Routing Stability in Hybrid Software-Defined Networks

Shih-Hao Tseng, Ao Tang, *Senior Member, IEEE*, Gagan L. Choudhury, *Fellow, IEEE*, and Simon Tse

***Abstract*— Software-defined networks (SDNs) facilitate more efficient routing of traffic flows using centralized network view. On the other hand, traditional distributed routing still enjoys the advantage of better scalability, robustness, and swift reaction to events such as failure. There are therefore significant potential benefits to adopt a hybrid operation where both distributed and centralized routing mechanisms co-exist. This hybrid operation however imposes a new challenge to network stability since a poor and inconsistent design can lead to repeated route switching when the two control mechanisms take turns to adjust the routes. In this paper, we discuss ways of solving the stability problem. We first define stability for hybrid SDNs and then establish a per-priority stabilizing framework to obtain stable routing patterns. For each priority class, we discuss three approaches to reach hybrid SDN stability: global optimization, greedy, and local search. It is argued that the proposed local search provides the best tradeoff among cost performance, computational complexity, and route disturbance. Furthermore, we design a system on a centralized controller, which utilizes those algorithms to stabilize the network. The design is implemented and extensively tested by simulations using realistic network information, including a trace of the Abilene network and data from a tier-1 Internet service providers backbone network.**

***Index Terms*— Stability, hybrid SDN.**

I. Introduction

SOFTWARE-DEFINED networking (SDN, an acronym also for software-defined network) has gained momentum among providers of network services, including data centers [1]–[3], wide-area networks (WANs) [4], [5], and cloud computing [6]–[8], as it utilizes resources more efficiently by decoupling the control plane from the data plane and introducing a (logically) centralized controller [9]. On the other hand, the advantages of adopting a centralized controller are accompanied by potential implementation challenges such as compatibility and scalability: Not all devices support full SDN functionality and the centralized controller can be overloaded when the network scales beyond its computational power [10]–[12]. Furthermore, centralized controllers cannot react to events as fast as a local router, especially for WANs.

For the providers that possess well-functioning networks already, e.g., the Internet service providers (ISPs), an attractive approach is to allow a hybrid operation where both centralized and distributed routings coexist. There are several advantages to this approach. For example, if one routing mechanism fails, the other can continue to function, providing great robustness. Another example is that when a change happens in a network spanning a large geographical area, a centralized controller may not be able to take decisions as swiftly as the router located close to the source of change. In this case, a network with hybrid operation can first use the decision made by the local router, although that may be only locally optimal, whereas a globally optimal routing computed by the centralized controller can be kicked in at a later designed time. With all these benefits, the hybrid framework nevertheless also poses new challenges to network management [13]. In this paper, we focus on a critical one: stability.

Stability is of fundamental importance in network routing. Several definitions are proposed to depict the idea [14]–[18]. In general, a stable routing mechanism keeps the same route for the same or similar traffic flow as long as it can. In the presence of multiple routing control units, a stable route is the route that would not be altered by any other routing units [15], [16]. Here, we are interested in how a stable routing pattern can be obtained in a hybrid SDN, where we have two control units: the centralized and the distributed. We define stability for such hybrid SDNs as the consistency of routing decisions made by the centralized controller and the local routers (§ IV). Based on the definition, we further develop a per-priority stabilizing algorithmic framework with three different kernel algorithms to stabilize each priority class: global optimization, greedy, and local search, to pursue stable routing patterns. The three kernel algorithms provide trade-off among time-complexity, cost-effectiveness, and purpose-flexibility (§ IV-A to § IV-C).

We start with examples showing the benefits and challenges of introducing a centralized controller (§ II). In Section III, we introduce the notation, the local source routing mechanism, the way the centralized controller acquires information, and how the centralized and the distributed control update the routing in the network. Then we define, in Section IV, the stability of a hybrid SDN and design algorithms to achieve stability. Partial or inconsistent information scenarios are discussed in Section V followed by the system design (§ VI). Simulation results are included in Section VII and we conclude the paper in Section IX.

Manuscript received October 22, 2017; revised June 18, 2018 and November 22, 2018; accepted January 25, 2019; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor T. Spyropoulos. Date of publication March 11, 2019; date of current version April 16, 2019. *(Corresponding author: Ao Tang.)*

S.-H. Tseng is with the Division of Engineering and Applied Science, California Institute of Technology, Pasadena, CA 91125 USA (e-mail: shtseng@caltech.edu).

A. Tang is with the School of Electrical and Computer Engineering, Cornell University, Ithaca, NY 14853 USA (e-mail: atang@ece.cornell.edu).

G. L. Choudhury and S. Tse are with AT&T Labs, Middletown, NJ 07748 USA (e-mail: gc2541@att.com; st2196@att.com).

Digital Object Identifier 10.1109/TNET.2019.2900199

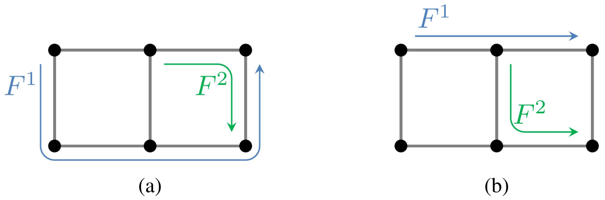


Fig. 1. A simple example showing that distributed routing can end up in stalemate. (a) Stalemate. (b) Optimal solution.

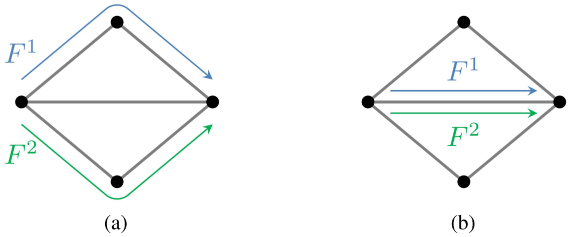


Fig. 2. A simple example showing that the inconsistency between the centralized controller and the local routers can cause instability. (a) Routing pattern preferred by the centralized controller. (b) Routing pattern preferred by the local routers.

II. Background

Routing is an essential functionality that a network needs to provide in order for users to send their traffic through, and one key property of a desirable routing mechanism is stability. A stable routing mechanism should not keep changing the route decisions if all the inputs remain the same. Otherwise, packets may be lost during the transition and more power is consumed to amend the routing table.

Nowadays, network traffic is usually routed by distributed routing protocols such as Open Shortest Path First (OSPF) and Border Gateway Protocol (BGP). The robustness and stability of these distributed routing protocols have been well-studied in the literature [14], [16], [19]. However, the distributed routing can only achieve a locally optimum routing pattern and may deviate from globally optimum routing.

Without coordination, distributed routing may result in a stable but inefficient routing pattern. For instance, two routers perform shortest-path source routing independently in Figure 1. All the edges have unit capacity and two flows F^1 and F^2 both require unit sending rate. Fig. 1a shows a case when no flow can find a shorter path given the existence of the other flow. However, under the intervention of a centralized controller, it is still possible to reach a state in which F^1 takes a shorter path (Fig. 1b).

The simple example above explains the motivation for the network operator to introduce a coordinator with global view to help resolve the stalemate and improve the utilization of the network. SDN is a perfect framework to deploy such a coordinator – using the SDN centralized controller.

Nevertheless, the network operators who have well-functioning networks already, like the ISPs, may not want to switch to a single centralized controller for reasons explained earlier. Instead, they may adopt a hybrid SDN approach, in which the centralized controller cooperates with the local routers to route the traffic.

This dual control framework has significant benefits over pure distributed or pure centralized routing. References [20] and [21] provide a perfect example showing that the centralized controller can help the network operator improve their efficiency significantly compared to the pure distributed case.[1] Also, a hybrid framework preserves the robustness and the swiftness provided by the distributed routing: Fast reroute can be performed to react to failures swiftly and maintain the packet level robustness. And the flow level post-failover convergence period can be shortened with the help of the centralized controller.

There are many possible hybrid SDN designs. For instance, the controller can take full control of the network and let the distributed routing handle only the path failures (like IBSDN [22]); the controller can alter the view of distributed routing to manage the routes (such as Fibbing [23]); or the controller can manage only parts of the network [13]. A critical fact is: no matter which approach we take in a hybrid approach, the controller should be able to override the distributed decisions in whole or in part. Given the fact, the minimum feasible design of a hybrid SDN network is to allow the controller to change the distributed routes.[2]

From the ISPs' perspective, such design avoids the need of upgrading distributed devices to support sophisticated mechanisms, and hence it is a better choice of incremental SDN deployment. However, it also introduces new stability challenges if the centralized controller is not designed carefully.

In Fig. 2, we demonstrate that the inconsistency of the routing decisions between the two control units can result in consecutive route switching. Consider two flows F^1 and F^2 which are routed through the network. A poorly designed centralized controller may prefer the routing pattern as in Fig. 2a, while the local routers would choose the routes as in Fig. 2b because they are the shortest paths. After the centralized controller deploys the left routing pattern, the local routers override the decision by the right routing pattern. This process may continue in poorly designed centralized controller.

Besides the concerns from dual control, the partial or inconsistent information can also drive the network into an unstable state. For instance, if the centralized controller in Fig. 2 cannot detect the middle link, Fig. 2b will never be a feasible solution for the controller and instability may result It should be pointed out that the above scenario is rare and non-persistent in practice, and if it happens the best approach is to simply rely on distributed routing as long as the problem lasts. However, we will explore other ways of addressing those imperfect information situations as well.

III. Formulation

In this section, we introduce a flow-level model to depict our problem and illustrate our system design in the following

[1] We refer the reader to [20, Fig. 2] and the explanations in [21] for more details.

[2] The centralized controller may be granted only partial controllability over the network. In this paper, we assume full controllability for simplicity.

sections. The objective of this work is to ensure the centralized controller deploys a routing pattern that is *stable*, i.e., consistent with the distributed routing mechanism described in Section III-B and Assumption 1 in Section IV. The framework and methods proposed in the following sections are devoted to this purpose, regardless of the *performance metric* of the network operator. That is, no matter which performance metric the network operator possesses, such as minimizing the overall cost or maximizing the total throughput, our methods can still find a stable routing pattern.

A. Notations

We denote by t the physical time of the system. A variable can attach parenthesized t to refer to its value at time t.

