Migrating to SDN: Achieving Near-Optimal Traffic Engineering in Hybrid Software Defined Networks

***Abstract*—Software Defined Networking (SDN) is an emerging networking paradigm which intends to merge networks into the age of the cloud, providing fine-grained control, simplified configurations, unprecedented flexibility and seamless scalability. However, due to the large set of unresolved challenges as well as the deployment cost, network evolution to fully SDN systems will take a long time. In fact, SDN elements are incrementally deployed in enterprise networks, producing a transitional network form of hybrid SDN (H-SDN). H-SDN consists of traditional networking elements and SDN elements, accommodating both conventional traffic and SDN traffic. In this paper, we investigate traffic engineering (TE) in an H-SDN system, where the SDN controller attempts to strategically route SDN traffic so as to optimize the TE performance over all in-system links shared with uncontrollable conventional traffic. Two hybrid modes are studied: (1) the *barrier* mode, where link capacities are split between the two forms of traffic by barriers; and (2) the *hybrid* mode, where the controller needs to route SDN traffic in accordance with the dynamic link occupancies by traditional traffic. We propose fast algorithms for the TE problems in both scenarios with provable approximation guarantees. Theoretical analysis and computer simulations validate the efficacy of our algorithms.**

***Index Terms*—Traffic engineering, software defined networks, hybrid SDN, flow optimization, request routing**

I. Introduction

A. H-SDN: The Transitional Form to The New Norm

By using distributed network protocols and vertically integrated routers/switches, traditional IP networks can effectively deliver information in the form of data packets and have successfully supported the explosive traffic growth in communication networks. However, the highly integrated networking functions, together with the absence of global network view and limited configurabilities by operators makes IP networks complex, ossified and thus hard to manage and evolve [1].

Software Defined Networking (SDN) is an emerging disruptor to the traditional networking paradigm, which unleashes a powerful new paradigm offering flow-level traffic control and programmable interfaces to network operators. SDN promises the ease of network design, operation and management and therefore, breaks through the bottleneck to the acceleration towards cloud computing that networking is evolving to. By decoupling the control framework from the data-plane functionalities, SDN supports centralized network control with global network view, complements network function virtualization (NFV) [2] and unlocks the latent potential of the network fabric. In conjunction with the growing support of SDN protocols like OpenFlow [3] and the continuous development of NFV, SDN is becoming a new norm of networks, which not only gains significant research attention, but also leads the majority of network operators to jump on the SDN bandwagon to exploit its early success [4], [5].

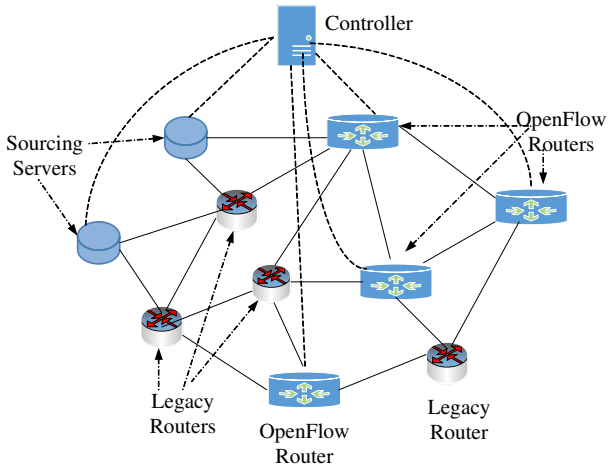


Figure 1. Architecture of a Hybrid SDN

However, as a newly born networking paradigm, SDN itself comes along with a large set of unresolved challenges [6], [7]. On the other hand, rather than a disruptive new technology, mature networks with tested functionalities and proven success are more attractive to network operators. Due to these technical and economical concerns, network operators are less likely to upgrade their entire networks to SDN. Instead, as a matter of fact, SDN elements are incrementally deployed to co-work with traditional networking elements, resulting in a transitional form of networks, *i.e.,* hybrid SDN (H-SDN) [8].

The architecture of an H-SDN system is depicted as Fig. 1. The SDN elements and flows are controlled by the (logically) centralized SDN controller, while legacy flows are uncontrolled traffic to SDN controller which are routed by legacy routers and possibly need to pass some OpenFlow routers. Furthermore, virtualized data sources are integrated by the SDN setup as shown in the figure, which physically could be in-network data sources, caching proxies or sourcing points that are further connected to content providers networks, such as remote content distribution networks (CDNs) or clouds.

B. Traffic Engineering in H-SDN

Most traditional networks are Commercial Off-The-Shelf (COTS) based systems, in which the networking elements are designed, supplied and evolved by hardware vendors while limited configurable options are provided to network operators. For the ease of presentation, henceforth, we term such traditional design and management model as COTS Networking (CN) and refer to the corresponding routers/switches therein as CN elements.

Traffic engineering (TE) aims to distribute traffic so as to optimize certain performance criterion, such as maximum link

congestion level, sum link cost, and maximum concurrent flow. Stability has been of great importance to the network performance, ever since communication networks started [9]. In this work, we adopt the natural and most popular optimization goal, *i.e.,* to minimize the maximum link utilization, and make traffic engineering and such goal interchangeable in this paper.

As a transitional form of networks, H-SDN inherits both advantages and disadvantages from CN and SDN. It requires extra caution to update elements and design new features in H-SDN [10]. Inappropriate deployment of an H-SDN system will result in a more complex network with severely degraded performance for packet delivering. Given the fact that TE remains intractable in CN [11] as well as in SDN [12], enabling TE in H-SDN is more challenging.

In an H-SDN system, CN traffic and SDN traffic co-exist and share the link capacities. To ensure the network stability, we assume no changes to the existing routing solutions to CN traffic, *e.g.,* Open Shortest Path First (OSPF) [13] or IS-IS [14], which have proven efficiency in delivering packets. We carefully design the routing protocol for SDN flows so that they pass through both SDN elements and CN elements in an efficient, friendly and elegant manner.

With virtualized data sources as shown Fig. 1, SDN also offers the network controller the capability to select sources for traffic demands. For instance, when an SDN node initiates a traffic demand (possibly for some specific object), not only will the flow be distributed among paths, but also the demand can be fulfilled by several sources to further improve the network performance. Therefore, the TE solution in H-SDN also needs to consider the source redirection problem, *i.e.,* how to split the demand among sources.

To support TE in H-SDN, we need to jointly counter the flow routing and source redirection problem and strategically route SDN traffic while maintaining the advantages of both CN and SDN elements.

C. Contributions

In summary, the contributions of this paper are three-fold.

Firstly, to the best of our knowledge, this is the first article to investigate the traffic engineering in hybrid SDN systems. The fine-grained control capability of SDN elements is exploited to further distribute SDN flows among sources, while routing configurations at CN elements are gently enhanced to accommodate the SDN flows. This paper shows an elegant attempt to orchestrate two types of traffic in the transitional H-SDN without losing the major advantages from either side.

Secondly, we analyze the routing protocols of CN and SDN, and propose a novel routing protocol for passing SDN traffic through CN elements. It is a simple hop-by-hop protocol with destination based forwarding and traffic aggregation, which integrates the efficiency of CN and the manageability of SDN. We show that the protocol is compatible with SDN protocols and can be easily supported by CN elements.

