow table relative to switch s_j.

the flow entry placement scheme can be modeled as a ordered subset $F_P = (f_1, f_2, ..., f_r)$ where $1 \leq r \leq N$ and $F_p \subseteq P$. The vector F_P is denoted the set of contact switches. A contact switch is a switch which loads a section of routing path for a flow. Usually, the ingress switch must be a contact switch, thus, for every flow entry placement scheme, we have f1 ={1}. Since the flow table usage u_j of the switch s_j should not exceed the capacity constraint c_j, we can formulate the flow table usage problem as follows (Eq 1) :

$$\forall f_j \in F_P : u_{f_j} + 1 \leq c_{f_j} \tag{1}$$

C. Routing information modeling

To model the routing information of a flow that crosses a specific path P, we first define a vector $K_P = (k_1, k_2, ..., k_N)$, where $k_j \in \mathbb{N}$ represents the class of the switch s_j. The switch's classification depends on the number of relative physical ports. Therefore, the class 0 denotes a 4-port switch, the class 1 denotes a 8-port switch, the class m denotes a 2^{m+2}-port switch, and so on. If n_j is the number of ports on the switch s_j, the class k_j can be obtained as follows[2] :

$$k_j = \lceil \log_2(n_j) \rceil - 2 \tag{2}$$

The class of the switch helps to compute the required number of bits which is enough to encode all ports. Indeed, if k_j is the class of the switch s_j, the required number of bits which is enough to encode all ports is $k_j + 2$.

For a packet on path P, a forwarding information i_j is required on each switch s_j to keep the packet forwarding. This forwarding information is defined using a pair of binary values $i_j = (k_j, pr_j)$, where k_j and pr_j are the class of the switch s_j and the local forwarding port, respectively.

To allow extraction of separate routing information by each switch, we should use the same number of bits to represent different classes. This number is defined by the switch having the greatest number of ports. Therefore, each class k_j in a routing information should be defined using a number of bits equal to $\widehat{N_p}$, which is shown in Equation (Eq.3).

$$\widehat{N_p} = \lfloor \log_2(\widehat{K_p}) \rfloor + 1 \tag{3}$$

Where $\widehat{K_p}$ denotes the class of the switch having the greatest number of ports and is computed using Equation (Eq.4) :

$$\forall s_j \in P : \widehat{K_p} = \lceil \log_2(\max_{s_j}(n_j)) \rceil - 2 \tag{4}$$

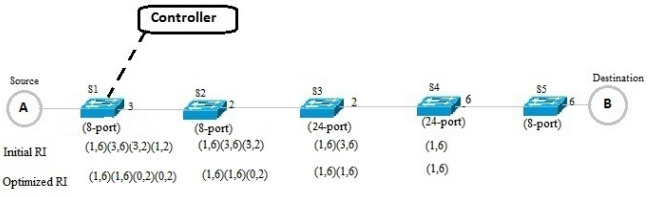

FIGURE 5. Optimizing Routing Information for $F_P = \{1\}$

D. Optimizing Routing information

Take figure Fig5 as an example. The initial routing information is I_P ={(1,6)(3,6)(3,2)(1,2)}. The total number of bits required to represent the routing information is 5+7+7+5=24 bits. The outgoing port pr_3 in the switch S_3 is number 2 which requires only 2 bits to be encoded. However, 5 bits are allocated since it is a 24-port switch($\lceil \log_2(24) \rceil$=5). The forwarding information can be reduced if we assume that the port pr_3 belongs to a 4-port switch. Therefore, the optimized routing information is I'_P={(1,6)(1,6)(0,2)(0,2)} which gives a total bit number of 5+5+4+4=18 bits.

Let $|I_P|$ be the size of the routing information. Hence, $|I_P|$ should not exceed an MPLS label size (32 bits). Otherwise, another contact switch should be defined. As the controller has a global overview of the network, it can create a MaxHop