The network is modeled as a directed graph $G = (V, E)$, where V is the set of nodes representing the local routers performing source routing and E is the set of directed edges representing the physical links between the routers. Each edge $e \in E$ has a capacity c_e and a cost metric m_e, which are both fixed constants. The connectivity of the edge e is indicated by a binary variable $z_e(t)$, which is 1 if the edge is up and 0 when it is down.

A set of flows indexed by the set N sends their traffic through the network. Each flow F^n, where $n \in N$, requires to send at the rate $r^n(t)$ from its source s^n to its destination d^n. s^n and d^n are connected by a path specified via the path indicator $x_e^n(t)$, which is 1 when the path includes edge e and 0 otherwise. We omit the subscript e of $x_e^n(t)$ to refer to the path as a vector of path indicators, and its corresponding cost is expressed by the shorthand notation $m(x^n(t)) = \sum_{e \in E} m_e x_e^n(t)$. To ensure $x^n(t)$ forms a path, an additional condition $x^n(t) \in \mathcal{P}^n$ is introduced.[3]

An ISP usually maintains at least two priority levels to provide different quality of services and pricing/marketing options. For instance, VoIP or video conference services might get higher priority than emails [25]. Therefore, we associate a priority class π^n with each flow F^n, and in this work the 3-tuple $< s^n, d^n, \pi^n >$ uniquely defines a flow. We say $\pi^1 \le \pi^2$ if the priority class π^1 has higher priority than π^2. The flows with (strictly) higher priority can acquire bandwidth from lower prioritized flows. We refer to the indices of the flows higher prioritized than π by $N_{\le \pi} = \{n \in N : \pi^n \le \pi\}$, and we let $N_\pi = \{n \in N : \pi^n = \pi\}$. We denote by $\Pi = \{\pi^n : n \in N\}$ the set of all priority classes.

Table I summarizes major notations for easier lookup.

TABLE I

MAJOR NOTATIONS

c_e	edge capacity
m_e	edge cost metric ("edge length")
$z_e(t)$	edge connectivity indicator

for each flow F^n, $n \in N$

$r^n(t)$	sending rate
(s^n, d^n)	(source,destination)
π^n	priority class
$x_e^n(t)$	path indicator of edge e
$x^n(t)$	vector of path indicators
$m(x^n(t))$	corresponding cost of $x^n(t)$
$x^n(t) \in \mathcal{P}^n$	condition that $x^n(t)$ forms a path

for priority class $\pi \in \Pi$

$N_{\le \pi}$	indices of the higher prioritized flows
N_π	indices of flows with priority π

B. Distributed Routing

In this work, each flow F^n is routed via solving the constrained shortest-path first (CSPF) problem $R^n(t)$ at its source router, which finds a path among a given set of paths from the source to the destination that has the minimum cost:

$$R^n(t) = \min\ m(x^n(t))$$

$$\text{s.t. } x^n(t) \in \mathcal{P}^n \tag{1a}$$

$$x_e^n(t) \in \{0, 1\} \quad \forall e \in E \tag{1b}$$

$$x_e^n(t) \le z_e(t) \quad \forall e \in E \tag{1c}$$

$$\sum_{n' \in N_{\le \pi^n}} r^{n'}(t) x_e^{n'}(t) \le c_e \quad \forall e \in E \tag{1d}$$

where the constraints (1a) and (1b) require that $x^n(t)$ be a path; the constraint (1c) ensures that the path can only take the up edges; and the constraint (1d) is the link capacity constraint. Notice that $R^n(t)$ is polynomial-time solvable: By setting $x_e^n(t) = 0$ for all the edges with $z_e(t) = 0$ and removing the constraint (1c), the problem is a shortest-path problem, which is polynomial-time solvable [24].

When equal-cost path solutions exist, only one of them is picked as the solution based on some tie-break rules provided by the system operator. In this work, we use the terms "path" and "route" interchangeably since single path is chosen as the route for the traffic.

C. Information Structure

As in [20], the centralized controller collects the data plane information via two different protocols:

- Path Computation Element Communication Protocol (PCEP, RFC 5440 [26]): The router can report $r^n(t)$ by the path computation request message and $x^n(t)$ by the path computation reply message to the centralized controller. The PCEP messages are marked by $< s^n, d^n, \pi^n >$, so that the flow can be identified.
- Border Gateway Protocol - Link-State (BGP-LS, RFC 7752 [27]): The centralized controller gathers the link-state information of each edge, which includes c_e, m_e, $z_e(t)$, and the aggregated traffic rate on the edge per priority $\sum_{n \in N_\pi} r^n(t) x_e^n(t)$.

Local routers report the information through these two protocols on a regular basis, but the reporting time is not

[3] Using the notations from [24, Ch. 7.3], we can express $\mathcal{P}^n$ as the set such that

$$\sum_{e \in \delta(S)} x_e^n(t) \ge 1, \quad \forall S \in \mathcal{S}.$$

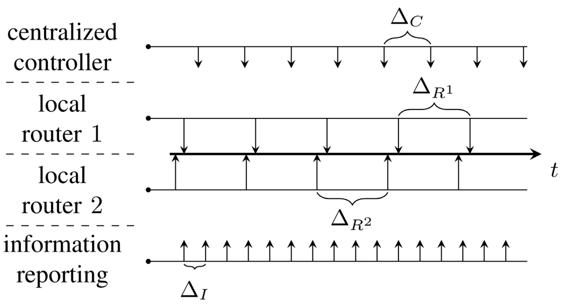


Fig. 3. The system is not necessarily synchronized. Each control unit/reporting protocol has its update interval.

necessarily synchronized. We assume that the centralized controller records the receiving time as well when collecting those reported information. The receiving time helps the centralized controller detect outdated information and allows further measures to be taken.

D. Update Timeline

The system updates in an asynchronous manner. Fig. 3 shows the update timeline of the system. The centralized controller collects the information and routes the traffic every Δ_C time unit. Similarly, each local router performs CSPF source routing $R^n(t)$ routinely with the interval Δ_{R^n}. Notice that the intervals Δ_{R^n} are not necessarily the same among all routers. If the local router computes the route for the flow F^n at time t, it will compute again at time $t+\Delta_{R^n}$. However, local routers will perform fast-reroute (by solving $R^n(t)$) whenever they find that the assigned routes are no longer feasible. PCEP and BGP-LS messages are sent in a much higher frequency than the route updating. The same message will be sent every Δ_I time units, while different messages are not synchronously sent. For simplicity, we assume that the information reporting interval, Δ_I, is a static value. In reality, the interval size may vary from message to message. However, we ignore the effect caused by varying interval size, since the reporting interval is much shorter than the update interval.

IV. Dual Control Consistency

The stability of a hybrid-software defined network involves the consistency between the centralized routing performed by the centralized controller and the distributed routing performed by the individual local routers. If these two control units are not consistent with each other, the routing decision may be overturned repeatedly as they take turns to modify the routes. Before proceeding to the dual control consistency, we should specify the underlying assumption about the behavior of local routers to ensure the stability of the distributed routing.

Assumption 1: A local router will not change the selected path for a flow unless any one of the following is encountered:

- *The centralized controller orders it to do so.*
- *The old path is no longer feasible.*
- *A new feasible path with strictly lower cost exists.*

Algorithm 1 Algorithmic Framework to Obtain Stable Routing Patterns

```
for From the highest priority $\pi \in \Pi$ tothe lowest do
    Invoke the kernel algorithm to get a stable routing pattern
    of the flows indexed by $N_\pi$ and deploy the corresponding
    routes.
end for
```

The assumption results from the fact that the local routers should not switch between equal cost paths, otherwise the distributed routing itself is not stable.

Given the assumption, we define the stability of a hybrid software-defined network below, similar to the way in [15] and [16]. It basically says that the hybrid SDN is stable as long as the centralized controller's decision is consistent with the local routers' decisions.

Definition 1: A hybrid software-defined network is stable *if the centralized controller deploys a routing pattern that is an optimal solution to $R^n(t)$ for all $n \in N$.*

Definition 1 matches the stability of a system in the ordinary sense: The assigned routes will not be switched back and forth. The reason is that a path corresponding to an optimal solution to $R^n(t)$ is feasible and admits no path with strictly lower cost. Therefore, based on Assumption 1, once the centralized controller deploys a routing pattern as described in Definition 1, the local routers will not change the selected paths since they are already optimal.

With Definition 1, we can design algorithms for the centralized controller to achieve a stable routing pattern. Since the flows are prioritized, it is intuitive to cope with the higher prioritized flows first. The intuition stems from the fact that the higher prioritized flows can acquire bandwidth from lower prioritized flows. If a lower prioritized flow is routed first, its bandwidth can still be taken by a higher prioritized flow, which leads to rerouting.

Therefore, we propose Algorithm 1 as a framework to pursue a stable routing pattern. The *kernel algorithm* in Algorithm 1 is another algorithm which gives a stable routing pattern for the flows in a priority class, with all higher prioritized flows routed already.

We show that Algorithm 1 generates a stable routing pattern as follows. If the generated pattern were not stable, there would exist a flow F^n which can find a path with strictly lower cost. Such path cannot occupy the bandwidth of the flows with priority higher than π^n. Therefore, at the iteration that considers the flows indexed by N_{π^n}, the path with strictly lower cost for F^n is feasible. That means the routing pattern given by the kernel algorithm is not stable, which contradicts to the definition of the kernel algorithms.

In the following subsections, we propose different kernel algorithms to obtain a stable routing pattern for a priority class.

A. Global Optimization Kernel Algorithm (GLO)

One way to obtain a stable routing pattern for a priority class π is by solving the following optimization problem $C_\pi(t)$, which globally minimizes the aggregated metric of the flows at the priority class π.

$$\begin{aligned} C_\pi(t) = \min \sum_{n\in N_\pi} & m(x^n(t)) \\ \text{s.t. } & x^n(t) \in \mathcal{P}^n \quad \forall n \in N_\pi \\ & x^n_e(t) \in \{0,1\} \quad \forall n \in N_\pi,\ e \in E \\ & x^n_e(t) \le z_e(t) \quad \forall n \in N_\pi,\ e \in E \\ & \sum_{n\in N_{\le\pi}} r^n(t)x^n_e(t) \le c_e \quad \forall e \in E \end{aligned} \tag{2}$$

The global optimization kernel (GLO) routes the flows based on the resulted $x^n_e(t)$.