Thirdly, based on the new protocol, we model the TE problems in two modes of H-SDN, a barrier mode and a hybrid mode. The barrier mode is a conservative method for deploying H-SDN, where a certain fraction of capacity over each link is reserved for SDN traffic such that the incrementally deployed SDN elements operate in an overlay network, logically isolated from the existing CN system. In the hybrid mode, the two types of traffic fully share the delivering capacities of the network. We extensively study the formulated TE problems and propose fully polynomial-time approximation schemes (FPTASs) for them. For both modes of H-SDN, our algorithms achieve a provably approximate guarantee of $(1+\omega)$ with computational complexity proportional to ω^{-2}. Theoretical analysis and numerological studies are then carried out to evaluate the efficacy of our algorithms.

The rest of this paper is organized as follows. Section II presents related works. Our design rationale and a new routing protocol are presented in Section III. In Section IV, we describe the system model, propose the optimization framework and analyze the hardness of the formulated problems. Section V and Section VI present our solutions to the problems introduced in Section III. Simulation studies are presented in Section VII. Finally, Section VIII concludes the paper.

II. Related Works

Traffic engineering has been of significant importance in flow optimization ever since networks started. We refer readers to [9] for the full story and thorough discussions on the topic. On the other hand, SDN has attracted widespread interests recently, from both academia and industry [8], [15].

Network migration from CN to SDN is extensively analyzed in [16] both technically and economically, which concludes that the coexistence of SDN and traditional elements in commercial networks could probably last for decades [8]. Despite the attractive prospect that SDN has promised, the hybrid mode inevitably brings more complicated network management issues to network operators. In [10], Vissicchio *et al.* revealed that the forwarding anomalies created by co-existence of SDN and CN routing protocols might eventually defeat the purpose of deploying SDN. Therefore, for the sake of stability and manageability, in short-term deployment, barriers are tentatively setup for separating SDN traffic and traditional CN traffic. This confirms the motivation of this study.

Traditional flow optimization works in a lower layer of the network, *e.g.,* OSPF [13] at the link layer. Merging to cloud, virtualized data sources are integrated in the SDN [17], [18]. This enables the SDN controller to manage data sources and accordingly to select data sources for SDN requests. Therefore, SDN supports fine-grained traffic control, *e.g.,* source redirection at the application layer. For SDN requests, the controller can gather the detailed information of the remote requests such that not only can the flow routing be optimized, but also the sources of each request can be strategically selected. Joint source redirection and flow routing are first investigated in [19]. However, the schemes therein are designed for mobile networks and the joint problem is formulated on an overlay network without background traffic. In this work, we go beyond the mobile routing problem investigated in [19], while we consider a more practical scenario in the hybrid SDN system with uncontrollable background traffic.

A similar work on traffic engineering in SDNs is carried

Rule	Priority	Source	Dest.	Actions
2	.	*	D	

Figure 2. Enhanced routing table at the SDN Forwarding Element.

out in [12], where the authors also studied a hybrid form of SDN. However, there are several essential differences between [12] and this work. Firstly, they focused on the deployment of SDN networking elements, while we focus on the hybrid traffic. Secondly, they assumed no changes to be made to non-SDN nodes and only attempted to optimize the traffic passing through SDN elements. In this work, we argue that the slight modified routing protocol can be easily implemented in SDN routers as well as CN routers. Instead of considering a routing table with specific traffic in [12], our protocol adds a factor of forwarding fraction for distinct destination-based traffic. We also consider the joint flow optimization together with source redirection. Lastly and most importantly, the TE problem formulated in [12] was solved by being converted to a deemed "equivalent" problem. However, by a counterexample, we show that the conversion is actually not equivalent in most scenarios. We then propose fast approximate algorithms with theoretical correctness guarantees. Therefore, this work complements [12] by considering more practical TE models and solutions.

III. Routing Design: A Tale of Two Networks

In this section, we first study important features in SDN and CN that can be used in our hybrid scenario. A new routing protocol is then proposed for H-SDN, integrating advantages from both networks.

A. SDN: Fine-Grained Flow Control and Source Virtualization

SDN is a programmable networking design offering global view of the network traffic and centralized flow management. The SDN controller is able to periodically gather traffic information from the control plane and accordingly distributes traffic demands so as to shape the traffic and optimize the network performance. The routing decisions are then disseminated as "policies" to SDN forwarding elements, which execute the decisions as "actions". SDN enables fine-grained flow control, where routing tables are also programmable to operators. Fig. 2 shows an example of the routing table. Extra options, such as "Rule", "Priority" and "Source", can be programmed as "tags" to identified flows for respective actions that are also re-programmable to the controller.

Another important feature that can be enabled by SDN is the source virtualization. Without a global view, traditional networks separate sourcing selection from other networking functionalities, where content providers are in charge of making server-redirection decisions for remote requests while networks only deal with the so-decided peer-to-peer traffic matrix. This leads to a series of suboptimal routing in CN [20]. On the contrary, in SDN, source servers, either caches, CDNs or clouds, are virtualized as source points with detailed insights provided to the SDN controller [21]. Apparently, extra TE gain can be obtained by strategically distributing traffic demands among sources. Later in this paper, we will exploit such a feature to further optimize the TE goal.

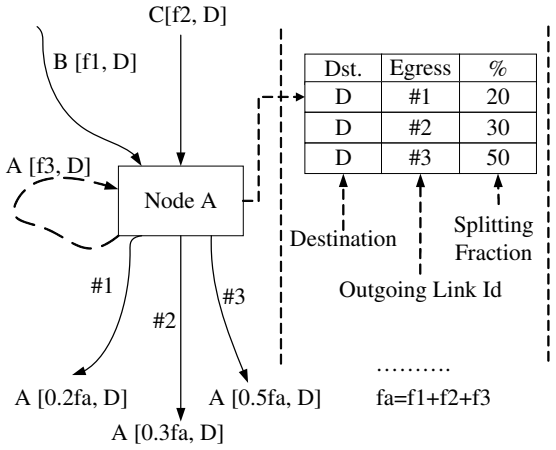


Figure 3. A routing example based on flow-splitting table.

B. CN: Destination-Based Routing and Traffic Aggregation

CN routers identify flows (packets) by their destinations and forward them to corresponding outgoing links along the shortest paths that are distributively calculated via OSPF or IS-IS. Traditionally, flows are equally split and forwarded among several shortest paths if they exist [13]. Moreover, the flow splitting fractions can also be online configurable via gently modified routing protocols [22]. In this paper, we assume that the CN routers are configured to support the adaptive flow-splitting commands.

The concept of destination-based traffic aggregation is proposed in [23] for OSPF networks and improved by [19] to aggregate user datagram protocol (UDP) based mobile packets in mobile core networks. At a generic CN router, incoming flows towards a common egress point can be aggregated, and then split and forwarded like one flow (regardless of sources). The rationale behind this is that all chosen routes are shortest paths and shortest paths have the property that any segments of shortest paths are also shortest paths.

We note that such destination-based routing and traffic aggregation can also be easily programmed at SDN routers.

C. A Transitional Routing Protocol for H-SDN

To provide a foundation for later discussion, in this section, we propose a simple routing protocol and assume that all nodes have supported it so that all traffic in the SDN domain can be strategically routed.

In the protocol, we assume that the information of link status can be periodically gathered by the SDN controller, through the control plane from SDN elements and periodical reports from CN nodes [24]. We also assume that flows are distinguished only by corresponding destination identifiers and flows towards common destinations are aggregated over routers.