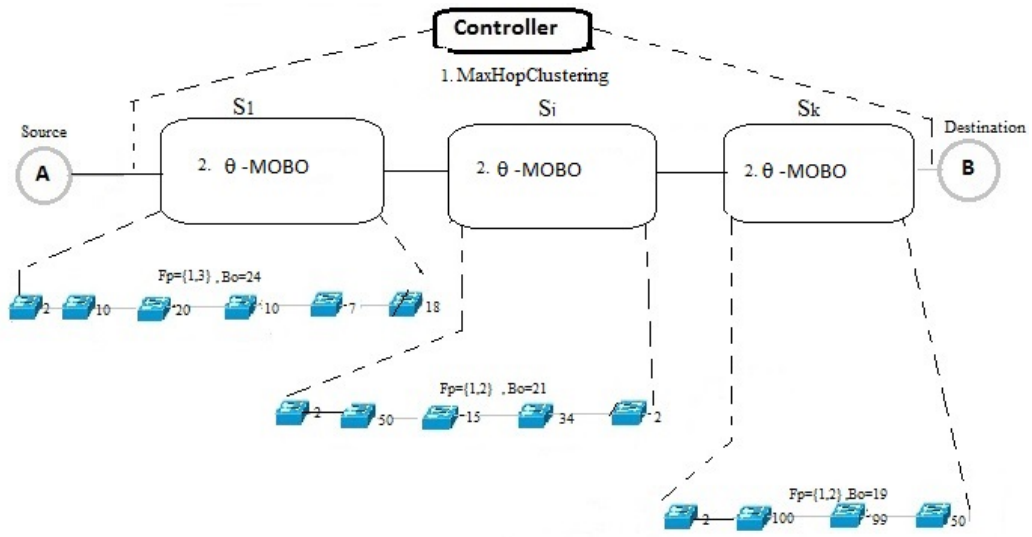

FIGURE 4. Optimal Flow Entry Placement based on the Bandwidth Overhead

Algorithm 1 MaxHopClustering

Require: $I_P = (i_1, \cdots, i_N)$ {initial routing information. Every item i_j is a pair of values (cl_j, pr_j) which defines the class of s_j and the local port used to forward packets, respectively.}

Ensure: $Sec = (S_1, \cdots, S_K)$

$\{\forall i, S_i$ are subsets from I_P. $\bigcup_{i=1}^{K} S_i = I_P$ and $\bigcap_{i=1}^{K} S_i = \emptyset\}$

$Sec \leftarrow \emptyset$

while $I_P \neq \emptyset$ **do**

  $S_k \leftarrow \emptyset$

  for all $i_j \in I_P$ **do**

    $x \leftarrow 2^{cl_j+2}$

    while $pr_j \leq x/2$ **do**

      $x \leftarrow x/2$

    end while

    $cl_j \leftarrow \lceil \log_2(x) \rceil - 2$

    $i_j \leftarrow (cl_j, pr_j)$

    if $|S_k|+|i_j| \leq 32$ **then**

      $S_k \leftarrow S_k \cup i_j$

    else

      Exit

    end if

  end for

  $Sec \leftarrow Sec + \{S_k\}$

  $I_P \leftarrow I_P \backslash \{S_k\}$

end while

return Sec

routing information by adding, incrementally, switches belonging to the path P until the destination is reached or the size of routing information is close to 32 bits.

The size of routing information $|I_P|$ is declared to be *close* to 32 bits if, by adding other forwarding information to I_P, $|I_P|$ exceeds 32 bits. We present the algorithm *MaxHopClustering*(Algorithm 1) to be used by the controller to split the path P into several sections. Each section should use one MPLS label to keep packet forwarding.

In each section S_k, the controller should select a set of contact switches $F_P \subsetneq S_k$ basing on three criteria :i)Minimizing the overall bandwidth overhead,ii)Minimizing the control traffic overhead and iii)Minimizing the standard deviation of the flow table usage compared to an up-to-date pre-computed average flow table usage to make sure that packets will be routed efficiently to the next section or, to the corresponding destination.

E. Bandwidth overhead

The Bandwidth overhead (Bo) refers to the total number of bits introduced along the path P. As an Example, using flow entry placement scheme $F_P = \{1, 3\}$ reduces the bandwidth overhead form 5+5+6+4=20 bits (Fig1) to 4+5+5=14 bits (Fig2). Moreover, considering flow entry placement scheme $F_P = \{1, 3, 4\}$ (Fig3), the bandwidth overhead achieves 4+5=9 bits.