The optimal solution to $C_\pi(t)$ is stable, which can be shown by contradiction: If not, there exists $n \in N_\pi$ such that the optimal solution to $C_\pi(t)$ is not an optimal solution to $R^n(t)$. As such, we can substitute the optimal solution $x^n_e(t)$ to $R^n(t)$ back to the optimal solution to $C_\pi(t)$, which results in a feasible solution to $C_\pi(t)$ with strictly lower cost than the optimal solution, and it is not possible.

The optimal solution to $C_\pi(t)$ guarantees not only the stability but also the lowest cost in the presence of the route assignment to higher prioritized flows. Unfortunately, obtaining an optimal solution to $C_\pi(t)$ is computationally intractable in general as $C_\pi(t)$ is a min-cost multi-source unsplittable flow problem, which is NP-hard [28]. Furthermore, even the single-source version is strongly NP-complete: determining the existence of a feasible solution to the single-source unsplittable flow problem is NP-complete [29, Proposition 3.0.1]. As a result, the major approximation results target three slightly tweaked versions that were first proposed in [29], which consider minimum congestion, maximum satisfiable subset, and minimum number of transmission rounds. We should point out that a routing pattern given by approximating $C_\pi(t)$ may lead to a low cost solution but the resulting pattern may not be stable.

Although the optimality of $C_\pi(t)$ guarantees the stability, $C_\pi(t)$ may be infeasible. Dealing with the infeasibility is a decision that would be made by the network operator. The network operator may try to satisfy a chosen subset of the flows or send the flows in reduced rates. We don't make any assumption on how the network operators deal with infeasibility, and our results, including the followings, can still find stable routing patterns.

The NP-hardness of $C_\pi(t)$ makes it hard to be solved efficiently. Therefore, we propose two other tractable methods in the following subsections.

B. Greedy Kernel Algorithm (GRE)

While NP-hardness prevents us from solving the global optimization problem $C_\pi(t)$, solving so is not necessary for obtaining a stable routing pattern for a priority class. For instance, there are two different stable routing patterns in Fig. 1, even though not both of them incur the lowest cost. Thus, we propose in the following subsections some approaches other than the global optimization to obtain a stable routing pattern.

We propose a greedy approach based on the following observation: Given a stable routing pattern and a new flow F^n, adding the path resulting from $R^n(t)$ on top of the given stable routing pattern yields another stable routing pattern. As such, we can build a stable routing pattern by adding the route from $R^n(t)$ one at a time, which results in the greedy kernel algorithm (GRE, Algorithm 2).

Algorithm 2 Greedy Kernel Algorithm (GRE)

```
for Choose $n \in N_\pi$ in an arbitrary order do
    Solve $R^n(t)$ and deploy the resulted path when $R^n(t)$ is feasible.
end for
```

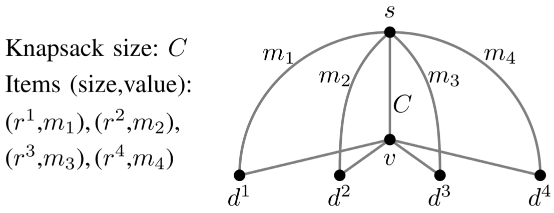


Fig. 4. NP-hardness of deciding the best greedy solving order can be shown by a polynomial-time reduction from the Knapsack problem.

GRE generates stable routing patterns. If the routing pattern given by Algorithm 2 were not stable, we would have a feasible path with strictly lower cost for some flow F^n. The path is then feasible at the iteration when F^n is chosen, which means the path selected by Algorithm 2 is not optimal to $R^n(t)$, and it leads to a contradiction.

GRE has one major drawback as shown in Fig. 1. That is, the performance of the algorithm depends on the solving order of $R^n(t)$. Solving $R^2(t)$ before $R^1(t)$ gives Fig. 1a; nevertheless, Fig. 1b can be reached by solving $R^1(t)$ first. Thus, better performance may be achieved by carefully aligning the solving order, however, deciding the best solving order itself is NP-hard as shown in Proposition 1.

Proposition 1: Finding a solving order of GRE that minimizes $\sum_{n\in N_\pi} m(x^n(t))$ *is NP-hard.*

Proof: We show the proposition by providing a polynomial-time reduction from the Knapsack problem, which is NP-hard. Fig. 4 shows the construction in the proof below.

Given an instance of the Knapsack problem, we construct a directed graph consisting of one edge (s, v), which has a capacity equal to the knapsack size. For each item n with size r^n and value m_n, we add a node d^n and two directed edges (s, d^n) and (v, d^n) with capacities r^n to the graph. All the edges are zero cost except for the edges (s, d^n), which have the cost metric m_n, respectively.

We create the flows F^n to send traffic from s to d^n at the rate r^n. Since each flow F^n has only two possible paths: $s \to d^n$ with cost m_n or $s \to v \to d^n$ with zero cost, deciding the best solving order is equivalent to answering what is the highest aggregated associated cost m_n given by the flows going through (s, v), which is the objective of the Knapsack instance. Therefore, the construction gives a polynomial-time reduction from the Knapsack problem. □

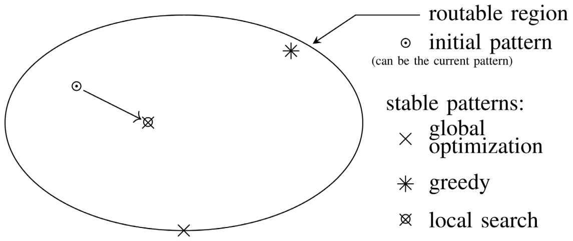


Fig. 5. Local search kernel finds a "nearby" stable routing pattern.

Notice GRE will always produce a *routable* routing pattern, i.e., the routed flows will respect the capacity constraints. However, it is not necessary that all the flows will be routed.

C. Local Search Kernel Algorithm (LOC)

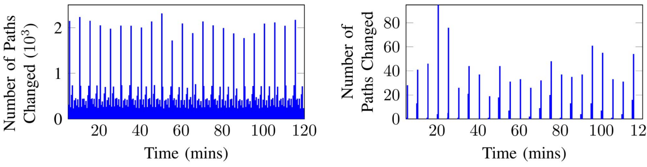
GRE builds a stable routing pattern for a priority class from scratch. However, the centralized controller may have derived a routable routing pattern from some heuristics already. For example, the controller may adopt some approximation algorithm, such as [30]–[33], to approach $C_\pi(t)$. As mentioned in Section IV-A, such approximation solution may not form a stable routing pattern. The only question left is how to shape the existing routable routing pattern to be a stable one. To deal with the situation, we take a local search approach. A local search kernel algorithm (LOC) finds a "nearby" stable pattern by "improving" the routable solution until no further change can be made (Fig. 5). In this case, we have to clarify the meaning of "improvement" such that the termination of LOC implies the stability of the resulted solution.

Definition 1 sheds light on the possible termination condition: A routing pattern is stable if there exists no $n \in N_\pi$ such that the routing pattern is not an optimal solution to $R^n(t)$. As a result, we can define the improvement as "finding $n \in N_\pi$ such that the routing pattern is not an optimal solution to $R^n(t)$" and improving the maintained routable solution by using the strictly shorter route given by the optimal solution to $R^n(t)$. The design of LOC is summarized in Algorithm 3, and we show its stability by contradiction: Suppose the routing pattern were not stable, we would have a path with strictly lower cost for some flow F^n. However, the routing pattern is generated when LOC terminates, which happens only when all $n \in N_\pi$ are "checked". A "checked" flow can have a path with strictly lower cost only if some other flow modifies its path, but meanwhile the modified flow marks all other flows " unchecked", and LOC will not terminate. Thus, the resulted routing pattern must be stable.

An important question about LOC is whether the algorithm will terminate in polynomial-time. Proposition 2 confirms that LOC terminates in $O(|N_\pi|^2)$ iterations, and hence the Algorithm 1 with LOC as its kernel terminates in $O(|N|^2)$.

Algorithm 3 Local Search Kernel Algorithm (LOC)

```
Invoke the controller's heuristic to get an initial routable routing pattern.
Put $n \in N_\pi$ in a circular buffer in an arbitrary order and mark them "unchecked".
Let $\hat{n}$ point to one element in the circular buffer.
while There exists $n \in N_\pi$ unchecked do
    Solve $R^{\hat{n}}(t)$.
    if The resulted path for $F^{\hat{n}}$ has strictly lower cost than the current path. then
        Deploy the new path and mark all elements in the circular buffer "unchecked".
    end if
    Mark $\hat{n}$ "checked" and point $\hat{n}$ to the next element.
end while
```



(a) (b)

Fig. 6. The local search kernel algorithm (LOC) can shape a routable routing pattern to be a stable one. (a) Careful Bounded Greedy (CBG) Approximation Algorithm. (b) Stabilizing CBG by Local Search Kernel Algorithm (LOC[CBG]).

Proposition 2: Given an initial routable routing pattern with the path indicators $x^n(t)$, $n \in N$, LOC terminates in $K_\pi |N_\pi|^2$ iterations, where

$$K_\pi = \max_{n \in N_\pi} |\{y^n \in \mathcal{P}^n : m(y^n) \le m(x^n(t))\}|$$

is the maximum number of shorter paths that each flow indexed by N_π has.

Proof: We show the proposition by considering the longest execution. Notice that the while loop in LOC will terminate if we mark $\hat{n}$ "checked" in $|N_\pi|$ consecutive iterations. Therefore, for every $|N_\pi|$ iterations, the longest execution has at least one $\hat{n} \in N_\pi$ which can find a strictly lower cost route via solving $R^{\hat{n}}(t)$. Because there are at most K_π shorter paths, or K_π smaller different costs, the longest execution has at most $|N_\pi|$ (every $|N_\pi|$ iterations) $\times |N_\pi|$ (possible flows) $\times K_\pi$ (times of cost decreasing) $= K_\pi |N_\pi|^2$ iterations before termination. □

A naive upper bound on K_π is $2^{|E|}$, the number of all edge combinations. As a result, although the LOC is quadratic in the number of the flows, the naive bound leads to a factor exponential in the number of the edges, which suggests a potential long worst-case termination time in theory.

However, if the initial routing pattern is chosen such that the initial paths are short already, K_π would be small and hence LOC is efficient. It can happen when the controller leverages LOC to deal with minor deviation from the desired routing pattern, in which all the paths may be chosen close to the shortest paths. For instance, when some high prioritized flow reduces its sending rate, the released bandwidth allows some low prioritized flows to take shorter paths. LOC will be able to efficiently deal with that case as the released bandwidth creates only few shorter possible paths in addition to the current ones.