We then slightly modify the routing table to support programmable flow splitting. We add a column of flow-splitting ratios to each routing table, representing how the incoming flows to be split and forwarded to outgoing links. Fig. 3 is an example showing how the protocol works. Router A has two incoming flows $f1$ from B and $f2$ from C to egress router

D. A is one of the source nodes for some objects requested by node D, thus it generates a local-loop flow $f3$ towards D, which is also treated as an incoming flow for A. For routing purposes, router A reads the destination of each incoming flow, and groups flows with respect to distinct destinations. Then for each destination, A combines flows in the group, looks up the routing table for the splitting ratios and accordingly splits the aggregate flow and forwards the split flows to respective outgoing links, *i.e.,* $\#1$, $\#2$, and $\#3$.

Here we note that it is a slight change to existing routing table, which can be easily implemented in existing routers [19], [22]. In our scenario, the controller computes these routing decisions and disseminates them in the control plane to SDN nodes and CN nodes. Accordingly, the traffic control policies at SDN nodes and routing configurations at CN nodes are updated.

IV. Traffic Engineering in Hybrid SDNs

In this section, we first formulate the system model in the paper. With the proposed routing protocol, we argue that fine-grained traffic control can be supported by both SDN routers and non-SDN routers. Afterwards, we propose the optimization frameworks for TE in H-SDN.

A. System Model and Notations

We target a hybrid software defined network depicted in Fig. 1, where SDN nodes and virtualized sourcing nodes co-exist with traditional routers. SDN nodes are incrementally deployed SDN-compliant routers, and a virtualized sourcing node can be treated as a special SDN node integrated with a sourcing server. A traditional router refers to the widely deployed CN node [8], whose behaviors can only be tweaked by network configurations, *e.g.,* routing tables. In-system links are shared by hybrid traffic of CN flows and SDN requests. CN traffic is assumed to be optimized via standard link-state routing protocols such as Open Shortest Path First Traffic Engineering (OSPF-TE), and is termed as *uncontrollable* background traffic in our scenario. As a transition system from CN to SDN, two basic link-sharing modes are considered in this paper: (1) the *barrier mode*, in which a certain portion of capacity is reserved on each link for SDN requests; and (2) the *hybrid mode*, in which either form of traffic can use up to the full capacity of each link.

SDN requests are initiated by destination (SDN) nodes requesting for respective network objects. The information of SDN requests, *e.g.,* requested objects, flow demands, is accessible to the SDN controller through the control plane. To optimize the overall traffic engineering, the SDN controller is then responsive to periodically schedule SDN requests according to the real-time network traffic gathered. Specifically, the controller first needs to strategically redirect requests to corresponding sourcing servers (source redirection). Then it needs to optimize flow routing for so-generated peer-to-peer traffic over multiple paths (flow routing). The scheduling optimization is assumed to be repeated from cycle to cycle at the same time scale as the dynamics of the SDN requests, *e.g.,* a few seconds, while during the period, network topology, background traffic and SDN requests are assumed to be fixed.

Table I
Basic notations.

Notation	Definition
$b(e)$	The uncontrollable traffic on edge e
$c(e)$	Capacity of edge e
d_i^k	Traffic demand of request (i,k)
E	Set of in-system links
O	Set of network objects available to SDN requests
$\mathcal{P}_i^k$	Set of admissible paths to request (i,k)
$\mathcal{P}$	Set of all involved paths for the current period
R	Set of SDN requests
R_i	Set of requests at node i
S_k	Set of nodes that store object k
V	Set of serving nodes
$x(P)$	Amount of flow routed on $P \in \mathcal{P}_i^k$ w.r.t. (i,k)

We formulate the hybrid network as a directed graph $G(V,E)$, where V is the set of nodes, including SDN nodes, sourcing nodes and CN nodes with $n = |V|$, E is the set of shared links with $m = |E|$. Network objects are distributed among sourcing nodes S. Let O represent the set of objects requested at the time period considered. Each requested object $k \in O$ corresponds to a subset of sourcing nodes $S_k \subset S$, which can serve the request. Let R denote the set of SDN requests initiated at the beginning of the period considered with $r = |R|$. An SDN request is defined by a tuple $(i,k), i \in V, k \in O$, representing node i requesting object k, and is further associated with a traffic demand d_i^k to be fulfilled.

Link $e \in E$ has a capacity $c(e)$, accommodating both SDN traffic and background traffic. The background traffic $b(e)$ on each link e is readily retrieved or estimated by the controller through the dynamic information disseminated by OSPF-TE [15]. A path P_{ji} is called *admissible* to request (i,k) if it is a simple path with $j \in S_k$, which can potentially serve the request. In the multiple-path network, we further use $\mathcal{P}_{ji}$ to denote all the simple paths from j to i, $\mathcal{P}_i^k$ to denote all admissible paths for request (i,k) and $\mathcal{P}$ to represent all involved paths in the current period. Table I summarizes the important notations used in this paper.

The request scheduling problem is now to fulfill all SDN requests by determining the amount of flow to be loaded among respective admissible path sets, termed as $x(P)$, where P is an admissible path with respect to corresponding request. We note that the path decisions of $\{x(P)\}$ here also imply the source redirection, since the starting node of each path P refers to a sourcing node. Furthermore, by enumerating path-flow values of $\{x(P)\}$, it is ready to compute the hop-by-hop flow-splitting fractions for the modified routing protocol described in Section III-C.

B. Traffic Engineering in Barrier Mode

The barrier mode intends to isolate SDN traffic from traditional CN traffic by setting provisioning barriers in the capacity space of shared links, where SDN traffic and CN traffic are actually routed in distinct overlay networks. Each form of traffic is restricted in its own capacity space, *e.g.,* 40 percent of link utilization is reserved for SDN flows while CN flows can use

up to the rest 60 percent. This is an instant solution for "hybrid" traffic and is readily to be implemented since routers are able to distinguish the two forms of traffic. Furthermore, due to the isolation it guarantees, new techniques can be introduced and tested in the hybrid network without influencing the stability of CN traffic. Therefore, it is a practical and conservative mode in the transition period. However, these barriers might lead to over-provisioning and potential underutilization.

Suppose that we have reserved a capacity space of η, $0 < \eta < 1$, for SDN traffic in the network[1]. The traffic-engineering problem is now to minimize the maximum link utilization in the provisioning capacity space, which is formulated as a linear programming (LP) problem:

$$\begin{aligned} \min_{x(P)} \ & \lambda \\ \text{s. t.} \ & \sum_{P:e\in P} x(P) \le \lambda\eta c(e), \ \forall e \in E \\ & \sum_{P\in\mathcal{P}_i^k} x(P) \ge d_i^k, \ \forall (i,k) \in R \\ & x(P) \ge 0, \ \forall P \in \mathcal{P} \end{aligned} \tag{1}$$

where the first constraint represents the capacity limitation in the link space, scaled by the factor λ, and the second constraint requires that all the demands should be fulfilled.

We note that this formulation potentially involves an exponential number of variables $x(P)$ due to the combinatorial explosion in path enumeration. It can be converted to an equivalent formulation with polynomial-size edge-flow variables with the method in [19]. However, as estimated therein, the size of the equivalent problem remains far beyond the LP-solving range of any modern LP solver, *e.g.,* CPLEX [25]. Instead, we attempt to develop a fast algorithm for problem (1) with the approximate guarantees.

C. Traffic Engineering in Hybrid Mode

The hybrid mode goes beyond provisioning barriers, and allows a fine-grained and cognitive way of traffic scheduling to the SDN controller.

