Maximizing Network Utilization in Hybrid Software-Defined Networks

Yannan Hu†, Wendong Wang†, Gong Xiangyang†, Xirong Que‡, Yue Ma‡ and Shiduan Cheng‡
†State Key Laboratory of Networking and Switching Technology, ‡School of Computer Science
Beijing University of Posts and Telecommunications, Beijing 100876, China
E-mail: †{huyannan, wdwang, xygong, rongqx, chsd}@bupt.edu.cn, ‡mayue@bupt.edu.cn

Abstract—By separating the control and forwarding planes, Software-Defined networking (SDN) enables the forwarding paths to be flexibly controlled by the logically centralized controllers using the global network view. To introduce SDN into existing networks, it is necessary to upgrade traditional devices to SDN-enabled ones. However, due to the business, economic and management limitations, it is difficult to realize full SDN deployment. As a result, how to migrate existing devices to SDN-compliant ones becomes the obvious dilemma for every network operator. In this paper, we address this question from the network performance perspective, and study how to leverage the capability of SDN to maximize traffic flow that can be achieved in hybrid SDNs. We formulate the maximum flow problem in networks with partial SDN deployment, and develop a fast Fully Polynomial Time Approximation Scheme (FPTAS) for solving it. Simulation results using real topologies show that hybrid SDNs outperform traditional networks, and we can obtain a near optimal network performance when 50% of SDN nodes are deployed.

I. INTRODUCTION

Software Defined Networking (SDN) is an emerging networking paradigm that separates control and forwarding planes in communication networks [1], [2]. By using a standardized interface between the two (e.g., OpenFlow), SDN enables the forwarding devices (e.g., switches) and traffic forwarding paths to be flexibly controlled by (logically) centralized controllers. Due to its promises to simplify management and improve programmability of networks, SDN has gained a lot of attention, and already lead to real deployment. For example, in data center environment, Google used an OpenFlow-based Software-Defined Network to interconnect its data centers across the planet due to the ease and flexibility in performing traffic engineering functions, which drives link utilization to nearly 100% [3]. Similarly, Microsoft also designed a software-driven WAN for its data centers and achieved a high network utilization [4]. These benefits provide a great incentive to the migrations to Software-Defined Networks.

However, for a medium/large-scale ISP network that consists of a large number of traditional devices, simply upgrading all the routers at once is a daunting task, since one could encounter business, economic and technical challenges or obstacles [5]. As a result, gradually migrating to SDN is a better and viable approach, where the SDN-compliant devices are deployed incrementally. During the migration process, the network will be in a hybrid mode for a relatively long time, where the traditional and SDN devices coexist in the hybrid Software-Defined Networks.

In a hybrid SDN, only the SDN devices can be controlled by the centralized controller, while the existing non-SDN nodes still use the traditional shortest path routing protocol, such as OSPF, to forward packets. In other words, not all the traffic can be flexibly controlled by the controller. As a result, the challenge is how to leverage the centralized control capability of SDN controller to tune the traffic distribution in the network, and reap the benefit of partial SDN deployment.

To this end, this paper studies the traffic engineering problem in hybrid SDN network, concentrating on satisfying global network performance objectives. Given the set of SDN nodes, our goal is to find the maximum flow that can be achieved by only tuning the forwarding behaviors of the SDN devices. We formulate the maximum flow problem in networks with partial SDN deployment, and develop a fast Fully Polynomial Time Approximation Scheme (FPTAS) for solving it. Through extensive simulation using real network topologies, results show that hybrid SDNs perform better than traditional OSPF networks. We also demonstrate the impact of the SDN node number on network maximum flow, and find that 50% of SDN deployment could lead to a near optimal performance gain.

The reminder of this paper is structured as follows. Section II introduces related work. Section III describes the hybrid SDN network model, and formulates the hybrid SDN maximum flow problem. In Section IV, we present a Fully Polynomial Time Approximation Schemes (FPTAS) for the maximum flow problem, which is a prime-dual algorithm. Section V presents the simulation study, and finally Section VI concludes the paper.

II. RELATED WORK

Since the emergence of SDN, the coexistence of SDN and traditional network has received increasing attention. To enable remote IP routing services in SDN, the authors of [6] proposed RouteFlow, through which the interconnection between SDNs and traditional networks can be realized.