We introduce the following example in Fig. 6 to illustrate how LOC can be useful.

TABLE II
COMPARISON OF ALGORITHM 1 WITH DIFFERENT KERNELS

	GLO	GRE	LOC
Cost Effectiveness	lowest	depending on the order	depending on the order and the initial pattern
Time Complexity	in general $O(2^{\|N\|})$	$O(\|N\|)$	$O(K_\pi \|N\|^2)$
Initialization Allowance	no	no	yes

The centralized controller may approximate $C_\pi(t)$ by the careful bounded greedy (CBG) heuristic [30], [31], which approximately finds a subset of flows that accounts for the most traffic. Such approximation algorithm has a fixed approximation ratio proven in [30], but the resulting pattern may not be stable. We perform a trace-based simulation as in Section VII-C, and the number of routes flapped are shown in Fig. 6.

The network traffic changes every five minutes, and we examine two different kernel algorithms: CBG only (Fig. 6a) and LOC with CBG generating the initial pattern (denoted by LOC[CBG], Fig. 6b). The routing patterns by CBG are not stable: the local routers and the centralized controller take turns to adjust the routes. In contrast, LOC stabilizes the routing pattern generated by CBG, and the network is stabilized quickly after the centralized controller intervenes.

D. Comparison Amongst the Kernels

We summarize Algorithm 1 with three different proposed kernels in Table II.

Since solving the global optimization problem $C_\pi(t)$ is NP-hard, it will take exponential-time to solve; GRE checks each flow only once, and hence it is linear-time solvable; and the time-complexity of LOC has been given by Proposition 2.

We can observe from the table that GLO and GRE are two extreme cases. Solving $C_\pi(t)$ is the most computationally expensive with the lowest (optimal) cost (per priority class), while GRE loses such optimality in exchange for lower computational complexity.

Besides these two extremes, LOC provides flexibility with a quadratic-time complexity by allowing the specification of an initial routable routing pattern. The importance of flexibility is argued in the following sense: Unless the network operator aims to minimize the same objective as $C_\pi(t)$, it may select a stable routing pattern based on some other criterion, such as route disturbance. In that case, the flexibility allows the operator to pursue the criterion as well as the stability in the same time. For example, if we specify the initial pattern as the current pattern, the local search algorithm may change much less routes than the other two kernels since it searches locally (which is confirmed in Section VII).

We remark that when only one priority class is considered, GLO simply solves the multi-source unsplittable flow problem and GRE converges as the legacy distributed routing. However, centralized control is still beneficial, since it can shift the network to a preferred routing pattern (Fig. 1) and quickly stabilize the network.

V. PARTIAL OR INCONSISTENT INFORMATION

In Section IV, we develop a framework with three different kernels to reach a stable routing pattern assuming complete and up-to-date information. If the centralized controller does not get complete information, i.e., some information is missing or inconsistent, the preferred approach for it is to pause the centralized controller's intervention until complete information is available. This approach works in the hybrid framework since the distributed routing continues to work, and it is preferable for the ISPs that operate highly robust networks. In a robust network, partial or inconsistent information is seldom encountered, and the system recovers from the abnormal information state quickly. Therefore, the centralized controller can soon resume working without any additional complex functions. For the sake of a complete design, we also consider a second approach where centralized controller takes action even if the information is incomplete in this section.

As to how the centralized controller can pursue a stable routing pattern under incomplete or inconsistent information, we decouple the issue into two stages: information recovery and stability pursuit. During the information recovery stage, the centralized controller tries to deduce the missing information from the available information or mitigate the impact of the inconsistent information. Then it pursues stability as if complete information is given, which has been examined in Section IV. We will hence focus on the first stage in this section.

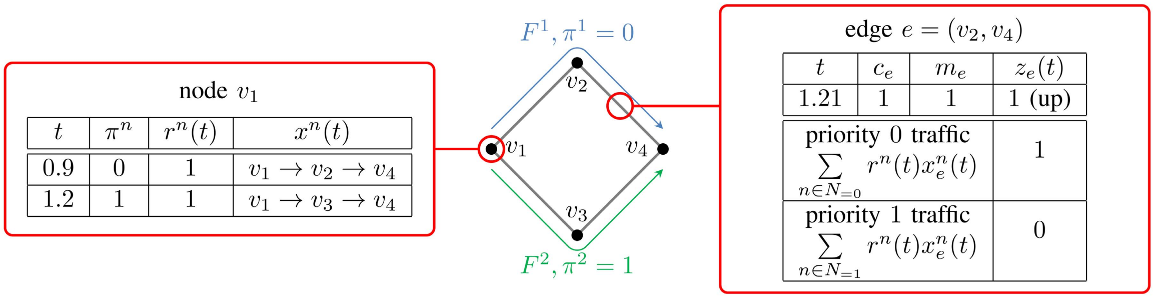
A. Partial Information

Since the centralized controller relies on the data plane to collect information, the information may be lost or delayed during the packet delivery. Also, the failures of routers or links prevent the centralized controller from probing the current states. Those reasons explain why the centralized controller may need to route based on partial information.

One simple observation of information recovery is that no information can be recovered if no information is available for the centralized controller. It motivates us to focus on the case in which only small part of the information is missing instead of those general cases such as a total blackout. Specifically, we borrow the idea of N-1 criterion [34] from power systems and provide our modified definition as follows.

Definition 2: The N-1 criterion *requires full information recovery of a variable when one protocol message is lost.*

We will show that N-1 criterion can be met for the flow rate $r^n(t)$ and the flow path $x^n(t)$, while the information recovery of the edge connectivity $z_e(t)$ is not guaranteed. We remark that meeting the N-1 criterion does not mean that we can only restore one missing variable, but that the system is robust enough to endure one protocol message lost without being blind to the variable. The system can still recover from the loss of multiple protocol messages when the messages are independent.



node v_1

t	π^n	$r^n(t)$	$x^n(t)$
0.9	0	1	$v_1 \to v_2 \to v_4$
1.2	1	1	$v_1 \to v_3 \to v_4$

edge $e = (v_2, v_4)$

t	c_e	m_e	$z_e(t)$
1.21	1	1	1 (up)

priority 0 traffic $\sum_{n \in N_{=0}} r^n(t) x_e^n(t)$	1
priority 1 traffic $\sum_{n \in N_{=1}} r^n(t) x_e^n(t)$	0

Fig. 7. The PCEP and BGP-LS information is marked with time stamps and stored as a graph.

Recall the information carried by the different protocol messages:

- PCEP: $< s^n, d^n, \pi^n >$, $r^n(t)$, and $x^n(t)$.
- BGP-LS: c_e, m_e, $z_e(t)$, and $\sum_{n \in N_\pi} r^n(t) x_e^n(t)$ for each priority class π.

The 3-tuple $< s^n, d^n, \pi^n >$ is used to identify the flow. If it is missing, the corresponding $r^n(t)$ or $x^n(t)$ is missing as well. The PCEP information $r^n(t)$ and $x^n(t)$ are linearly dependent on the aggregated flow information $\sum_{n \in N_\pi} r^n(t) x_e^n(t)$ provided by BGP-LS. As such, one flow information can be fully-restored per priority class via linear algebra, which suggests that $r^n(t)$ and $x^n(t)$ meet the N-1 criterion.

We illustrate the benefit of meeting such N-1 criterion by the following proposition.

Proposition 3: Assume that each PCEP message gets lost with probability q, independent with each other. With information recovery, the probability that the controller still has complete information is at least

$$(1-q)^{|N|-|\Pi|} \prod_{\pi \in \Pi} \left(1 + (|N_\pi| - 1)q\right),$$

which is lower bounded by the probability that at most one PCEP message is lost within $|N| - |\Pi| + 1$ *messages:*

$$(1-q)^{|N|-|\Pi|} \left(1 + (|N| - |\Pi|)q\right).$$

Proof: The probability that at most one PCEP message is lost within y messages is

$$\prod_{\pi \in \Pi} (1-q)^{y-1} \left(1 + (y-1)q\right).$$

Since the controller can restore one PCEP message per priority class using BGP-LS information (N-1 criterion), the probability that the controller has complete information is

$$\prod_{\pi \in \Pi} (1-q)^{|N_\pi|-1} \left(1 + (|N_\pi| - 1)q\right),$$

and the proposition is derived through simple calculations. □

Without information recovery, the probability that the controller has complete information is $(1-q)^{|N|}$. In general, q is small and $|N|$ is large. Therefore, information recovery can help improve the probability that the centralized controller maintains complete information.

On the other hand, the PCEP information π^n and the BGP-LS information c_e, m_e, and $z_e(t)$ do not have the dependency, and N-1 criterion is not met. However, π^n, c_e, and m_e are time-independent. It is less likely to lose the information. As to $z_e(t)$, it can be related to $x^n(t)$ through the constraint (1c). For any $e \in E$, $x^n(t) \le z_e(t)$ should be satisfied for all $n \in N$. As such, $z_e(t) = 1$ if there exists a flow F^n routed through the edge e. Therefore, revealing $z_e(t)$ is still possible unless no flow is going through the edge. In that case, we simply assume $z_e(t) = 0$ (i.e., the edge e is down) until some flow is routed through the edge by a local router.

B. Inconsistent Information

In addition to the partial information issue, it is potentially possible that the latest information may be inconsistent as they are reported at different time. For instance, the sending rate of F^1 in Fig. 7 drops from one unit traffic to 0.4 unit traffic. When the centralized controller intervenes, the new PCEP information has been received, and $r^1(t)$ is updated to 0.4. But, the new BGP-LS on edge e has not yet propagated to the centralized controller, and hence the centralized controller finds that one unit priority 0 flow is going through e from the obsolete BGP-LS.

To tackle the conflict view from the two information sources, we calculate the *effective capacity*, which is the minimum possible available capacity, and solve for stable routing patterns based on that conservative capacity estimation. The idea behind effective capacity is to avoid occupying the bandwidth that is being used but not well detected. Among the reported information of a variable, we take the minimum as its value. Meanwhile, we estimate the maximum possible amount of "hidden flows" and deduct them from the capacity. By doing so, a routable solution based on the effective capacities remains routable even under the presence of hidden flows, which prevents packet dropping caused by unawareness of the hidden flows and occupation of their bandwidth.