At the beginning of each cycle, the controller schedules SDN requests with awareness of uncontrollable traffic on each link, so as to optimize the overall TE. The problem is formulated as a similar LP problem as follows:

$$\begin{aligned} \min_{x(P)} \ & \lambda \\ \text{s. t.} \ & \sum_{P:e\in P} x(P) + b(e) \le \lambda c(e), \ \forall e \in E \\ & \sum_{P\in\mathcal{P}_i^k} x(P) \ge d_i^k, \ \forall (i,k) \in R \\ & x(P) \ge 0, \ \forall P \in \mathcal{P}. \end{aligned} \tag{2}$$

The only difference here lies in the first constraint, which considers the background traffic on each link $e \in E$. Likewise, this LP problem has a similar size as problem (1). Therefore, a fast approximate algorithm is more desirable.

[1]Here, we only formulate the unified scenario, where all links reserve η portion of capacity for SDN traffic. However, our formulation is readily extended to scenarios with various reservation portion for fine-grained configurations.

V. Algorithm for Barrier Traffic Engineering

For ease of presentation, we define $c'(e)$ to be the capacity of link e in the overlay network, *i.e.,* $c'(e) \triangleq \eta c(e), \forall e \in E$, where η is the capacity provisioning for SDN requests. To solve problem (1), we first study the problem of maximizing the concurrent flow, which is formulated as:

$$\begin{aligned} \max_{y(P)} \ & \pi \\ \text{s. t.} \ & \sum_{P:e\in P} y(P) \le c'(e), \quad \forall e \in E \\ & \sum_{P\in\mathcal{P}_i^k} y(P) \ge \pi d_i^k, \quad \forall (i,k) \in R \\ & y(P) \ge 0, \quad \forall P \in \mathcal{P}. \end{aligned} \tag{3}$$

By letting $x(P) = y(P)/\pi, \lambda = 1/\pi$, we note that problem (3) is equivalent to problem (1).

Problem (3) is in a similar form as the maximum concurrent multi-commodity flow (MCMF) problem, which is extensively investigated in [26]–[28]. The key difference lies in the fact that flows in MCMF are identified by fixed source-destination pairs, while in our problem, flow demands are destination-oriented and the sourcing nodes remain to be determined. Although problem (3) can be viewed as a variant of the MCMF problem with multiple sources for each requested flow, the algorithms developed in the literature depend heavily on the source-destination pairs, thus cannot be applied to our scenario.

In the following, we leverage the results from [28] while modifying the kernel scheme of shortest-path calculation so as to solve the flow routing problem (3) in a scenario with multiple sources as well as multiple paths.

A. Proposed Approximation Algorithm

First, we look at the dual formulation of problem (3):

$$\begin{aligned} \min_{l(e)} \ & D(l) \triangleq \sum_{e\in E} c'(e) l(e) \\ \text{s.t.} \ & \sum_{e\in P} l(e) \ge z_i^k, \quad \forall (i,k) \in R, \forall P \in \mathcal{P}_i^k \\ & \sum_{(i,k)\in R} z_i^k d_i^k \ge 1 \\ & l(e), z_i^k \ge 0, \quad \forall (i,k) \in R, \forall e \in E \end{aligned} \tag{4}$$

where $l(e)$ is the length function assigned to each edge e and z_i^k is a dual variable w.r.t each request $(i,k) \in R$. For a given length function l, we can eliminate the first constraint in problem (4) by rewriting z_i^k as a function of l, *i.e.,* $\psi_i^k(l)$, representing the length of the shortest path from the nearest admissible node in S_k to node i, mathematically defined as:

$$\psi_i^k(l) \triangleq \min_{j\in S_k} \psi_{ji}(l), \quad \forall (i,k) \in R \tag{5}$$

where $\psi_{ji}(l)$ presents the minimum distance from node j to node i using length function l.

If we define $\alpha(l)$ as the aggregate shipping cost for all requests via the "shortest" paths, *i.e.,*

$$\alpha(l) \triangleq \sum_{(i,k)\in R} d_i^k \psi_i^k(l) \tag{6}$$

Algorithm 1: Algorithm for problem (3).

Input: Network graph $G=(V,E)$, network object set O, overlay link capacities $c'(e)$, source set $\{S_k\}, k\in O$, SDN requests set R, request demands $\{d_j^k\}$, accuracy ϵ

Output: Primal solution $\mathbf{y}$ and π

Initialize $l(e)\leftarrow \delta/c'(e), \forall e, y(P)\leftarrow 0, \forall P$
while $D(l)<1$ **do**
  for $j=1$ to n **do**
    For all requests $(j,k)\in R_j$, initialize $\tilde{d}_j^k = d_j^k$.
    while $D(l)<1$ and $\tilde{d}_j^k>0$ for some k **do**
      $K\leftarrow \{k|(j,k)\in R_j, \tilde{d}_j^k>0\}$
      $P_j^k \leftarrow$ shortest path in $\mathcal{P}_j^k$ using length l
      $\rho \leftarrow \max\left\{1, \max\limits_{e\in\cup_{k\in K}P_j^k} \dfrac{\sum_{k:e\in P_j^k}\tilde{d}_j^k}{c'(e)}\right\}$
      for $k\in K$ **do**
        $f_j^k \leftarrow \tilde{d}_j^k/\rho$
        $\tilde{d}_j^k \leftarrow \tilde{d}_j^k - f_j^k$
        $y(P_j^k)\leftarrow y(P_j^k)+f_j^k$
      end for
      $l(e)\leftarrow l(e)\left(1+\epsilon\frac{\sum_{k:e\in P_j^k} f_j^k}{c'(e)}\right), \forall e\in\cup_{k\in K}P_j^k$
    end while
  end for
end while
$y(P)\leftarrow y(P)/\log_{1+\epsilon}\frac{1+\epsilon}{\delta}, \forall P$
$\pi \leftarrow \min\limits_{(j,k)\in R} \dfrac{\sum_{P\in\mathcal{P}_j^k} y(P)}{d_j^k}$

then solving problem (4) is equivalent to finding a length function l, such that $D(l)/\alpha(l)$ is minimized [26]. We further denote the optimal value of problem (4) by β

$$\beta \triangleq \min_{l} D(l)/\alpha(l) \tag{7}$$

which is equal to the optimal objective value of the primal problem (3) according to the primal-dual theory.

Alg. 1 summarizes the procedure of our algorithm for problem (4). The algorithm proceeds in a similar phase-step-iteration manner as the scheme in [28]. Each phase consists of n iterations. Each iteration j reflects to a node j, in which all the traffic demands requested by the node are fulfilled through a series of steps. Inside each step, only uncompleted requests are considered and partially fulfilled by the nearest source through the shortest path under the current length function l, *i.e.,* the shortest admissible path. The link function is then updated at the end of each step. The shortest admissible paths are computed by constructing a shortest path tree at each iteration. For instance, in iteration j, we consider a reversed graph G' of G, where the directions of all edges E are reversed. Using Dijkstra's algorithm, we obtain the shortest-path tree T_j rooted at node j in $O(n\log n+m)$, which contains all the possible admissible paths to be used in the series of steps inside.

Different from [12], where they investigated almost the same problem as the MCMF problem and slightly modified the algorithm in [28] for their purposes, our algorithm only leverages the framework of the one proposed in [28], while most of the details inside are re-factored to jointly consider the source-redirection problem.

B. Correctness and Approximation

The correctness of Alg. 1 can be revealed by comparing it to the naive approach of adding a super source node for each object and directly applying the solutions to the maximum concurrent multi-commodity flow problem (*e.g.,* in [28]). Super source nodes are connected to responding source nodes via infinite-capacity links. Problem (3) then reduces to the maximum concurrent flow problem [26]. For each request (i,k), the centralized controller needs to compute the shortest path from the super source node of commodity k to node i under the length function l. Since the outgoing links of any super source have an infinite capacity, the length of these links stays 0, according to the length definition in Alg. 1. Correspondingly, finding the shortest path from the super source for k to node i is equivalent to computing the shortest admissible path among all potential sourcing nodes. Therefore, Alg. 1 finds the same solution as the naive scheme, which validates its correctness.