Generally, The flow entry placement scheme is initialized as $F_p = \{1\}$. For any flow placement scheme $F_P = \{f_1, \cdots, f_r\}$, the bandwidth overhead $Bo(F_P)$ is expressed as follows(Equation 5) :

$$Bo(F_P) = \sum_{i \in [f_1, f_r[} Bo(f_i, f_{i+1}) \tag{5}$$

$Bo(f_i, f_j)$ is defined as the total number of bits introduced between contact switches f_i and f_j. Conventionally, we assume that $Bo(f_i, f_j) = 0$ if f_i and f_j are contiguous switches. Otherwise, the bandwidth overhead is computed using Equation (Eq6) :

$$Bo(f_i, f_j) = (f_j - f_i - 1)\lceil \widehat{K}_{f_i,f_j} \rceil + \sum_{p \in]f_i,f_j[} \lceil \log_2(n_p) \rceil \quad (6)$$

Where

$$\widehat{K}_{f_i,f_j} = \log_2(\max_{p \in]f_i,f_j[}(n_p)) \quad (7)$$

Let $\mathcal{F}$ be the set of all possible flow entry placements in a subset of nodes of a specific path P. Our objective is to find a flow entry placement $F_P \in \mathcal{F}$ that minimizes the bandwidth overhead. Thereby, we formulate the bandwidth overhead minimization problem as follows(Equation 8) :

$$\begin{cases} \min\limits_{F_P \in \mathcal{F}} Bo(F_P) \\ \\ \textit{subject to} \\ Bo(F_P) \leq 32 \end{cases} \quad (8)$$

F. Control traffic overhead

The control traffic overhead refers to the total bit number of control traffic resulting from defining a flow entry placement $F_P = (f_1, \cdots, f_r)$. Let $d(s_i)$ be the link cost between the controller and the switch s_i and let M be the mean size of control messages. Thereby, we define the control traffic overhead as follows(Equation 9) :

$$CTo(F_P) = \frac{\sum\limits_{f_i \in F_P} d(f_i) M}{r} \quad (9)$$

Thereby, Our objective is to find a flow entry placement scheme $F_P = (f_1, \cdots, f_r) \in \mathcal{F}$ that minimizes the control traffic overhead (Equation 10).

$$\min_{F_P \in \mathcal{F}} CTo(F_P) \quad (10)$$

G. Flow table usage balancing

Unfortunately, some flow tables of core switches may be overflowed while underflowed in other switches, which may lead to unbalanced resource usage and performance degradation [13]. Therefore, selecting efficiently a flow table placement $F_P = (f_1, \cdots, f_r)$ where $1 \leq r \leq N$ that can balance the flow table consumption is a relevant problem.
Let $w_i = u_i / c_i$ be the weight of flow table usage in the switch s_i. Let $\widehat{W}$ be the average of flow table weights w_i; $\forall i$.
Given a set $\mathcal{F}$ of flow entry placements, the goal is to achieve a balanced flow table usage by properly minimizing the standard deviation of the current average of switch's weights compared to the resulting average after placing relative flow entries. Thereby, for a flow entry placement $F_P = (f_1, \cdots, f_r)$ with current average of flow table weights $\widehat{W}$, we define the standard deviation $\Delta(F_P, \widehat{W})$ as follows(Equation 11) :

$$\Delta(F_P, \widehat{W}) = \sqrt{\frac{\sum\limits_{f_i \in F_P} (w_{f_i} - \widehat{W})^2}{r}} \quad (11)$$

Thereby, Our objective is to find a flow entry placement scheme $F_P = (f_1, \cdots, f_r) \in \mathcal{F}$ that minimizes the standard deviation.

$$\begin{cases} \min\limits_{F_P \in \mathcal{F}} (\Delta(F_P, \widehat{W})) \\ \\ \textit{subject to} \\ w_{f_i} \leq 1 \qquad\qquad \forall f_i \in F_P \end{cases} \quad (12)$$

H. Multi-objective programming solution

Our objective consists of finding a set of contact switches that achieves a trade-off between the bandwidth overhead, the control traffic overhead and the flow table consumption by optimizing simultaneously the three objective functions defined in Equation 8, Equation10 and Equation12, respectively.
Given a set $\mathcal{F}$ of feasible solutions, we formulate a multi-objective optimization problem as follows(Equation 13) :

$$\begin{cases} \min\limits_{F_P \in \mathcal{F}} (Bo(F_P), CTo(F_P), \Delta(F_P, \widehat{W})) \\ \\ \textit{subject to} \\ F_P \subseteq P \end{cases} \quad (13)$$

In mathematical terms [14], it has been proved that it does not typically exist a solution that minimizes different objective functions simultaneously. Therefore, we define a set of Pareto Optimal solutions. A solution is called a *Pareto Optimal*, if there does not exist another solution that dominates it.
Let $Ob = (ob_1, ob_2, ob_3) = (Bo, CTo, \Delta)$ be the set of objective functions. Therefore, Given two feasible solutions F_P^1 and F_P^2 in $\mathcal{F}$, F_P^1 *dominates* F_P^2 if (Equation 14) :

$$\begin{cases} 1. \qquad ob_i(F_P^1) \leq ob_i(F_P^2)\,;\ \forall i = 1 \cdots 3 \\ 2. \quad \exists \textit{ at least one index } j\,;\ ob_j(F_P^1) < ob_j(F_P^2) \end{cases} \quad (14)$$