We calculate the effective capacity for the example in the beginning of this section: We deem $r^1(t) = 0.4$ as it is the minimum reported value of $r^1(t)$. As such, 0.6 priority 0 flow on edge e is unseen (the hidden flow), and the effective capacity of e is computed as $1 - 0.6 = 0.4$. On the other hand, if the capacity of e is 2 instead, the effective capacity will be $2 - 0.6 = 1.4$.

We have the following proposition discussing when the idea of effective capacity can maintain a routable solution under inconsistent information.

Proposition 4: A routable routing pattern given by the controller under effective capacities remains routable when

- *the PCEP information of some new flows are not yet known by the controller.*
- *the actual sending rates are no more than the reported values.*

Proof: Under the first circumstance, the unknown flows are reported by BGP-LS but not PCEP, and hence the controller won't redirect those flows. A routable routing pattern given by the controller fits other known flows into the network, and the bandwidth for the unknown flows are reserved by the design of effective capacity.

The second one is trivial. Since the actual sending rates are no more than the reported values, and the effective capacity is always smaller than the real capacity, satisfying the reported sending rates under effective capacity implies the satisfaction of the actual sending rates under the real capacity. □

Notice that Proposition 4 guarantees *feasibility* instead of *stability* of the pattern given by the controller. To obtain a stable pattern, complete information is always favorable than inconsistent one.

VI. System Design

In this section, we elaborate how to build a centralized controller that generates stable routing patterns.

A. Information Collection

The centralized controller keeps the received information from PCEP and BGP-LS messages as a graph. The PCEP messages are translated as π^n, $r^n(t)$, and $x^n(t)$, and stored at the source nodes. The BGP-LS messages update c_e, m_e, $z_e(t)$, and $\sum_{n \in N_{=\pi}} r^n(t)x_e^n(t)$ that are stored at the edges. When a new message is received, the centralized controller checks if there exists an entry corresponding to the message already. If so, it updates the entry and resets the time stamp t as the current time. Otherwise, it creates a new entry to record the message and sets the time stamp as the current time.

Fig. 7 is an example with two flows F^1 and F^2. Each flow sends unit traffic from v_1 to v_4. The network consists of four unit-capacity edges. The information reported to the centralized controller is stored in a corresponding graph.

B. System Workflow

At the moment when it is the centralized controller's turn to update the routes, it goes through the following process phase by phase:

1) **Outdated information removal**: In the phase, the centralized controller identifies if the stored information is outdated by comparing the time stamp, which is the last receiving time, with the current time. The centralized controller collects only the information which is either time-independent or within an appropriate timeout Δ_T. Δ_T must be larger than Δ_I in order not to discard the latest information.
2) **Information recovery**: The centralized controller applies the skills discussed in Section V to the collected information from the first phase to obtain a recovered information.
3) **Stable routing pattern generation**: Based on the recovered information and the effective capacities of the edges, we then adopt Algorithm 1 with different kernels to find stable routing patterns.
4) **Route deployment**: The centralized controller sends out PCEP commands to the local routers to adjust the routes of the traffic.

VII. Simulations

We evaluate our methods by implementing the system described in Section VI and conducting simulations regarding their statistical and large-scale performance.

A. Setup

We build a simulation system as a platform to test our methods. The system structure has been discussed in Section VI. Following parameters are adopted for the simulations: $\Delta_C = 30$ seconds, $\Delta_{R^n} = \Delta_R = 4$ minutes for all the flows, and $\Delta_I = 10$ seconds. We ignore the route setup time in our simulation. In practice, it takes at least 1.5 round-trip time (RTT) for traffic to follow an established new route (1 RTT for the new route establishment and 0.5 RTT for the traffic to follow a new route). Also, we don't consider the fluctuation during the route change. The only focus of the following simulations is the establishment of a stable routing pattern. The asynchronization of the routers is simulated by assigning random deviation of the decision time to each router. The deviation is created uniformly random over $[0, \Delta_R]$ minutes for each router.

On top of the simulation system, distributed routing (DIS) and Algorithm 1 are implemented as well as the three kernel algorithms: global optimization (GLO), greedy (GRE), and local search (LOC). Dijkstra algorithm is programmed to solve shortest path problems, which is a critical component not only for DIS, but also for GRE and LOC. In order to deal with the $C_\pi(t)$ in GLO, we express it as an integer program with the decision variables $x_e^n(t)$, and utilize COIN-OR Branch and Cut (CBC) [35] as the integer program solver.

The $C_\pi(t)$ in GLO is not always feasible. When $C_\pi(t)$ is not feasible, the controller can only guarantee the sending rate for a subset of the flows, and hence the controller would have to drop some traffic or reduce the sending rate. As to how to deal with infeasibility is a design issue and beyond the scope of this research. Therefore, we simply invoke GRE to select a subset of the flows that admits a routable solution, and route those flows by GLO when $C_\pi(t)$ is infeasible.

We assign each flow an id number and GRE updates according to the order of id number. The flow with the smaller id number would be routed earlier than the flows with larger id numbers. The id numbers are also used in LOC when adding flows into the circular buffer.

In LOC, we adopt the simplest controller heuristic to obtain a routable routing pattern, which does nothing but putting back the feasible flows in the current routing pattern.

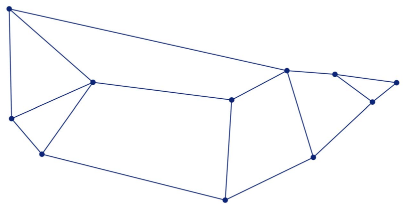


Fig. 8. The Internet2 Network topology.

B. Statistical Performance

We first compare the performance of the kernels based on the layer-3 topology of the Internet2 Network [36], which has 11 nodes and 17 edges (Fig. 8). Each edge has 100 Gbps bidirectionally, and its metric is set proportional to the distance between its endpoints.

Two priority classes are allowed, and among all the possible flows $< s^n, d^n, \pi^n >$, we include each with probability 0.8 to form the set of flows and index them by N. For each selected flow, we assign an initial sending rate and a target sending rate uniformly random from the interval $[5\gamma, 15\gamma]$ Gbps, where γ is the *tightness parameter* to control the size of the generated flow. The initial stable routing pattern is achieved by DIS. As all the flows switch to the target sending rate, our methods are applied to stabilize the network. Under 1000 independent simulations and different γ, we compare cost performance, computational efficiency, and route disturbance among the kernel algorithms.

1) Cost Performance: The first issue is the cost performance. As demonstrated in Section II, a stable routing pattern does not necessarily imply the shortest aggregated path metric. That is also the reason why the intervention of the centralized controller may improve the efficiency of the bandwidth utilization. To capture the phenomenon, we define the path metric ratio of a flow to be the metric of the path assigned to the flow divided by the metric of the shortest possible path in the network when all the resources are available for the flow. Let $p^n(t)$ be the shortest possible path, which is independent of the other traffic, we have

$$\text{path metric ratio of } F^n \text{ at time } t = \frac{m(x^n(t))}{m(p^n(t))}.$$

According to the definition, the path metric ratio is always greater than or equal to one. We compare the average of the path metric ratio of those routed flows, i.e., excluding the flows that cannot be routed. If a kernel gives the average path metric ratio close to one, the kernel can route most of the flows through the shortest possible path, which suggests a good cost performance.

In Fig. 9, LOC performs similar to GRE under small γ. When the network becomes highly utilized, LOC beats GRE. GLO outperforms the other two kernels when γ is small, as we would expect. As γ growing larger, not all the flows can be routed, and GRE is introduced to select the routed flows, which causes the performance downgrade of GLO. While LOC still performs well in that case.

2) Computational Efficiency: Even though GLO performs the best when the flows can all be routed, it achieves the performance with a huge computational effort. The computational effort is evaluated through the real CPU time. Such computational requirement is not reflected directly in all the simulation results. The reason behind the demonstration decision is that the computational time depends on the deployed hardware. The commercial routers or controllers can normally compute much faster than our simulation platform as those are highly specialized equipment. But it is worth taking a closer look at the computational effort spent in our simulation as it may shed light on the computational requirement in the real setting.

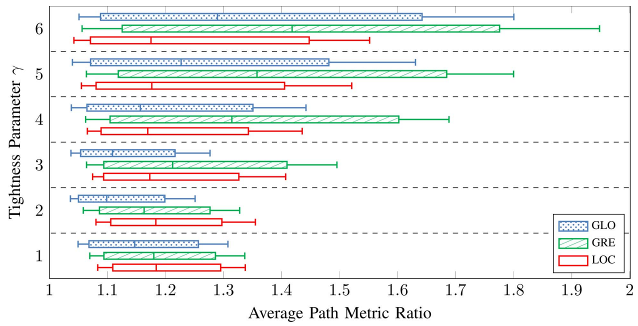


Fig. 9. The 1^{st}-5^{th}-50^{th}-95^{th}-99^{th} percentiles of the average path metric ratio of the routed flows under different tightness parameter γ. In general, GLO gives the lowest average metric ratio.

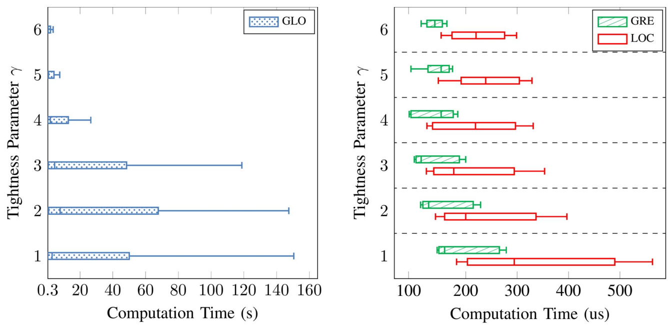


Fig. 10. The 1^{st}-5^{th}-50^{th}-95^{th}-99^{th} percentiles of the computation time. GRE is faster than LOC, and GLO is significantly slower than the other two.

In Fig. 10, GLO requires up to 6 orders of magnitude longer time than the other two kernels. GRE consumes the least time, and the computation time decreases as γ increases, since the number of the flows that can be routed is decreasing. The results also show that the computation time of LOC is merely around twice the time needed by GRE in practice, although it is of quadratic time complexity in theory.