We next analyze the approximation ratio of Alg. 1. For ease of presentation, we add subscripts to indicate phase i, iteration j and step s. Let $\tilde{d}_{i,j,s}^k$ be the demand for object k that has not been fulfilled at step s and it is set to d_j^k at step 0. In Alg. 1, the length function is updated at the end of each step by:

$$\begin{aligned} l_{i,j,s}(e) &= l_{i,j,s-1}(e)(1+\epsilon\cdot\frac{\text{sum of new flows on } e}{c(e)}) \\ &= l_{i,j,s-1}(e)(1+\epsilon\cdot\frac{\sum_{k:e\in P_j^k} f_{i,j,s}^k}{c(e)}) \end{aligned} \tag{8}$$

for step (i,j,s), where $0<\epsilon<1$ is to be defined later.

At step s of iteration j of the ith phase, the dual objective value $D(l)$ is increased by

$$\begin{aligned} \triangle D(l_{i,j,s}) &\overset{(8)}{=} \epsilon \sum_{e\in\cup_{k\in K}P_j^k} l_{i,j,s-1}(e) \sum_{k:e\in P_j^k} f_{i,j,s}^k \\ &= \epsilon\sum_{k\in K} f_{i,j,s}^k \sum_{e\in P_j^k} l_{i,j,s-1}(e) \\ &\overset{(5)}{=} \epsilon\sum_{k\in K} f_{i,j,s}^k \psi_j^k(l_{i,j,s-1}) \end{aligned} \tag{9}$$

where P_j^k is the shortest admissible path w.r.t. request (j,k) and $\psi_j^k(l_{i,j,s-1})$ is the corresponding shortest distance, which is updated using the length function in the previous step. Since $\psi_j^k(l)$ is monotonically increasing through the steps, if κ_j denotes the number of steps in iteration j of the ith phase, we have $\psi_j^k(l_{i,j,s})\le\psi_j^k(l_{i,j,\kappa_j}), \forall s\le\kappa_j$. Hence, for the entire iteration, the dual objective increase is bounded by

$$\begin{aligned} \sum_{s=1}^{\kappa_j}\triangle D(l_{i,j,s}) &\overset{(9)}{=} \epsilon\sum_{s=1}^{\kappa_j}\sum_{k\in K} f_{i,j,s}^k\psi_j^k(l_{i,j,s-1}) \\ &\le \epsilon\sum_{s=1}^{\kappa_j}\sum_{k\in K} f_{i,j,s}^k\psi_j^k(l_{i,j,\kappa_j}) \\ &= \epsilon\sum_{k\in K} d_j^k\psi_j^k(l_{i,j,\kappa_j}). \end{aligned} \tag{10}$$

Let $l_{i,1,0}$ denote the length function after the last step of the last iteration in phase $i-1$, $D(i)$ denote the corresponding dual objective value $D(l_{i,1,0})$, and $\alpha(i)$ denote $\alpha(l_{i,1,0})$. We further obtain the increase bound of the dual objective during the ith phase as

$$\begin{aligned}\sum_{j,s} \triangle D(l_{i,j,s}) &\leq \epsilon \sum_{j} \sum_{k\in K} d_j^k \psi_j^k(l_{i,j,\kappa_j}) \\ &\leq \epsilon \sum_{j} \sum_{k\in K} d_j^k \psi_j^k(l_{i+1,1,0}) \\ &= \epsilon\alpha(l_{i+1,1,0}). \end{aligned} \tag{11}$$

Therefore, we have the following recursive relationship:

$$\begin{aligned} D(l_{i+1,1,0}) &\leq D(l_{i,1,0}) + \epsilon\alpha(l_{i+1,1,0}) \\ \Longleftrightarrow D(i+1) &\leq D(i) + \epsilon\alpha(i+1). \end{aligned} \tag{12}$$

According to (7), the fact that $\beta \leq D(i)/\alpha(i)$ implies that

$$\frac{D(i+1)}{D(i)} \leq \frac{1}{1-\epsilon/\beta}, \quad \forall i \geq 0, \epsilon < 1. \tag{13}$$

Together with the initial setting $D(0) = m\delta$ (see line 1 of Alg. 1) and the assumption $\beta \geq 1$, we finally have the non-recursive inequality for $D(i)$, given by

$$\begin{aligned} D(i) &\leq \frac{m\delta}{(1-\epsilon/\beta)^i} \\ &= \frac{m\delta\beta}{\beta-\epsilon}(1+\frac{\epsilon}{\beta-\epsilon})^{(i-1)} \\ &\leq \frac{m\delta\beta}{\beta-\epsilon} e^{\epsilon(i-1)/(\beta-\epsilon)} \\ &\leq \frac{m\delta}{1-\epsilon} e^{\epsilon(i-1)/(\beta-\epsilon)}. \end{aligned} \tag{14}$$

If the algorithm terminates after q phases, which implies that $D(q) \geq 1$, we have

$$\beta \leq \frac{\epsilon(q-1)}{(1-\epsilon)\ln\frac{1-\epsilon}{m\delta}}. \tag{15}$$

Therefore, similar to [26], [28], we can claim the following conclusions. We omit the proofs here because all the remaining details are the same as those in [26], [28].

Theorem 1. *If $\beta \geq 1$, the number of phases in Alg. 1 is bounded by $\lceil \beta \log_{1+\epsilon} \frac{1+\epsilon}{\delta} \rceil$.*

Theorem 2. *Provided $\beta \geq 1$, if π denotes the objective value of problem (3) found by Alg. 1, we have $\pi > \frac{q-1}{\log_{1+\epsilon}\frac{1+\epsilon}{\delta}}$, where*

$$\delta = \frac{1}{(1+\epsilon)^{\frac{1-\epsilon}{\epsilon}}} \cdot \left(\frac{1-\epsilon}{m}\right)^{1/\epsilon}.$$

The obtained objective value is related to the optimal value β according to the inequality: $\frac{\beta}{\pi} < (1-\epsilon)^{-3}$.

Therefore, if we choose ϵ such that

$$(1-\epsilon)^{-3} = 1+\omega \tag{16}$$

the approximation ratio $\frac{\beta}{\pi}$ is less than $(1+\omega)$ for any $\omega > 0$.

C. $\beta < 1$ and Running Time

In any case with $\beta < 1$, we show that the demands of requests can always be scaled down to produce a corresponding instance with $\beta \geq 1$. To see this, we look at the dual problem (4) and its optimal objective in Eq. (7). An instant observation suggests that, in an instance of problem (4), if all demands are scaled down by a factor γ, its optimal value β will increase by a factor $1/\gamma$. Then, the remaining procedure is to derive a lower bound and an upper bound of β for a given instance.

We first consider a special case where we only route the request for object k at node i, and denote the optimal fraction of demands (*i.e.,* maximum flow divided by d_i^k) by γ_i^{k*}. Using a modified version of Dijkstra's algorithm introduced in [27], we can obtain an m-approximation γ_i^k of γ_i^{k*}. In other words, $\gamma_i^k \geq \frac{1}{m}\gamma_i^{k*}$. Let $\bar{\gamma} = \min_{i\in V, k\in R_i} \gamma_i^k$. It is not hard to see that the optimal objective $\beta \leq \min_{i\in V, k\in R_i} \gamma_i^{k*} \leq m\bar{\gamma}$. On the other hand, it is feasible to route γ_i^k/r fraction of demand simultaneously for every object k at every node i. Hence, $\bar{\gamma}/r \leq \beta \leq m\bar{\gamma}$. Therefore, we can develop an instance that ensures $\beta \geq 1$ by multiplying the initial demands by $\bar{\gamma}/r$.