To be resolved, the linear scalarization [15] [16] of the multi-objective optimization problem is adopted. indeed, each objective is associated with a weight θ_i; $;\forall i = 1..3$. Thereby, the multi-objective optimization problem is formulated as a single-objective optimization problem that depends on $\theta = (\theta_1, \theta_2, \theta_3)$, as follows(Equation 15) :

$$\begin{cases} \min\limits_{F_P \in \mathcal{F}} g(Ob(F_P), \theta) \\ \\ \textit{subject to} \\ F_P \subseteq P \end{cases} \quad (15)$$

Where

$$g(Ob(F_P), \theta) = Ob.\theta^T = \sum_i \theta_i ob_i(F_P) \quad (16)$$

For the rest of this paper, we denote F_P^* a Pareto solution and $\mathcal{F}^*$ the set of all Pareto solutions. $\mathcal{F}^*$ is said to be a *θ-Optimization* of the multi-objective optimization problem (we called this the θ-MOBO). Finally, we suggest the algorithm shown in Algorithm2.

Algorithm 2 θ-MOBO

Require: $\mathcal{F},\ \theta = (\theta_1, \theta_2, \theta_3)$
Ensure: $\mathcal{F}^*$
  $M \leftarrow \infty$
  $\mathcal{F}^* \leftarrow \emptyset$
  for all $F_P \in \mathcal{F}$ **do**
    $X \leftarrow Bo(F_P)$
    $Y \leftarrow CTo(F_P)$
    $Z \leftarrow \Delta(F_P, \widehat{W})$
    if $\theta_1 X + \theta_2 Y + \theta_3 Z < M$ **then**
      $M \leftarrow \theta_1 X + \theta_2 Y + \theta_3 Z$
    end if
  end for
  for all $F_P \in \mathcal{F}$ **do**
    $X \leftarrow Bo(F_P)$
    $Y \leftarrow CTo(F_P)$
    $Z \leftarrow \Delta(F_P, \widehat{W})$
    if $\theta_1 X + \theta_2 Y + \theta_3 Z = M$ **then**
      $\mathcal{F}^* = \mathcal{F}^* \bigcup \{F_P\}$
    end if
  end for
  return $\mathcal{F}^*$

IV. Experimental settings

In this section, we evaluate the performance of MPLS-based forwarding scheme, Hop-by-hop forwarding scheme and our scheme, respectively.
In our simulations, we use the beacon controller (Beacon1.0.4) which communicates with switches through OpenFlow 1.0.3. The topology, shown in Figure 6, is created using the Mininet simulator (Mininet2.2.0). For simplicity, we assume that all switches have the same number of ports (64 ports). Port numbers in each switch are chosen randomly into the set [0,63]. Initially, the flow table capacity at each switch is fixed at 500 flow entries. A path is defined using the maximum number of crossed switches to simulate large scale-networks (the path is colored red in the network topology). The simulation consists of 6 steps. In each step, the source will generate multiple 1Mbps-CBR flows to be sent toward the same destination. At each step, a flow arrival rate is picked from the set [10,100,200,500,700,1000]. The link cost between the controller and a contact switch is configured according to the distance which separates the contact switch to the ingress switch, incremented by 1. We run our simulation 30 times.

V. Results analysis

Table 7 shows resulting Pareto Optimal sets $\mathcal{F}^*$ for different vectors θ_a,θ_b and θ_c. Figure 8 shows the result of the MaxHopClustering algorithm on our topology. Obviously, the θ-MOBO algorithm gives the Pareto optimal solution that satisfies simultaneously the three objective functions in the best way.
Figure 9 shows the total bandwidth overhead of MPLS-based forwarding scheme and our scheme as the flow arrival rate