3) Route Disturbance: Besides the cost performance and computational efficiency, the operator might want to stabilize the network with minimum number of routes changed, since route switching is one of the main packet lost reasons.

In Fig. 11, GLO and GRE both change around 1.5 to 2 times more routes than LOC does when stabilizing the network. The reason is that the heuristic in LOC applies the current routing pattern as the initial routable pattern, and LOC will terminate once a stable pattern is found by changing one path at a time

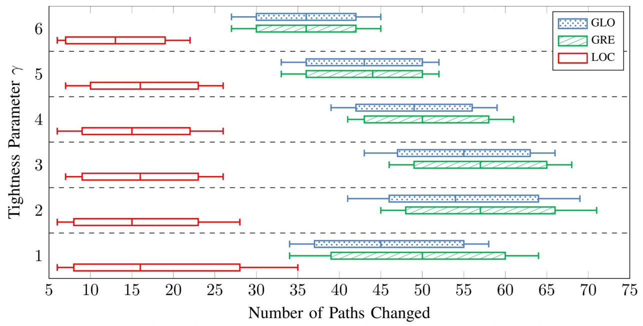


Fig. 11. The 1^{st}-5^{th}-50^{th}-95^{th}-99^{th} percentiles of the number of paths changed. LOC changes least paths to stabilize the network.

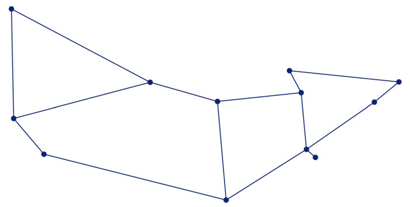


Fig. 12. The Abilene Network topology.

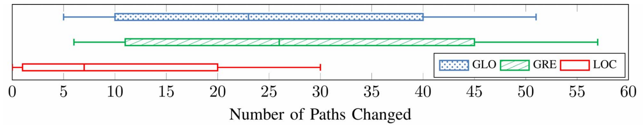


Fig. 13. The 1^{st}-5^{th}-50^{th}-95^{th}-99^{th} percentiles of the number of paths changed. LOC changes least paths to stabilize the network.

to search locally, which usually changes only small fraction of the routes.

We also simulate the Abilene network, the ancestor of the Internet2 Network, using the trace from [37]. The trace provides both the measured network topology and traffic matrix. Fig. 12 shows the topology, which is slightly different from Internet2 (Fig. 8).

The traffic matrix consists of aggregated traffic per source/destination pair. We consider two priority classes – high and low – and partition the aggregated traffic randomly into the priority classes to create the flows.

As in Section VII-B, we first obtain a stable routing pattern by DIS. We then vary the sending rate of each flow by a random per-flow factor, uniformly distributed between 0.8 and 1.2. Over 1000 independent simulations, we collect the results and discuss their statistics in two aspects: the number of routes flapped to stabilize the routing pattern and the average path metric ratio in the achieved pattern.

Fig. 13 shows the number of paths changed to obtain a stable routing pattern. LOC finds a stable routing pattern with the least paths changed. The average path metric ratio, however, is the best under GLO as in Fig. 14, and sophisticated centralized control (GLO and LOC) can perform better than DIS. The trends are consistent with Fig. 11 and 9.

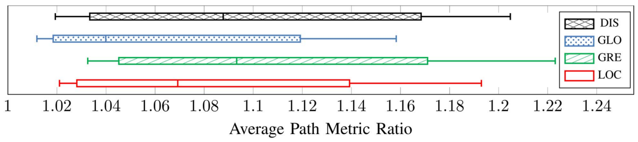


Fig. 14. The 1^{st}-5^{th}-50^{th}-95^{th}-99^{th} percentiles of the average path metric ratio of the routed flows. GLO gives lowest cost among the three kernel algorithms. Sophisticated centralized control (GLO and LOC) can perform better than distributed control (DIS).

C. Large-Scale Data-Driven Tests

To further evaluate the practical efficiency of the algorithms, we establish a modified but realistic wide-area network (WAN) topology based on a tier-1 ISP's backbone network. The WAN has 92 nodes and 418 edges. The capacity, ranging from 100 to 5000 Gbps, and the metric of the edges are set based on the measurements of the capacity and the OSPF cost of the corresponding IP links.

The sending rate used in the simulations comes from the 5-minute measurements of the real traffic data on the WAN. The 5-minute measurements are obtained by first partitioning the whole day into 288 disjoint 5 minute periods and taking the average of the per-priority-class aggregated traffic at the edge routers within each period. As defined in Section III, we deem all the traffic with the same source, destination, and priority class to be the same flow. As such, if two network sessions of different purposes, such as web searching and video streaming, are sending data through the same path with the same priority, we don't differentiate them and treat them as the same flow. A total of 3547 different flows from two priority classes are identified, and each flow updates its sending rate every 5 minutes. The sending rate is highly diversified among the flows: It distributes from few hundred Kbps to several hundred Gbps.

1) Routing Stability: Firstly, we inspect how well our methods can stabilize the routes. Four different scenarios are considered. In the distributed scenario, each flow is routed through the shortest path at its source router. The other three scenarios correspond to Algorithm 1 with the kernel algorithms GLO, GRE, and LOC, respectively.

Under these four scenarios, we turn on the system at time 0 and count the number of routes changed for 120 minutes. We reset the count every 12 seconds and plot them in a set of figures. Since the system is mostly stabilized within 5 minutes under all four scenarios after the system starts, the change of the routes after the fifth minute results from the traffic variation. As to how well the stabilizing algorithms can deal with traffic variation, we plot in Fig. 15 the number of changed path from 5 to 120 minutes after the system starts.

Starting with a stable routing pattern, DIS still needs several cycles before reaching a stable routing pattern as in Fig. 15a. All proposed kernels stabilize the routing pattern rapidly, while the numbers of changed paths are slightly different. We will discuss the difference in more details in Section VII-C.4.

2) Supported Throughput: In addition to stability, the supported throughput, i.e., how many flows can be routed at each moment, is also an important aspect to consider for

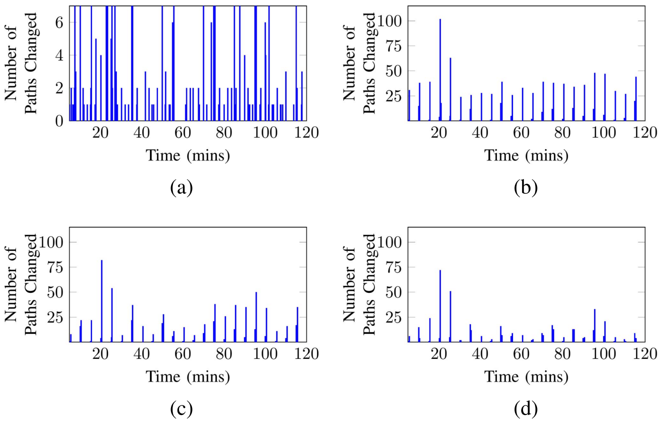


Fig. 15. Number of paths changed during 5−120 mins after the system starts. All three kernel algorithms can effectively stabilize the network. Among them, LOC changes the least paths. (a) Distributed (DIS). (b) Global optimization (GLO). (c) Greedy (GRE). (d) Local search (LOC).

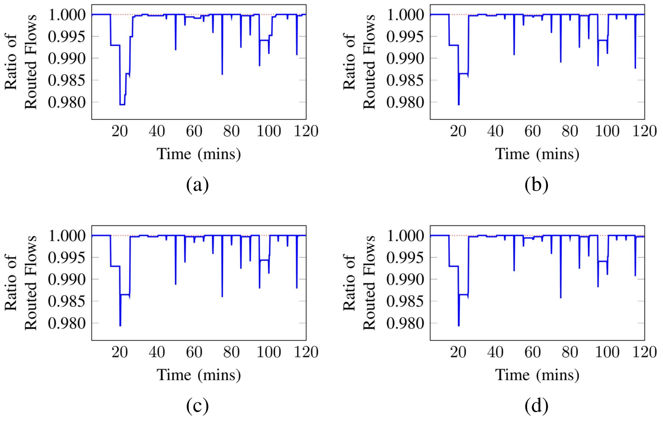


Fig. 16. Ratio of routed flows. The kernel algorithms can shorten the convergence time to start serving more flows earlier. (a) Distributed (DIS). (b) Global optimization (GLO). (c) Greedy (GRE). (d) Local search (LOC).

the network operator. We keep track of the number of flows successfully routed to their destination, normalize it by the total number of non-zero flows, and plot it in Fig. 16 under the four different scenarios. We can observe that local routers (Fig. 16a) mostly support fewer flows than those cases with controller intervention (Fig. 16b-16d). In fact, for more than 65% of the time, the three controller involved cases can route all the flows, but the local routers are never able to do so within the 120 minutes. That confirms the discussion in Section II: The network utilization can be improved by introducing a centralized controller.

3) Computational Efficiency: We also compare the computation time needed for each controller in Fig. 17. GLO requires considerable computation time (Fig. 17a). Although not shown in the figure, it may take more than thousand seconds (1754.74 seconds in this case) to route the traffic, which weakens GLO's ability to deal with the quick variation of the sending rate. GRE, as expected, takes almost constant time to route the traffic (Fig. 17b), while LOC needs to check and reroute the traffic several rounds before reaching a stable routing pattern, which is reflected by the peaks after each sending rate change moment (Fig. 17c).

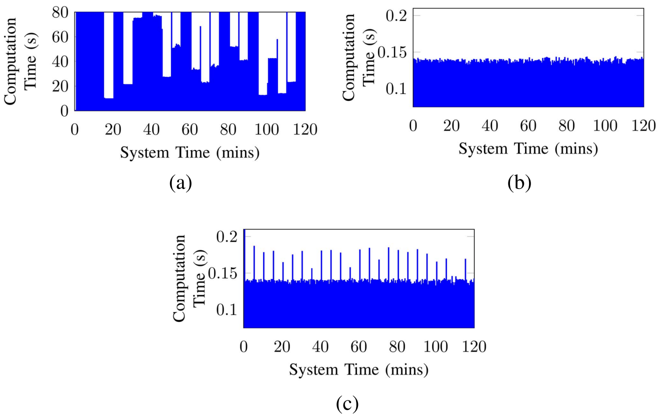


Fig. 17. Computation time of the controller. GLO takes too much time to compute a stable routing pattern, which confines its application in practice. (a) Global optimization (GLO). (b) Greedy (GRE). (c) Local search (LOC).