However, the procedure above makes β as large as mr. Recalling Theorem 1, we observe that a large β results in more phases. We apply Garg's techniques in [26] and define $Q \triangleq 2\lceil \frac{1}{\epsilon}\log_{1+\epsilon}\frac{m}{1-\epsilon}\rceil$. If the procedure does not stop within Q phases, we know that $\beta \geq 2$. Thus, we proceed to a reduction scheme by doubling the demands of all commodities so that β is halved. After at most $\log(mr)$ times of reduction, we will reach $1 \leq \beta \leq 2$. Accordingly, the total number of phases required for the reduction is at most $\log(mr)\cdot Q$.

To further reduce the possible phases, we adopt the technique in [29], and first compute a 2-approximation to β which requires at most $O(\log(mr)\log m)$ phases[2] and returns $\bar{\beta}$, $\beta \leq \bar{\beta} \leq 2\beta$. Now we create a new instance by multiplying all the demands by $\bar{\beta}/2$. As a result, the new instance has $1 \leq \beta \leq 2$, and our procedure terminates within Q phases.

For a given small value $\epsilon > 0$, we can further rewrite $Q = O(\epsilon^{-2}\log m)$. Thus, the total number of phases for the original problem is $O((\epsilon^{-2}+\log(rm))\log m)$.

The number of iterations in each phase is bounded by the number of serving nodes (n). In an iteration, at every step except the last one, the length of at least one edge is increased by a fraction of at least $(1+\epsilon)$. The length of each edge $e \in E$ is initialized to be $\delta/c(e)$, and it grows up to $(1+\epsilon)/c(e)$ when the procedure terminates. Thus, the number of steps for the $(1+\epsilon)$-length increase is at most $m\log_{1+\epsilon}\frac{1+\epsilon}{\delta} = O(\epsilon^{-2}m\log m)$. Together with the total number of iterations (incurred by the last step of each iteration), the total number of required steps is at most $O(n(\epsilon^{-2}+\log(rm))\log m) + O(\epsilon^{-2}m\log m) = O(\epsilon^{-2}m\log m + n\log m\log r)$, where ω and ϵ are chosen as in (16). Therefore, we arrive at the following conclusion:

Theorem 3. *Let $r_{\max}$ be the maximum number of SDN requests initiated by one node, i.e., $r_{\max} \triangleq \max_{i\in V}|R_i|$. Define $T_s \triangleq O(n\log n + m + r_{max})$. Alg. 1 computes a $(1+\omega)$-approximation solution to problem (1) in $\tilde{O}((\omega^{-2}+\log r)m\cdot T_s)$*

[2]Here, we run Alg. 1 together with the above scaling procedure for the approximation ratio of $\omega = 1$. Thus, $Q = O(\log m)$.

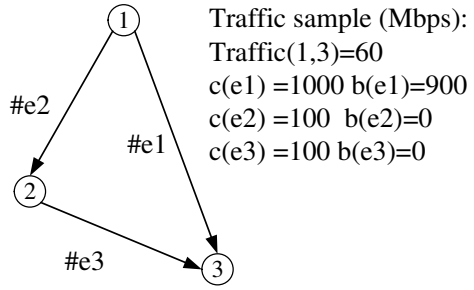


Figure 4. An example of multipath flow scheduling with background traffic.

time ($\tilde{O}(f) = O(f \cdot \log^{O(1)} m)$), for any $\omega > 0$, where ω and ϵ are in the same order.

VI. Algorithm for Hybrid Traffic Engineering

If we ignore the source-redirection subproblem implied, the formulation of problem (2) is very similar to the TE problem formulated in [12]. However, from the route adopted therein, problem (2) is converted to an "equivalent" form:

$$\begin{aligned} \max_{y(P)} \quad & \theta \\ \text{s. t.} \quad & \sum_{P:e\in P} y(P) \le c(e) - b(e), \quad \forall e \in E \\ & \sum_{P\in\mathcal{P}_i^k} y(P) \ge \theta d_i^k, \quad \forall (i,k) \in R \\ & y(P) \ge 0, \quad \forall P \in \mathcal{P} \end{aligned} \tag{17}$$

After the conversion, they solved the "equivalent" problem via a slightly modified algorithm from [28]. If we can follow the route, then problem (2) is readily solved using the algorithm developed in Section V. However, we argue that these two problems are actually inequivalent. For instance, we consider the flow scheduling example depicted in Fig. 4. A traffic demand of 60Mbps from node 1 to node 3 is to be scheduled by the network. It can be fulfilled through path $(e1)$ or $(e2, e3)$, where link $e1$ is a backbone link of capacity $1,000$Mbps with background traffic of 900Mbps, $e2$ and $e3$ are local links of 100Mbps capacity each with no background traffic. Using the route from [12], the optimal routing is to split the traffic and to load each admissible path with an amount of 30Mbps, resulting in a global maximum link utilization of $\frac{900+30}{1000} = 0.93$. Obviously, if we load all the demands to path $(e2, e3)$, the resulted TE factor remains $\frac{900}{1000} = 0.90$. Therefore, this counterexample reveals the inequivalence between a hybrid TE problem and a barrier TE problem.

We also note that Alg. 1 relies heavily on the scalability of variables, which does not exist in problem (2) with constant values of background traffic. This means that Alg. 1 cannot be modified in any way to fit the hybrid TE scenario. Instead, we need to develop a new approach.

We denote the maximum link utilization before scheduling SDN requests by f_{te}, *i.e.,* $f_{te} \triangleq \max_{e\in E} b(e)/c(e)$. Apparently, f_{te} is a lower bound of the optimal TE value of problem (2). Suppose that up to λ_0 fraction of link capacity can be used for data traffic[3]. Thus we also obtain an upper bound for the optimal TE. Therefore, we can strategically find the optimal solution to problem (2) via a binary search. The detailed procedure is summarized in Alg. 2.

Algorithm 2: Binary-Search Algorithm for problem (2)

Input: Network Graph $G = (V, E)$, capacities $\{c(e)\}$, network object set O, background traffic $\{b(e)\}$, source set $\{S_k\}$, request set R, link utility upper bound λ_0 and accuracy δ, appr. ratio ω