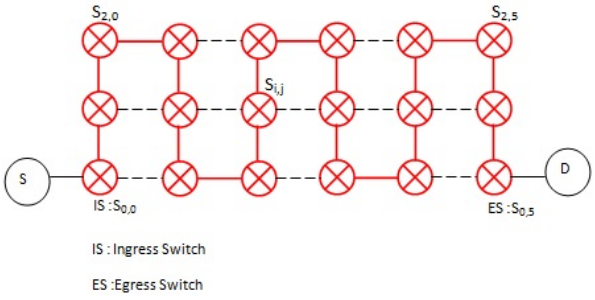

FIGURE 6. Network topology

	Pareto Optimal Set			
Section Si	1	2	3	4
θ_a(1/4,1/4,1/2)	{1,2,4}	{6,7,9}	{11,14}	{16,17}
θ_b(1/4,1/2,1/4)	{1,2}	{6,7}	{11}	{16}
θ_c(1/2,1/4,1/4)	{1,2,4,5}	{6,7,9}	{11,12,14}	{16,17}

FIGURE 7. Pareto optimal sets for different vectors θ

changes. Firstly, we consider our scheme when the MaxHop-Clustering algorithm is not activated. After that, we launch the MaxHopClustering algorithm. In this figure, the bandwidth overhead of traditional MPLS-based forwarding scheme increases exponentially as the flow arrival rate increases from 10 to 1000 flows/step. However, it increases linearly for our scheme.
Obviously, Our scheme is able to reduce effectively the bandwidth overhead over 79% when MaxHopClustering is inactivated and reaches over 90% if the algorithm is activated. Indeed, the network is split into sections. In each section, routing information is carried using only one MPLS label. The total number of used MPLS labels falls from 17 in MPLS-based forwarding to only 4 labels in our scheme per each flow (more than 75%). Traditional MPLS-based forwarding scheme introduces a huge amount of bandwidth overhead because it requires the complete forwarding information to be encapsulated in the ingress switches. However, our scheme performs best since the network is divided into sections based on the routing information. In each section, a set of contact switches is picked and the complete routing information is divided into small parts and distributed to different contact

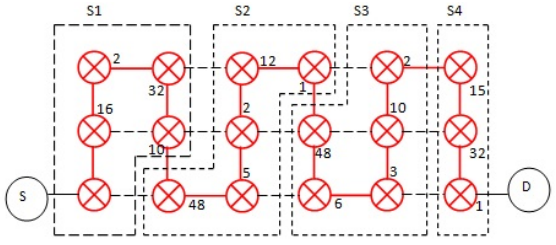

FIGURE 8. MaxHopClustering : Dividing topology into sections

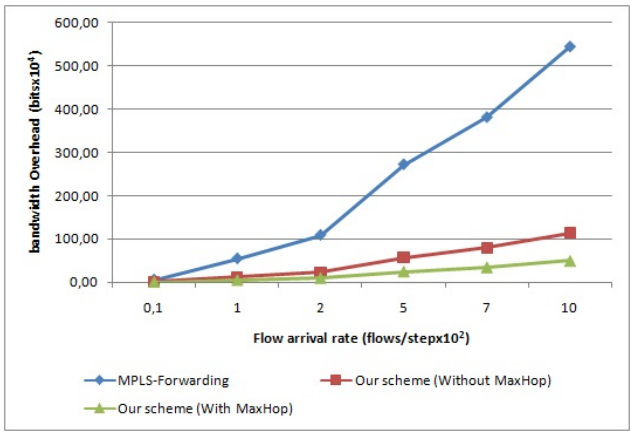

FIGURE 9. Bandwidth overhead as a function of flow arrival rate

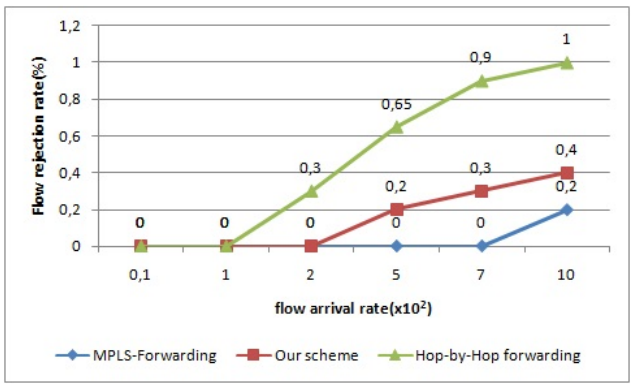

FIGURE 10. Flow rejection rate as a function of flow arrival rate

switches along the flow's routing path.

Figure 10 shows the flow rejection rate for the hop-by-hop forwarding scheme, the MPLS-based forwarding scheme and our scheme. The MPLS-based forwarding scheme outperforms the standard openflow scheme since flow rejection depends only on current flow table usage of the ingress switch. However, the flow rejection on hop-by-hop forwarding scheme depends on the switch having the greatest flow table usage. Our scheme is able to achieve performances of MPLS-based forwarding if the ingress switch has the greatest flow table consumption. Elsewhere, the flow rejection occurs faster than MPLS-based forwarding, but, it is significantly postponed (most than 3 times) compared to hop-by-hop forwarding scheme.