TABLE III

TOTAL NUMBER OF PATHS CHANGED

γ	GLO	GRE	LOC
1	883	556	411
1.2	1814	1249	695
1.4	2410	2299	985
2	computationally intractable	6221	2659
4		16838	7886
6		24609	11686
8		30791	13253

TABLE IV

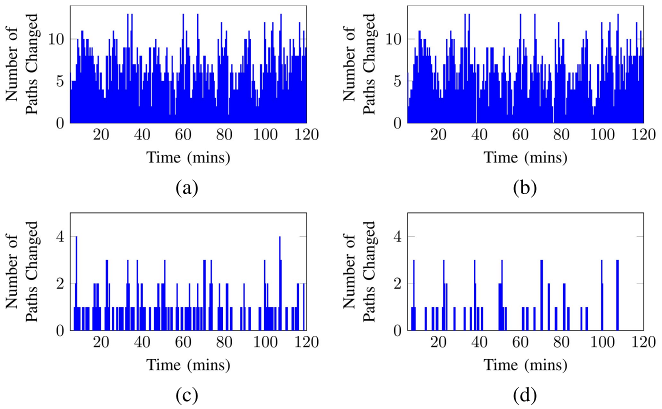
AVERAGE RATIO OF ROUTED FLOWS

γ	GLO	GRE	LOC
1	0.998478	0.998391	0.998417
1.2	0.997598	0.997647	0.997681
1.4	0.996485	0.99656	0.996301
2	computationally intractable	0.981571	0.987322
4		0.900165	0.932775
6		0.802259	0.878906
8		0.685143	0.843711

4) Load Management: Applying the tightness parameter γ as the multiplier to the traffic sending rate, we scrutinize the performance of the algorithms under different network loading within the time period from 5 to 120 minutes.

In Table III, GLO and GRE change up to 2 times more routes than LOC. The reasons are as follows. The stable routing pattern with the lowest cost is not necessarily taking the similar routes as the current routing pattern. The routes can be totally different and hence several routes will be changed. GRE ignores the current routing pattern and fits in the flows one by one, which can cause cascade route flapping when a new route claims the bandwidth from the old routes. LOC changes the least paths to obtain a stable routing pattern, which is consistent with the result in Section VII-B.3.

Table IV shows that LOC outperforms GRE. Besides the lightest loading condition – under which all the kernels can



(a)

(b)

(c)

(d)

Fig. 18. Number of paths changed under partial information. Information recovery helps mitigate path flapping caused by partial information. (a) $\rho_m = 10^{-3}$ without information recovery. (b) $\rho_m = 10^{-3}$ with information recovery. (c) $\rho_m = 10^{-4}$ without information recovery. (d) $\rho_m = 10^{-4}$ with information recovery.

route 99.5% of the flows – LOC can route up to 10% more flows than GRE under those heavier loading conditions.

5) Partial Information: To realize how well the information recovery phase in our design can help alleviate the impact of information missing, we let the PCEP information be missing with *information missing rate* ρ_m and run LOC to stabilize the network. We fix the traffic sending rate as the sending rate at time 0 (min) and observe the network starting from the fifth minute, when the network is supposed to be stable.

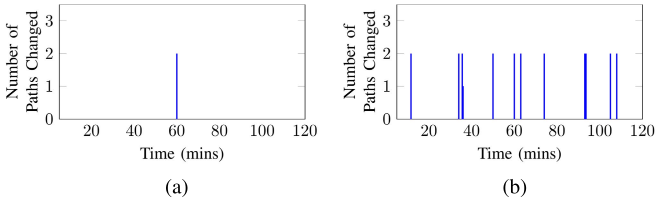
When the information missing rate is large (10^{-3}), the information recovery phase is helpful but not significant (Fig. 18a and 18b). However, when the information missing rate is small, which is usually the case in a commercial network, the information recovery phase can improve the stability by reducing the frequency and the number of route flapping (Fig. 18c and 18d).

6) Inconsistent Information: We also consider the issue of inconsistent information. Similar to the partial information simulation, we fix the sending rate and make each router send a PCEP message which is inconsistent with the BGP-LS information with *information inconsistent rate* ρ_i. The inconsistent PCEP message reports the sending rate only 95% of the true rate. Again, we run LOC to stabilize the network.

The simulation result is a landslide: In all simulated cases with the information recovery phase to obtain the effective capacity, the routing pattern remains the same and no path is changed. However, those without information recovery change paths, and the network is less stable as ρ_i increases.

D. Failover Mechanism

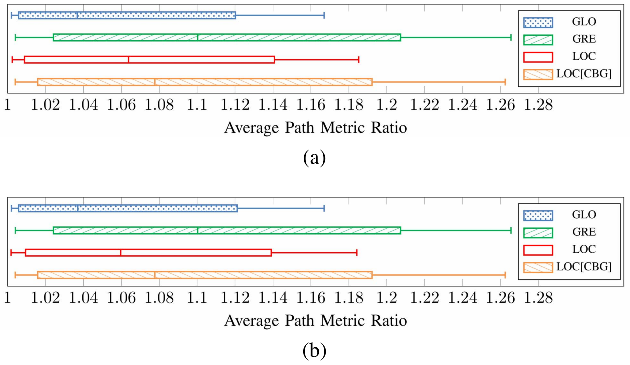
The legacy distributed routing usually has some failover mechanism that can reroute the packets swiftly once the original route fails, e.g., [38] lists few such local fast failover mechanisms. We compare two different failover mechanisms via simulations: fast source reroute by CSPF and depth first search (DFS) based local fast failover.



(a)

(b)

Fig. 19. Number of paths changed under inconsistent information. No path is changed if information recovery phase is adopted. (a) $\rho_i = 10^{-3}$ without information recovery. (b) $\rho_i = 10^{-2}$ without information recovery.



(a)

(b)

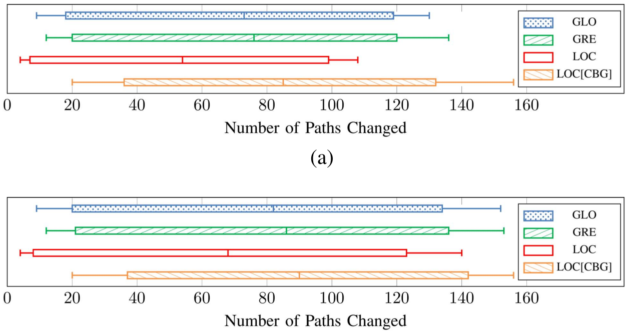
Fig. 20. The 1^{st}-5^{th}-50^{th}-95^{th}-99^{th} percentiles of the average path metric ratio of the routed flows. The kernel algorithms can lead to similar average path metric ratio under both failover mechanism. (a) Fast source reroute by CSPF. (b) Local fast failover using DFS.

Fast source reroute relies on the source router to detect and reroute the traffic once the original route does not work anymore. Once the route failure is detected by the source router, it solves the CSPF $R^n(t)$ to obtain a new shortest route and deploys the new route. On the other hand, the DFS based local fast failover is performed by the router where the failure is sensed. As soon as a router finds that an outport is no longer available due to failure or congestion, the router redirect the traffic to new paths discovered by DFS.

To evaluate how the failover mechanisms affect our design, we generate 1000 random stable routing patterns using Abilene network and trace, cut one random edge per pattern, and stabilize the pattern after the failover mechanism takes place.

Fig. 20 show that the failover mechanisms have little impact on the average metric ratio achieved by the stabilization processes. In other words, the methods reach similar stable states regardless of the underlying failover mechanisms.

However, the "distance" between the post-failover pattern and the reached stable pattern can be different. Fig. 21 shows the number of modified paths to reach the stable patterns. The local source reroute leads to a routing pattern closer to a stable one than the pattern resulted from DFS based local fast failover. It is as expected, as the local source reroute tries to reroute the traffic through available shortest paths, but the DFS simply finds a feasible path that is not necessarily the shortest. As a result, more paths are adjusted to reach a stable pattern under DFS based local failover.



(a)

(b)

Fig. 21. The 1^{st}-5^{th}-50^{th}-95^{th}-99^{th} percentiles of the number of paths changed. As DFS usually doesn't discover the shortest path, more paths are changed to reach a stable routing pattern. (a) Fast source reroute by CSPF. (b) Local fast failover using DFS.

VIII. Related Work

To the best of our knowledge, our work is the first one to address the stability issue in hybrid SDN. We briefly discuss below the existing related work.

Hybrid SDN: Hybird SDN combines the "efficient global routing" benefit of centralized routing with scalability, robustness and immediate decision benefits of distributed routing [12]. Reference [13] categorizes several hybrid SDN designs into four categories: topology-based, service-based, class-based, and integrated hybrid SDN. In the first three categories, SDN and the legacy routing control disjoint sets of traffic (or forwarding information bases). Therefore our work should belong to a special kind of integrated hybrid SDN, in which the traffic is controlled by both control units. IBSDN [22] is an idea similar to the hybrid SDN in this work. The main difference is our hybrid SDN does not require the routers to maintain a primary state and a backup state. Fibbing [23] is also a hybrid framework. The centralized controller utilizes distributed routing to direct the traffic by creating augmented topology including fake nodes and links for the local routers. In our case, the centralized controller does not modify the local routers' views, but try to be consistent with them.

There is also an another kind of hybrid SDN in the literature that considers the mix use of SDN and legacy switches, such as Panopticon [39] and SHEAR [40]. In both work, the centralized controller controls only the SDN switches in the network to collaborate with the legacy switches. In our work, however, the centralized controller can control all the switches, and each switch also runs legacy distributed routing that is independent of the centralized controller. Our research aims to make these two independent control mechanisms coherent.

Consistent SDN Update: There exists a body of work on consistent network update which also involves stability issues. We refer the reader to [41] for a comprehensive survey. The key difference between their work and ours is in terms of control granularity, with consistent SDN update concerning about packet level consistency while ours primarily dealing with flow level control coherence.

Stability: The network stability issue has been discovered for a long time [42] and scrutinized for distributed routing mechanisms, including OSPF [14], BGP [19], [43], [44], and some other AS-based path-vector protocols [15], [16], [18], [45]. In particular, a stable state is defined as the state when the centralized controller and the local routers reach a consensus [15], i.e., the centralized controller's decision is the best possible one for the local routers as in [16]. Another similar concept is the "persistence" in [45], which is how long a route can stay unchanged.