Output: Hybrid TE value λ, link flow assignment $\mathbf{x}$

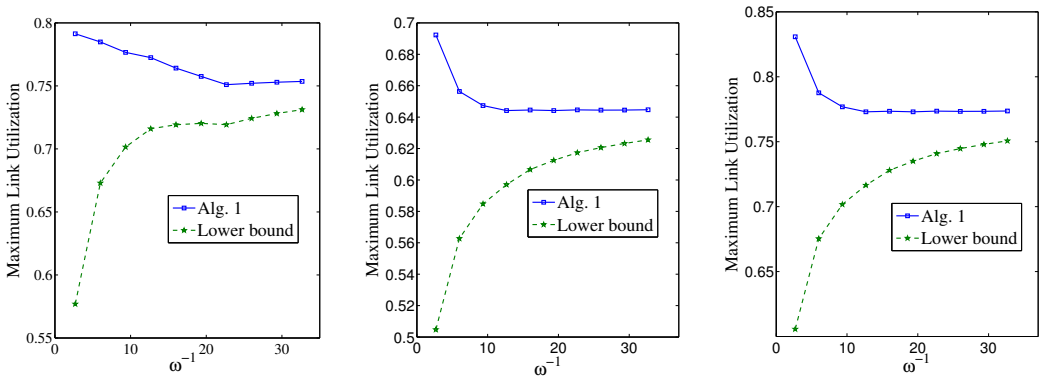
$f_{te} \leftarrow \max_{e\in E} b(e)/c(e)$
$\lambda_+ \leftarrow \lambda_0,\ \lambda_- \leftarrow f_{te}$
while $\lambda_+ - \lambda_- > \delta$ **do**
  $\lambda \leftarrow (\lambda_+ + \lambda_-)/2$
  For all $e \in E$, compute provisioning link capacity $c'(e) \leftarrow \lambda c(e) - b(e)$.
  Use Alg. 1 to compute a $(1+\omega)$-approximate result π^* for problem (3) with overlay link capacities of $\{c'(e)\}$.
  $\lambda^* \leftarrow 1/\pi^*$
  if $\lambda^* > 1$ **then**
    Print "infeasible"
    $\lambda_+ \leftarrow \lambda$
  else
    Print "feasible"
    $\lambda_- \leftarrow \lambda$
    $\mathbf{z} \leftarrow \mathbf{y}$, $\pi^{fes} \leftarrow \pi^*$ // Record feasible solution
  end if
end while
if No feasible solution is found **then**
  $\mathbf{z} \leftarrow \mathbf{y}$, $\pi^{fes} \leftarrow \pi^*$ // Record the last solution
end if
$\mathbf{x} \leftarrow \mathbf{z}/\pi^{fes}$
Output λ_- and $\mathbf{x}$

In each search step of the algorithm, we first compute an overlay network. Capacity provisionings vary among links. For link $e \in E$, the provisioning ratio $\eta(e) = \lambda - b(e)/c(e)$. Then we use Alg. 1 to check if the requests are routable in the overlay network with specific capacity provisioning. By selecting appropriate ω in the same order as δ, Alg. 1 obtains a δ-suboptimal solution to problem (2) for any $\delta > 0$.

Let B be the largest number used to specify the runs of binary search with respect to λ_0, f_{te} and the accuracy threshold δ. Hence, Alg. 2 runs $\log B$ times of Alg. 1. According to Theorem 3, Alg. 2 consumes a computation time of $\tilde{O}((\omega^{-2} + \log r)m \cdot T_s \cdot \log B)$.

We also note that in some cases with heavy loads, Alg. 1 may not be able to produce a feasible solution even if λ_0 is selected for provisioning computation. In these scenarios, Alg. 2 returns a solution in which all requested demands are proportionally scaled down. We comment that this is still a reasonable traffic assignment for the hybrid network.

[3]Practical networking systems usually control an upper bound for the maximum link utilization so as to avoid over-congestions which greatly degrade the stability of the networks. For instance, $\lambda_0 = 0.95$.



(a) $n = 30$. (b) $n = 50$. (c) $n = 70$.

Figure 5. Maximum link utilization with different approximation ratios ω.

VII. Performance Evaluation

A. Setup

We developed a simulator to implement our algorithms with around 1500 lines of C++ code. As shown in Section V and Section VI, the efficacy of the proposed algorithms is independent of specific network factors, such as system topologies, source placement and request patterns. In this section, we simulate the routing algorithms in hybrid SDN networks with randomized configurations.

We use Inet-3.0 [30] to generate network topologies with a mean degree of 4 by default. All links are considered to be bi-directional with the capacity of 1Gbps for each direction. SDN nodes and sourcing points are randomly selected from the generated networking nodes. For SDN requests, $20,000$ network objects are randomly distributed among the sourcing nodes such that each object has a mean number of 5 replicas. SDN nodes randomly initiate data requests with traffic demands uniformly selected from 512kbps to 3Mbps discretely with a step size of 128kbps. SDN traffic load is then controlled by the factor *traffic density*, which denotes the average number of requests collected from SDN nodes in each cycle. The default accuracies for ϵ, ω, δ are 0.10, 0.05, 0.10, respectively.

B. Barrier Mode: The Overlay Network

Similar to the TE solution in [12], SDN flows are scheduled in a separate overlay network without exploiting the information of CN traffic. In this section, we show the numerical results of Alg. 1 developed in overlay networks, where the maximum link utilization is also calculated over corresponding overlay links. Here we assume that SDN traffic can use up to 40 percent of capacity on each link.

1) The Approximation Gap: In Section V, we have shown that Alg. 1 achieves a $(1 + \omega)$-approximation solution to each case of TE problem. Therefore, for each maximum link utilization value λ produced by Alg. 1, a lower bound of the optimal TE can be obtained by $\frac{\lambda}{1+\omega}$. Together with the fact that for the optimization problem (1), any calculated result is an upper bound of the optimum, the optimal value then lies within $\frac{\lambda}{1+\omega}$ and λ for any selected ω.

We simulate this conclusion in three H-SDN systems with the number of nodes $30, 50, 70$ and the number of SDN nodes $10, 20, 30$, respectively. The traffic density is 1500 in all cases. We then run Alg. 1 in so-generated network instances with different approximation ratios ω. Fig. 5 shows the relation between the approximation gap and ω. We conclude from the figure that with proper selection of ω, the approximation gap of Alg. 1 can be arbitrarily small.

2) Running Time: In each instance of problem (2), the running time of Alg. 1 is mainly determined by the number of phases executed before it terminates (see Section V-C). In the simulations above, we also sum up the total number of phases in each running cases and show the results in Fig. 6. The three quadratic curves in it confirm the conclusion that the total number of phases in Alg. 1 is proportional to ω^{-2}.

3) TE Performance and Multi-Source Gain: For comparison, we align our algorithms with the OSPF-based TE routing scheme, in which traffic demands are assumed to be fulfilled by the "nearest" source via shortest paths. We then show the gain of source redirection enabled in this paper for TE problem by comparing Alg. 1 to a slightly modified version, in which each available sourcing set S_k reduces to a single sourcing node with a minimum distance to corresponding SDN node.

The simulated network has 50 nodes, 20 of which are SDN nodes. Traffic density is gradually increased from 100 to 2000. The numerical results are shown in Fig. 7. From the figure, we conclude that having SDN traffic distributed among sources can significantly improve the TE performance without introducing much overhead.

C. Hybrid Mode: Fully Shared Links

Different from the overlay networking mode, the hybrid mode has the capacities of network links fully shared by CN traffic and SDN traffic. This makes it more difficult to show the efficacy of our algorithms, since the output relies heavily on the uncontrolled CN traffic. We are not going to align our algorithms with the scheme developed in [12]. As shown in Section VI, the performance of their method could be arbitrarily low with unpredictable, time-varying CN traffic.

To provide a fundamental understanding, we compare the performance of Alg. 2 to the barrier algorithm, *i.e.,* Alg. 1, where 40 percent of capacity is reserved for SDN traffic on each link. We use the hybrid network with 50 nodes and 20 SDN nodes. CN traffic are randomly generated with maximum

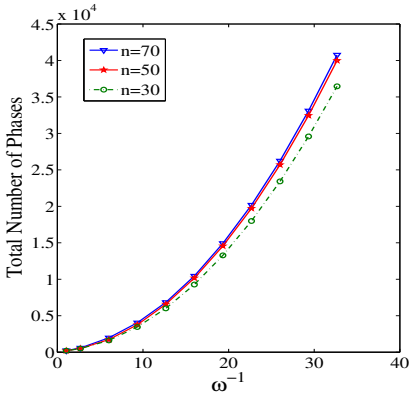


Figure 6. Number of phases in Alg. 1.