In [7], [8], the authors proposed a hybrid SDN network planning and operating architecture, Panopticon. Similarly, Lu et al. proposed an automated network management framework for hybrid SDNs, called HybNET[9], which provides a common configuration mechanism for both SDN and legacy switch. Still in this vein, the authors of [10] proposed a hybrid...
SDN management approach called fibbing, which enables centralized control of the distributed computation of forwarding paths by injecting “fake” link-state advertisements into the network.

In [11], Vissicchio et al. found that the presence of SDN and traditional routing protocols can create forwarding anomalies. As a result, anomaly-free network update techniques for hybrid SDNs are proposed. Hybrid SDN network is a network node that the centralized and distributed control planes coexist at the same time. The authors of [12] thus developed a theoretical framework to reason about the coexistence of arbitrary control planes. However, none of the above work considered the performance of hybrid SDNs.

In [13], Agarwal et al. addressed network performance issues in an incrementally deployed hybrid SDNs, and investigated how to leverage SDN to dynamically manage traffic so as to reduce packet losses and delays. Similarly, the traffic engineering problem in SDN/OSPF networks is also studied in [14]. Besides considering flow splitting at the SDN nodes, the paper also jointly optimized the OSPF weight settings to obtain a lower maximum link utilization. Another study in [15] proposed a two-stage algorithm that optimizes the SDN migration sequence with the goal of providing large number of path alternatives and reducing the need for network capacity upgrades. Complementarily, this paper investigates the network migration problem from the perspective of maximum flow that can be achieved.

Finally, several hybrid SDN models that combine SDN with traditional network approach are explored in [16]. The authors also exposed qualitative tradeoffs between different hybrid SDN models and further outlined research challenges and open problems in hybrid SDN networks.

III. NETWORK MODEL AND PROBLEM FORMULATION
A. System Description

This paper focuses on hybrid SDNs with partial SDN deployments. The forwarding behaviors of SDN nodes are controlled by the logically centralized controller, while the rest of the network are traditional devices that run standard distributed routing protocols like OSPF and forward packets along the shortest paths. Note that we only consider IP network, not network technologies with built-in traffic engineering capabilities, such as MPLS-TE. Besides, for the sake of routing stability, network operators rarely deploy OSPF link weight changes. Therefore, as opposite to [14], we do not consider improving network performance through weight changes. To enable inter-operations between SDNs and traditional networks, we assume that the SDN controller can receive/send OSPF link-state messages from/to non-SDN devices using mechanisms similar to [6], and provide traditional routing services for hosts connecting to SDN devices. This also enables the SDN controller to know the overall network topology and traffic forwarding paths according to OSPF weights. Finally, for the need of traffic engineering, suppose the SDN controller is able to obtain network traffic information using the following approaches: (1) Inquire statistical data on SDN devices (e.g., the counter fields in OpenFlow flow tables), and determine the amount of traffic between SDN devices and other devices in the network; (2) Directly or indirectly collect load information on each link using SNMP or other technologies. Based on this information, the controller need to compute forwarding rules for SDN nodes so as to tune the traffic distribution in the network.

B. Network Model

We denote the physical network by a graph \( G(V, E) \), where \( V \) is the set of nodes and \( E \) is the set of links. Let \( S \) be the set of SDN nodes, and \( D = V \setminus S \) be the set of traditional nodes. For each link \( e \in E \), let \( c(e) \) be the capacity of \( e \). For each SDN node \( s \in S \), we use \( T_{sd} \) to represent the amount of traffic for destination \( d \in V \) that either originates from or passes through \( s \). As mentioned above, \( T_{sd} \) can be computed by the SDN controller using traffic statistic obtained from the SDN devices. For node \( u, d \in V \), let \( NH(u, d) \) be the next hop of \( u \) on the shortest path between \( u \) and \( d \).

In a hybrid SDN network, as the controller can only change the forwarding behaviors of SDN nodes and manipulate the traffic that pass through them. As a result, we have the following definitions.

**Definition 1: (Controllable Path):** In a hybrid SDN network, given the set of SDN nodes \( S \), if there exists a loop-free path \( s = u_0, u_1, u_2, ..., u_k = d \) from a SDN node \( s \in S \) to a destination \( d \in V \) such that the following two conditions hold, the path is called a controllable path:

1) For any \( j = 1, 2, ..., k, (u_{j-1}, u_j) \in E \).
2) If \( u_{j-1} \in D \), then \( u_j = NH(u_{j-1}, d) \).