VI. CONCLUSION

This paper proposes an efficient scheme which achieves a trade-off between switch-to-switch communication overhead, controller-to-switch communication overhead and flow table consumption by combining those objectives into a multi-objective optimization problem.

In this paper, both SD-LAN and SD-WAN are considered. In our scheme, the network is divided into several sections. Only one MPLS label is required in each section. We use a pair of binary values to express per-hop forwarding information.

As shown in the results, our scheme performs best compared to the MPLS-based forwarding in terms of bandwidth overhead. It seems also that it can ensure a flow rejection rate close to that offered by the MPLS-based forwarding scheme. More simulations will be done in future work. Our scheme will be compared with parallel schemes to prove its efficiency.

RÉFÉRENCES

[1] Soliman M, Nandy B, Lambadaris I, et al. , *Exploring source routed forwarding in SDN-based WANs* , IEEE International Conference on Communications. IEEE, 2014 :3070-3075.

[2] P. Ashwood-Smith,M. Soliman,W. Tao, *Sdn state reduction* , (IEFT draft).

[3] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, J. Turner,'*Openflow : enabling innovation in campus networks*, ACM SIGCOMM CCR 38, 2008 :69–74

[4] Guo, Zehua, et al. *JumpFlow : Reducing flow table usage in softwaredefined networks*, Computer Networks 92 2015 : 300-315.

[5] X. Dong, Z.Guo, et al., *AJSR : an Efficient Multiple Jumps Forwarding Scheme in Software-Defined WAN*,IEEE Access V5, 2017 :3139-3148

[6] Jain, Sushant, et al. *B4 : Experience with a globally-deployed software defined WAN*. ACM SIGCOMM Computer Communication Review43.4 2013 : 3-14.

[7] Cohen R, Lewin-Eytan L, Naor J S, et al. *On the effect of forwarding table size on SDN network utilization*, INFOCOM IEEE, 2014 :1734-1742.

[8] Kanizo Y, Hay D, Keslassy I. *Palette : Distributing Tables in Software-Defined Networks*, Proceedings - IEEE INFOCOM, 2013 :545-549.

[9] Kang, Nanxi, Liu, Zhenming, Rexford, Jennifer, et al.*Optimizing the one big switch abstraction in software-defined networks*, ACM Conference on Emerging NETWORKING Experiments and Technologies. 2013 :13-24.

[10] Iyer, S. Aakash, Vijay Mann and Naga Rohit Samineni. *Switchreduce : Reducing switch state and controller involvement in openflow networks* IFIP Networking Conference, 2013.

[11] Bidkar S, Gumaste A, Somani A. *A scalable framework for segment routing in service provider networks : The Omnipresent Ethernet approach*, High Performance Switching and Routing (HPSR), 2014 :76-83.

[12] Bidkar, Sarvesh, et al.*Field trial of a software defined network (SDN) using carrier ethernet and segment routing in a tier-1 provider*,Global Communications Conference (GLOBECOM), 2014.

[13] Kreutz D, Ramos F M V, Esteves Verissimo P, et al. *Software-Defined Networking : A Comprehensive Survey*. Proceedings of the IEEE 2014-103 :10-13.

[14] Ching-Lai Hwang, Abu Syed Md Masud (1979), *Multiple objective decision making, methods and applications : a state-of-the-art survey*. Springer-Verlag. ISBN 978-0-387-09111-2. Retrieved 29 May 2012.

[15] Matthias Ehrgott (1 June 2005), *Multicriteria Optimization*. Birkhäuser. ISBN 978-3-540-21398-7. Retrieved 29 May 2012

[16] Carlos A. Coello Coello, Gary B. Lamont, David A. Van Veldhuisen (2007). *Evolutionary Algorithms for Solving Multi-Objective Problems*. Springer. ISBN 978-0-387-36797-2. Retrieved 1 November 2012

[17] Nick McKeown, et al., *OpenFlow : enabling innovation in campus networks*, ACM SIGCOMM CCR, v.38 n.2, April 2008.

[18] Mourad Soliman, et al., *Source Routed Forwarding with Software Defined Control, Considerations and Implications*, ACM CoNEXT Student'12, December 10, 2012 :43-44.