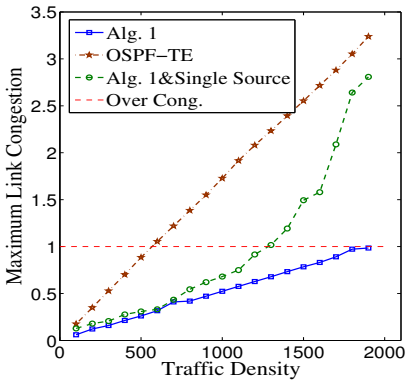


Figure 7. TE performance and multi-source gain.

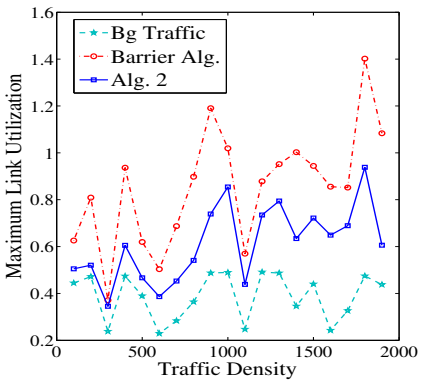


Figure 8. TE performance in hybrid mode.

link utilization between 0.2 and 0.5. The same set of SDN requests are then scheduled by Alg. 1 and Alg. 2, respectively. We show the TE performance of the two algorithms in Fig. 8. We can see from the figure that our hybrid-mode traffic scheduling effectively adapts to uncontrolled CN flows and produces much better overall TE results than the routing developed for the barrier H-SDN.

VIII. Concluding Remarks

In this work, we have studied the traffic engineering problems in the transitional networks of hybrid SDN and CN. We have formulated TE problems in two hybrid modes, *i.e.,* the barrier mode and the hybrid mode. In the barrier mode, SDN traffic and CN traffic are actually routed in separate capacity spaces guided by pre-set capacity provisioning. On the other hand, in the hybrid mode, TE is optimized beyond barriers and link capacities are dynamically shared by the two forms of traffic. We have proposed fast approximate algorithms for the formulated problems in the two hybrid modes respectively. For both scenarios, approximation ratios and running time are theoretically guaranteed.

An instant extension of this paper is to implement the algorithms in hybrid SDN simulators or prototypes with dynamic background traffic and time-varying SDN requests. Numerous networking factors need to be considered. This is an ongoing next-step work that is not presented in this paper.

References

[1] T. Benson, A. Akella, and D. A. Maltz, "Unraveling the complexity of network management." in *Proc. USENIX NSDI*, 2009, pp. 335–348.

[2] J. Martins *et al.*, "ClickOS and the art of network function virtualization," in *Proc. USENIX NSDI*, 2014, pp. 459–473.

[3] N. McKeown *et al.*, "OpenFlow: Enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[4] Infonetics, "SDN, 40G/100G, and MPLS control plane strategies: Global service provider survey," 2012.

[5] S. Jain *et al.*, "B4: Experience with a globally-deployed software defined WAN," in *Proc. ACM SIGCOMM 2013*, 2013, pp. 3–14.

[6] N. Feamster, J. Rexford, and E. Zegura, "The road to SDN," *ACM Queue*, vol. 11, no. 12, pp. 1–21, 2013.

[7] N. McKeown, "Software-defined networking," 2009, INFOCOM Keynote Talk.

[8] S. Vissicchio, L. Vanbever, and O. Bonaventure, "Opportunities and research challenges of hybrid software defined networks," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 2, pp. 70–75, 2014.

[9] R. Roess, E. Prassas, and W. McShane, *Traffic Engineering*. Prentice Hall, 2004.

[10] S. Vissicchio *et al.*, "Safe updates of hybrid SDN networks," Université catholique de Louvain, Tech. Rep., 2013.

[11] M. Chiesa, G. Kindler, and M. Schapira, "Traffic engineering with ECMP: An algorithmic perspective," in *Proc. IEEE INFOCOM*, 2014.

[12] S. Agarwal, M. Kodialam, and T. Lakshman, "Traffic engineering in software defined networks," in *Proc. IEEE INFOCOM*, 2013, pp. 2211–2219.

[13] J. Moy, "OSPF version 2," IETF RFC 2328, 1998.

[14] R. W. Callon, "Use of OSI IS-IS for routing in TCP/IP and dual environments," IETF RFC 1195, 1990.

[15] M. Nascimento *et al.*, "Virtual routers as a service: The routeflow approach leveraging software-defined networks," in *Proc. International Conference on Future Internet Technologies*, 2011, pp. 34–37.

[16] T. Das, M. Caria, A. Jukan, and M. Hoffmann, "A techno-economic analysis of network migration to software-defined networking," 2013, arXiv:1310.0216.

[17] ADVA *et al.*, "Horizon 2020 advanced 5G network infrastructure for future Internet PPP, draft version 2.1," 2013.

[18] ONF, "OpenFlow-enabled SDN and network functions virtualization," 2014.

[19] J. He and W. Song, "Evolving to 5G: A fast and near-optimal request routing protocol for mobile core newtorks," in *Proc. GLOBECOM*, 2014.

[20] W. Jiang, R. Zhang-Shen, J. Rexford, and M. Chiang, "Cooperative content distribution and traffic engineering in an ISP network," in *Proc. ACM SIGMETRICS*, 2009, pp. 239–250.

[21] D. Kreutz *et al.*, "Software-defined networking: A comprehensive survey," 2014, arXiv:1406.0440.

[22] D. Xu, M. Chiang, and J. Rexford, "Link-state routing with hop-by-hop forwarding can achieve optimal traffic engineering," *IEEE/ACM Transactions on Networking*, vol. 19, no. 6, pp. 1717–1730, 2011.

[23] A. Sridharan, R. Guerin, and C. Diot, "Achieving near-optimal traffic engineering solutions for current OSPF/IS-IS networks," *IEEE/ACM Transactions on Networking*, vol. 13, no. 2, pp. 234–247, 2005.

[24] J. He, H. Zhang, B. Zhao, and S. Rangarajan, "A collaborative framework for in-network video caching in mobile networks," in *Proc. IEEE SECON*, 2013, pp. 406–414.

[25] IBM Inc., "IBM ILOG CPLEX Optimizer," http://www.ibm.com/software/commerce/optimization/cplex-optimizer/.

[26] N. Garg and J. Könemann, "Faster and simpler algorithms for multicommodity flow and other fractional packing problems," *SIAM Journal on Computing*, vol. 37, no. 2, pp. 630–652, 2007.

[27] L. K. Fleischer, "Approximating fractional multicommodity flow independent of the number of commodities," *SIAM Journal on Discrete Mathematics*, vol. 13, no. 4, pp. 505–520, 2000.

[28] G. Karakostas, "Faster approximation schemes for fractional multicommodity flow problems," in *Proc. ACM-SIAM Symposium on Discrete Algorithms*, 2002, pp. 166–173.

[29] S. A. Plotkin, D. B. Shmoys, and É. Tardos, "Fast approximation algorithms for fractional packing and covering problems," in *Annual Symposium on Foundations of Computer Science*, 1991, pp. 495–504.

[30] J. Winick and S. Jamin, "Inet-3.0: Internet topology generator," CSE-TR-456-02, University of Michigan, Tech. Rep., 2002.