Definition 1 implies that when a packet needs to be forwarded from a SDN node to a destination, the path it takes should be loop-free and the next hops to the destination for all non-SDN nodes should follow the shortest path routing. We use \( P_{sd} \) to represent the set of controllable paths between SDN node \( s \) to destination \( d \).

**Definition 2: (Controllable traffic):** In a hybrid SDN network, if a traffic flow originates from a SDN node, or there is at least one SDN node on the shortest path between the source and destination of it, such traffic is called controllable traffic. Otherwise, it called uncontrollable traffic.

In other words, if all traffic is forwarded along the shortest paths, Definition 2 implies that the traffic that traverses at least one SDN node is controllable. Therefore, we have at least one chance to change the path of a controllable traffic flow at a SDN it traverses. However, due to flexibility of SDN, after transiting the first SDN node, a controllable traffic flow may not encounter another SDN node. As a result, for each SDN node, it can only manipulate the traffic that definitely pass through it, we thus have the following definition.

**Definition 3: (SDN Controllable traffic):** In a hybrid SDN network, for a SDN node \( s \in S \), there exists a traffic flow to destination \( d \in V \) such that the following two conditions hold, such traffic is called SDN controllable traffic of \( s \):
1) The traffic is controllable traffic.
2) The traffic passes through \( s \) before it passes through any other SDN nodes.

Let \( W_{sd} \) be the SDN controllable traffic of node \( s \in S \) to destination \( d \in V \). According to Definition 3, if the controller knows the traffic matrix of the network, the value of \( W_{sd} \) can be computed by adding the following two types of traffic: 1) the amount of traffic that originates from \( s \) to \( d \), and 2) the amount of traffic to \( d \) that passes through \( s \) before passes through other SDN nodes. However, in practice, the controller can only obtain the amount of traffic to \( d \) that either originates from or passes through \( s \), which is \( T_{sd} \), by inquiring traffic statistic from SDN node \( s \). Since \( T_{sd} \) not only contains the amount of traffic that originates from \( s \), but also may includes traffic from other SDN nodes that passes through \( s \), \( W_{sd} \) is a subset of \( T_{sd} \). As a result, the controller needs to deduce the non-measurable value \( W_{sd} \) from the measurable value \( T_{sd} \). For this purpose, this paper leverages the technique proposed in [13]. The key insight is to eliminate the redundancy between different \( T_{sd} \). We refer readers to [13] for more details.

C. Problem Formulation

One goal of this paper is find the maximum flow that can be achieved in hybrid SDNs. This section presents the formulation of the hybrid SDN max-flow problem. In a hybrid SDN network, for each destination \( d \in V \), the controller can only manipulate the SDN controllable traffic \( W_{sd} \) from SDN node \( s \in S \) to \( d \), and the traffic has to be transmitted along the controllable paths in \( P_{sd} \). Let \( P \) be the set of all controllable paths, i.e., \( P = \bigcup_{(s,d) \in P_{sd}} P_{sd}, s \in S, d \in V \). Let \( f(e) \) be the total amount of traffic on link \( e \), and \( r(e) \) be the amount of uncontrollable traffic on link \( e \). According to [13], \( r(e) \) can be computed by removing all SDN controllable traffic on \( e \) from \( f(e) \). The controller then needs to route the SDN controllable traffic along controllable paths so as to maximize the traffic flow. For a controllable path \( p \in P_{sd} \) between \( s \) and \( d \), let variable \( x(p) \) be the flow in path \( p \). We have the following linear optimization problem:

\[
\text{Maximize } \sum_{p \in P} x(p) \quad \text{(Max-Flow)}
\]

Subject to:
\[
\begin{align*}
    r(e) + \sum_{p \in P} x(p) &\leq c(e) \quad \forall e \in E \quad (1) \\
    \sum_{p \in P_{sd}} x(p) &\leq W_{sd} \quad \forall s \in S, d \in V \quad (2) \\
    x(p) &\geq 0 \quad \forall p \in P \quad (3)
\end{align*}
\]

Equation (1) ensures that the total flow on any link is no more than the link capacity. Equation (2) states that the sum of flows over all controllable paths between any SDN node \( s \) and destination \( d \) is at most the amount of SDN controllable traffic of \( s \) to \( d \). Equation (3) ensures that the flow on any controllable path is non-negative.

IV. Algorithm

This section presents an algorithm for the Max-Flow problem. Note that the number of paths can be exponential in the number of nodes, the linear program has an exponential number of variables and thus cannot be solved directly in an efficient manner. We therefore develop a Fully Polynomial Time Approximation Scheme (FPTAS) to solve the Max-Flow problem. An FPTAS for a maximization problem is a family of algorithms that gets to within \((1-\epsilon)\) of the optimum for any \( \epsilon > 0 \), in time polynomial in the problem input and \( 1/\epsilon \). The FPTAS used in this paper is a primal-dual algorithm.

A. The Primal-dual Algorithm

In the dual linear program, there is a dual variable \( l(e) \) for each link capacity constraint (Equation (1)), and a dual variable \( u_{sd} \) for each SDN controllable traffic constraint (Equation (2)). The dual problem (Max-Flow Dual) is as follows.

Minimize \( \sum_{e \in E} q(e) \cdot l(e) + \sum_{s \in S, d \in V} W_{sd} \cdot u_{sd} \quad \text{(Max-Flow Dual)} \)

Subject to:
\[
\begin{align*}
    \sum_{e \in P} l(e) + u_{sd} &\geq 1 \quad \forall p \in P \quad (4) \\
    l(e), u_{sd} &\geq 0 \quad \forall e \in E \text{ and } d \in V \quad (5)
\end{align*}
\]

where \( q(e) \) is the remaining link capacity after removing the uncontrollable traffic \( r(e) \) from the link capacity \( c(e) \). The dual problem can be visualized as assigning costs \( l(e) \) and \( u_{sd} \) for each link \( e \) and each SDN controllable traffic \( W_{sd} \) respectively, such that \( D(l, u) \) is minimum.

Let \( length_{sd}(l, u) \) be the cost of the shortest path with respect to the dual variables \( l \) and \( u \). Define \( \alpha(l, u) \) as cost of the minimum cost path between any SDN node and other nodes. Then, the Max-Flow Dual problem is equivalent to find a cost setting \((l, u)\) such that \( D(l, u)/\alpha(l, u) \) is minimum. Let \( \beta \) be the minimum value, i.e., \( \beta \leq \min_{(l, u)} D(l, u)/\alpha(l, u) \).

We now describe a Fully Polynomial Time Approximation Scheme to solve the Max-Flow and Max-Flow Dual problems as shown in Algorithm 1. The algorithm first computes a value \( \delta \) according to the desired accuracy level \( \epsilon \), the number of links and the number of SDN controllable traffic. The dual costs of each link \( e \in E \) and each SDN controllable traffic \( W_{sd} \) are initialized to \( l(e) = \frac{\delta}{q(e)} \) and \( u_{sd} = \frac{\delta}{W_{sd}} \), respectively. The algorithm then operates iteratively. In each iteration, it first computes the shortest controllable path \( p \) between SDN nodes and other nodes under the dual costs \( l \) and \( u \), and then updates the variables related to \( p \) in the primal and dual problems as follows. For the primal problem, the algorithm augments flow along the path \( p \). The amount of flow \( \mu \) sent along the path is determined by the minimum remaining link capacity on \( p \) and the controllable traffic between the two terminals of the path. As a result, the primal variable \( x(p) \) and the primal solution increase by \( \mu \), respectively. After updating the primal
variables, the algorithm continues to update the dual variables, which ensures that the value of the minimum dual cost related to path $p$ increases by a factor of $(1 + \epsilon)$. The algorithm stops after $t$ iterations such that $D(l, u) \geq 1$.

**Algorithm 1: Algorithm for Max-Flow and Max-Flow Dual problems**

**Data:** network $G$, remaining link capacities $q(e)$, all node controllable traffic $W_{sd}$, approximation accuracy $\epsilon$

**Result:** primal solutions $x$

$$\delta = (1 + \epsilon)/(1 + \epsilon) \cdot m^{1/2}$$

$p(e) \leftarrow \pi q(e)$, $\forall e \in E$;

$u_{sd} \leftarrow \frac{W_{sd}}{\epsilon}, \forall(s, d), s \in S, d \in V$;

$x(p) \leftarrow 0, \forall p \in P$;

while $D(l, u) < 1$ do

  for each node $p$: shortest controllable path between any SDN node and other nodes under dual costs $l$ and $u$;

  $\mu = \min(\min_{e \in P} q(e), W_{sd})$

  $x(p) = x(p) + \mu$;

  $l(e) = l(e)(1 + \frac{\mu}{q(e)}), \forall e \in p$;

  $u_{sd} = u_{sd}(1 + \frac{\mu}{W_{sd}}), p \in P_{sd}$;

  Recompute $D(l, u)$;

end

return $x$;

**Algorithm 2: Algorithm for computing shortest controllable path with dual costs**

**Data:** network $G$, SDN node set $S$, dual edge cost $l(e)$, dual traffic cost $u_{sd}$ and shortest path set under OSPF weights

**Result:** shortest controllable path under edge cost $l(e)$ and traffic cost $u_{sd}$

for each node $d \in V$ do

  $T_d \leftarrow$ the directed shortest path set from all other nodes to $d$;

  for each edge $(s, j)$, $s \in S$ do

    if $j$ is a SDN node then
      $T_d(s, j) \leftarrow 1$; $T_d(j, s) \leftarrow 1$;
    end

  else

    if there is no directed edge from $i$ to $j$ in $T_d$ then
      $T_d(s, j) \leftarrow 1$;
    end

  end

end

$p_d \leftarrow$ the shortest controllable path under edge cost $l(e)$ and traffic cost $u_{sd}$ in $T_d$ using a variant of Dijkstra’s algorithm;

$p^* \leftarrow$ the shortest controllable path among all $p_d$;

In each iteration, Algorithm 1 needs to compute the shortest controllable path between SDN nodes and other nodes under dual costs $l$ and $u$, i.e.,

$$\arg\min_{p \in P_{sd}, s \in S} \{\sum_{e \in p} l(e) + u_{sd}\}$$  \hspace{1cm} (6)

As the non-SDN nodes on any controllable path still forward packets along the shortest path under OSPF weights, we solve the problem of (6) using use Algorithm 2. First, for each destination $d$, the algorithm computes a directed shortest path tree from all SDN nodes to $d$ using the OSPF weights, denoted as $T_d$. In $T_d$, $T_d(i, j) = 0$ indicates that there is no directed edge between $i$ and $j$, while $T_d(i, j) = 1$ represents the edge direction of $(i, j)$ is from $i$ to $j$. After that, for each SDN node $s \in S$, the algorithm examines all links incident to it, and adds directed edges in $T_d$ as follows. If the other terminal of edge $(s, j)$, which is node $j$, is also a SDN node, it adds a bidirectional edge in $T_d$; If $(s, j)$ is not in $T_d$, the algorithm adds a directed edge from $s$ to $j$ in $T_d$. Then, by using a variant of Dijkstra’s algorithm with dual cost $(l, u)$ on $T_d$, the algorithm finds the shortest controllable path from any SDN node to $d$. Finally, it outputs the controllable path with the minimum cost among all the shortest ones to different destinations.

**B. Algorithm Analysis**

The above algorithm is analysed using the techniques in [17]. For every iteration $i \geq 1$, the dual objective is

$$D^i(l, u) = \sum_{e \in E} q(e) \cdot l^i(e) + \sum_{s \in S, d \in E} W_{sd} \cdot u_{sd}^i$$

$$= \sum_{e \in E} q(e) \cdot l^i-1(e) + \sum_{s \in S, d \in E} W_{sd} \cdot u_{sd}^{i-1} + \sum_{e \in P} \epsilon \cdot l^i-1(e) + \sum_{e \in P} \epsilon \cdot u_{sd}^{i-1}$$

$$= D^{i-1}(l, u) + \epsilon \cdot (f^i - f^{i-1}) \cdot \alpha^{i-1}(l, u)$$  \hspace{1cm} (7)

where $f^i$ is the primal objective after the $i$th iteration. According to Equation (7), thus

$$D^k(l, u) = D^0(l, u) + \epsilon \cdot \sum_{i=1}^k (f^i - f^{i-1}) \alpha^{i-1}(l, u)$$  \hspace{1cm} (8)

Since $\beta \doteq \min_{l, u} D(l, u)/\alpha(l, u)$, we have

$$\beta < D^{i-1}(l, u)/\alpha^{i-1}(l, u)$$  \hspace{1cm} (9)

Then,

$$D^k(l, u) \leq m \cdot \delta + \frac{\epsilon}{\beta} \sum_{i=1}^k (f^i - f^{i-1}) D^{i-1}(l, u)$$  \hspace{1cm} (10)

where $m$ is the sum of the number of links and the number of SDN controllable traffic, and thus

$$D^k(l, u) \leq m \cdot \delta + \frac{\epsilon}{\beta} \sum_{i=1}^k (f^i - f^{i-1}) D^{i-1}(l, u)$$

$$\leq D^{k-1}(l, u) + \epsilon \cdot (f^k - f^{k-1}) D^{k-1}(l, u)$$

$$= (1 + \frac{\epsilon}{\beta} \cdot (f^k - f^{k-1})) \cdot D^{k-1}(l, u)$$  \hspace{1cm} (11)

$$\leq e^{\frac{\epsilon}{\beta}f^{k-1}} \cdot D^{k-1}(l, u)$$

$$\leq e^{\frac{\epsilon}{\beta}k} \cdot D^{0}(l, u) = e^{\frac{\epsilon}{\beta}k} \cdot m \cdot \delta$$

By the stopping condition, let $t$ be the last iteration,

$$1 \leq D^t(l, u) \leq e^{\frac{\epsilon}{\beta}t} \cdot m \cdot \delta$$  \hspace{1cm} (12)
and hence
\[ \frac{f^t}{\beta} \geq \frac{\ln \frac{1}{m^2}}{\epsilon} \] (13)

**Lemma 1:** Scaling the final solution obtained using Algorithm 1 by \( \log_{1+\epsilon} \frac{1+\epsilon}{\delta} \) is a feasible solution to the Max-Flow problem.

**Proof.** Every time the flow on the shortest controllable path increases by \( \mu \), the dual costs \( l \) and \( u \) increase by a factor of \( (1 + \frac{\mu}{W_{sd}}) \) and \( (1 + \frac{\mu}{W_{sd}}) \), respectively. Since \( \frac{\mu}{W_{sd}} \cdot \frac{\mu}{W_{sd}} \leq 1 \), when the left-hand-side of any constraint of the primal problem increases by a factor of 1, the dual costs will at most increase by a factor of \( 1 + \epsilon \). By the stopping condition, \( D^t(l, u) < 1 \). Thus \( l^{t-1}(e) < \frac{1}{q(e)} \) and \( u^{t-1}_e < \frac{1}{q(e)} \). Therefore, when the algorithm stops we have \( l^t(e) < \frac{1}{q(e)} \) and \( u^t_e < \frac{1}{W_{sd}} \). Initially \( l^0(e) = \frac{\delta}{q(e)} \) and \( u^0_e = \frac{\delta}{W_{sd}} \), so any constraint of the primal problem cannot be violated by a factor of \( \log_{1+\epsilon} \frac{1+\epsilon}{\delta} \).

Since the primal and dual problems have the same optimal solution, the approximation rate \( \lambda \) of Algorithm 1 is \( \frac{f^t}{\beta \cdot \log_{1+\epsilon} \frac{1+\epsilon}{\delta}} \). According to Equation (13), we have
\[ \lambda \geq \frac{\ln \frac{1}{m^2}}{\epsilon \cdot \log_{1+\epsilon} \frac{1+\epsilon}{\delta}} = \frac{\ln(1 + \epsilon) \cdot \ln \frac{1}{m^2}}{\epsilon} \]

Let \( \delta = \frac{1+\epsilon}{(1+\epsilon) \cdot m^2} \), then
\[ \lambda \geq \frac{(1 - \epsilon) \cdot \ln(1 + \epsilon)}{\epsilon} \geq \frac{(1 - \epsilon) \cdot (\epsilon - \frac{\epsilon^2}{2})}{\epsilon} \geq (1 - \epsilon)^2 \] (14)

**Lemma 2:** Algorithm 1 stops after \( O(\epsilon^{-2} \cdot m \cdot \log m) \) iterations.

**Proof.** Since \( l^0(e) = \frac{\delta}{q(e)} \) and \( u^0_e = \frac{\delta}{W_{sd}} \), and \( l^t(e) < \frac{1+\epsilon}{q(e)} \) and \( u_e^t < \frac{1}{W_{sd}} \), each dual cost can be increased at most \( \log_{1+\epsilon} \frac{1+\epsilon}{\delta} \) times. Since the total number of dual variables is \( m \), the algorithm iterates at most \( O(m \cdot \log_{1+\epsilon} \frac{1+\epsilon}{\delta}) \) times. Since \( \delta = \frac{1+\epsilon}{(1+\epsilon) \cdot m^2} \), the number of possible iterations is at most \( O(\epsilon^{-2} \cdot m \cdot \log m) \).

Finally, the complexity of Algorithm 2 is \( O(|V| T_{Dijkstra}) \), where \( T_{Dijkstra} \) is the run time of Dijkstra’s algorithm. According to Equation (14) and lemma 2 we get the following theorem.

**Theorem 1.** There exists an algorithm that computes a \((1 - \epsilon)^2\)-approximation to the Max-Flow problem in time \( O(\epsilon^{-2} m |V| T_{Dijkstra} \log m) \).

V. EVALUATION

A. Experiment Setup

We conduct simulation experiments on the following three SNDlib topologies [18]: (1) The France topology, which contains 25 nodes, 45 links and 300 traffic flows; (2) The Cost266 topology, which contains 37 nodes, 57 links and 1332 traffic flows; and (3) The Germany50 topology, which contains 50 nodes, 88 links and 662 traffic flows.

For each topology, we use the reported link capacity values as the link capacities, and assume the link weights to be the inverse of link capacities. We then carry out two sets of experiments on all the three topologies. The first set of experiments use traffic matrixes provided by SNDlib to investigate the impact of SDN node number on the network performance. We then generate random traffic matrixes to examine the impact of traffic matrix variations on network performance when the locations of SDN nodes are fixed.

B. Impact of SDN node number on network performance

This section investigates the impact of SDN node number on network performance from the maximum flow perspective. The locations of a given number of SDN nodes are computed based on the reported traffic matrix for each SNDlib topology, using greedy algorithm that is similar to the approach in [13], [14]. Specifically, this approach iteratively deploys SDN nodes one after another until the required number of SDN nodes are deployed. In each iteration, it picks the set of locations that can achieve the maximum flow. To facilitate comparison across topologies, we use normalized maximum flow as a performance metric. The normalized maximum flow is defined as
\[ \frac{\text{MaxFlow}(C)}{\text{MaxFlow}(N)} \]

Clearly, the value of normalized maximum flow is at most one.

![Fig. 1. Effect of SDN node number](image-url)

Figure 1 shows the relationship between the SDN deployment rate and the maximum flow on different topologies, where the rate of 0 corresponds to the case of traditional OSPF network. From this figure, we can see that the maximum flow increases with the increasing of deployment rate. When the deployment rate is in the range of 0% – 50%, the maximum flow increases rapidly, while deploying more SDN nodes could eventually lead to diminishing returns. Compared with OSPF routing, when SDN is 50% deployed we observe 10.51% - 43.08% improvement in the maximum flow across topologies, which is near optimal.

C. Effect of traffic matrix variation

In the first set of experiments, we choose the locations of SDN nodes according to the reported traffic matrixes in SNDlib topologies. To test the impact of traffic variations on network performance when the SDN locations are determined, we fix the deployment rate to 10%, 30% and 50% on each
topology, and examine the maximum flow that can be achieved under 20 different randomly generated traffic matrices.

Figure 2 plots the normalized maximum flow on all three topologies for each of the 20 experiments. We can see that the hybrid SDNs perform better than OSPF routing for all the experiments, and the maximum flow increases with the deployment rate. When deploying 10%, 30% and 50% SDN nodes, we observe averagely 8.58%, 30.11% and 36.77% improvement in the maximum flow on France topology, 1.29%, 5.44% and 7.3% on Cost266 topology, and 3.44%, 15.92% and 23.65% on Germany50 topology, respectively.

VI. CONCLUSION

While SDN promises to ease design, management and operation of communication networks, it is nearly impossible to be fully deployed in short term due to business, management and technical obstacles. Accordingly, this paper studied how to improve the maximum flow that can be achieved in hybrid SDN networks, where SDN are incremental deployed. We formulated the maximum flow problem in hybrid SDNs, and developed a Full Polynomial Time Approximation Scheme (FPTAS) to solve it. Experiment results using real topologies indicated that hybrid SDNs significantly perform better than traditional OSPF networks in the context of network maximum flow. We also showed that when 50% of nodes are SDN-enabled, a near optimal network performance can be realized.

ACKNOWLEDGMENT

This work partially supported by the National High Technology Research and Development Program (863 Program) of China (Grant No. 2013AA013303 and No. 2013AA013301), and the National Natural Science Foundation of China (Grant No. 61471055).

REFERENCES