IX. Conclusion

We define and study dual control stability in hybrid software-defined networks where distributed and centralized routing coexist. To avoid potential route flapping as two control units alternatively take charge of routing, the centralized controller has to ensure its decision to be consistent with the local routers' decisions. This observation then leads to the development of an algorithmic framework with three different stabilization kernels: global optimization, greedy, and local search. We design and implement a system on a centralized controller which utilizes those kernels to stabilize the network. Through extensive simulations using realistic network information including data from a tier-1 ISP's backbone network, our conclusion is that the proposed local search provides the best trade-off among computational complexity, cost performance and route disturbance.

References

[1] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks," in *Proc. USENIX NSDI*, 2010, pp. 89–92.

[2] V. Mann, A. Vishnoi, K. Kannan, and S. Kalyanaraman, "CrossRoads: Seamless VM mobility across data centers through software defined networking," in *Proc. IEEE/IFIP NOMS*, Apr. 2012, pp. 88–96.

[3] A. Iyer, P. Kumar, and V. Mann, "Avalanche: Data center multicast using software defined networking," in *Proc. IEEE COMSNETS*, Jan. 2014, pp. 1–8.

[4] C.-Y. Hong *et al.*, "Achieving high utilization with software-driven WAN," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 15–26, 2013.

[5] S. Jain *et al.*, "B4: Experience with a globally-deployed software defined WAN," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 3–14, 2013.

[6] C. J. S. Decusatis, A. Carranza, and C. M. DeCusatis, "Communication within clouds: Open standards and proprietary protocols for data center networking," *IEEE Commun. Mag.*, vol. 50, no. 9, pp. 26–33, Sep. 2012.

[7] M. F. Bari *et al.*, "Data center network virtualization: A survey," *IEEE Commun. Surveys Tuts.*, vol. 15, no. 2, pp. 909–928, 2nd Quart., 2013.

[8] R. Jain and S. Paul, "Network virtualization and software defined networking for cloud computing: A survey," *IEEE Commun. Mag.*, vol. 51, no. 11, pp. 24–31, Nov. 2013.

[9] D. Kreutz, F. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proc. IEEE*, vol. 103, no. 1, pp. 14–76, Jan. 2015.

[10] M. Yu, J. Rexford, M. J. Freedman, and J. Wang, "Scalable flow-based networking with DIFANE," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 4, pp. 351–362, Oct. 2010.

[11] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "DevoFlow: Scaling flow management for high-performance networks," *Comput. Commun. Rev.*, vol. 41, no. 4, pp. 254–265, Aug. 2011.

[12] S. Sezer *et al.*, "Are we ready for SDN? Implementation challenges for software-defined networks," *IEEE Commun. Mag.*, vol. 51, no. 7, pp. 36–43, Jul. 2013.

[13] S. Vissicchio, L. Vanbever, and O. Bonaventure, "Opportunities and research challenges of hybrid software defined networks," *ACM Comput. Commun. Rev.*, vol. 44, no. 2, pp. 70–75, Apr. 2014.

[14] A. Basu and J. Riecke, "Stability issues in OSPF routing," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 31, no. 4, pp. 225–236, 2001.

[15] L. Gao and J. Rexford, "Stable Internet routing without global coordination," *IEEE/ACM Trans. Netw.*, vol. 9, no. 6, pp. 681–692, Dec. 2001.

[16] T. G. Griffin, F. B. Shepherd, and G. Wilfong, "The stable paths problem and interdomain routing," *IEEE/ACM Trans. Netw.*, vol. 10, no. 2, pp. 232–243, Apr. 2002.

[17] F. Kelly and T. Voice, "Stability of end-to-end algorithms for joint routing and rate control," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 35, no. 2, pp. 5–12, 2005.

[18] D. Papadimitriou, F. Coras, and A. Cabellos, "Path-vector routing stability analysis," *ACM SIGMETRICS Perform. Eval. Rev.-Meas. Eval.*, vol. 39, no. 3, pp. 22–24, 2011.

[19] J. Rexford, J. Wang, Z. Xiao, and Y. Zhang, "BGP routing stability of popular destinations," in *Proc. ACM IMW*, 2002, pp. 197–202.

[20] M. Birk *et al.*, "Evolving to an SDN-enabled ISP backbone: Key technologies and applications," *IEEE Commun. Mag.*, vol. 54, no. 10, pp. 129–135, Oct. 2016.

[21] G. Choudhury *et al.*, "Centralized optimization of traffic engineering tunnels in a large ISP backbone using an SDN controller," in *Proc. INFORMS Optim. Soc. Conf.*, Mar. 2016.

[22] O. Tilmans and S. Vissicchio, "IGP-as-a-backup for robust SDN networks," in *Proc. IEEE CNSM*, Nov. 2014, pp. 127–135.

[23] S. Vissicchio, O. Tilmans, L. Vanbever, and J. Rexford, "Central control over distributed routing," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, pp. 43–56, 2015.

[24] D. P. Williamson and D. B. Shmoys, *The Design Approximation Algorithms*. Cambridge, U.K.: Cambridge Univ. Press, 2011.

[25] *AT&T Class of Service*. Accessed: Feb. 26, 2019. [Online]. Available: http://carecentral.att.com/downloads/Class of Service.pdf

[26] J. Vasseur and J. Le Roux, *Path Computation Element (PCE) Communication Protocol (PCEP)*, document RFC 5440, 2009.

[27] H. Gredler, J. Medved, S. Previdi, A. Farrel, and S. Ray, *North-Bound Distribution of Link-State and Traffic Engineering (TE) Information Using BGP*, document RFC 7752, 2016.

[28] M. Belaidouni and W. Ben-Ameur, "On the minimum cost multiple-source unsplittable flow problem," *RAIRO-Oper. Res.*, vol. 41, no. 3, pp. 253–273, 2007.

[29] J. M. Kleinberg, "Approximation algorithms for disjoint paths problems," Ph.D. dissertation, Dept. Elect. Eng. Comput. Sci., Massachusetts Inst. Technol., Cambridge, MA, USA, 1996.

[30] P. Kolman and C. Scheideler, "Improved bounds for the unsplittable flow problem," in *Proc. SODA*, 2002, pp. 184–193.

[31] P. Kolman, "A note on the greedy algorithm for the unsplittable flow problem," *Inf. Process. Lett.*, vol. 88, no. 3, pp. 101–105, 2003.

[32] Y. Azar and O. Regev, "Combinatorial algorithms for the unsplittable flow problem," *Algorithmica*, vol. 44, no. 1, pp. 49–66, 2006.

[33] A. Chakrabarti, C. Chekuri, A. Gupta, and A. Kumar, "Approximation algorithms for the unsplittable flow problem," *Algorithmica*, vol. 47, no. 1, pp. 53–78, 2007.

[34] *NERC Reliability Concepts*. Accessed: Feb. 26, 2019. [Online]. Available: http://www.nerc.com/files/concepts v1.0.2.pdf

[35] *CBC (COIN-OR Branch and Cut)*. Accessed: Feb. 26, 2019. [Online]. Available: https://projects.coin-or.org/Cbc

[36] *The Internet2 Network*. Accessed: Feb. 26, 2019. [Online]. Available: http://www.internet2.edu/

[37] S. Orlowski, M. Pióro, A. Tomaszewski, and R. Wessäly, "SNDlib 1.0—Survivable network design library," in *Proc. INOC*, 2007, pp. 1–16. [Online]. Available: http://sndlib.zib.de

[38] M. Borokhovich, L. Schiff, and S. Schmid, "Provable data plane connectivity with local fast failover: Introducing OpenFlow graph algorithms," in *Proc. ACM SIGCOMM HotSDN Workshop*, 2014, pp. 121–126.

[39] D. Levin *et al.*, "Panopticon: Reaping the benefits of incremental SDN deployment in enterprise networks," in *Proc. USENIX Annu. Tech. Conf.*, 2014, pp. 333–345.

[40] M. Markovitch and S. Schmid, "SHEAR: A highly available and flexible network architecture marrying distributed and logically centralized control planes," in *Proc. IEEE ICNP*, Nov. 2015, pp. 78–89.

[41] K.-T. Foerster, S. Schmid, and S. Vissicchio. (2016). "Survey of consistent software-defined network updates." [Online]. Available: https://arxiv.org/abs/1609.02305

[42] C. Labovitz, G. R. Malan, and F. Jahanian, "Internet routing instability," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 27, no. 4, pp. 115–126, Oct. 1997. [Online]. Available: http://doi.acm.org/10.1145/263109.263151

[43] R. Govindan and A. Reddy, "An analysis of Internet inter-domain topology and route stability," in *Proc. INFOCOM*, vol. 2, Apr. 1997, pp. 850–857.

[44] K. Varadhan, R. Govindan, and D. Estrin, "Persistent route oscillations in inter-domain routing," *Comput. Netw.*, vol. 32, no. 1, pp. 1–16, 2000.

[45] V. Paxson, "End-to-end routing behavior in the Internet," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 26, no. 4, pp. 25–38, 1996.

Shih-Hao Tseng received the Ph.D. degree in electrical and computer engineering from Cornell University in 2018. He is currently a Post-Doctoral Scholar with the California Institute of Technology. His research interests include software-defined networking, network function virtualization, network optimization, algorithm, and system design.

Ao Tang (S'01–M'07–SM'11) received the B.E. degree in electronics engineering from Tsinghua University, Beijing, China, in 1999, and the M.S. and Ph.D. degrees in electrical engineering with a minor in applied and computational mathematics from the California Institute of Technology, Pasadena, CA, USA, in 2002 and 2006, respectively. He is currently an Associate Professor with the School of Electrical and Computer Engineering, Cornell University, Ithaca, NY, USA, where he is involved in control and optimization of computer networks with a particular focus on autonomous network management.

Gagan L. Choudhury (F'09) received the Ph.D. degree in electrical engineering from the State University of New York at Stony Brook in 1982. He is currently a Lead Inventive Scientist with AT&T Labs, Middletown, NJ, USA. He has over 100 technical publications and around 30 granted patents. His research interests are in the optimization, analysis, and design of software defined networks and mobility/5G networks. He became an AT&T Fellow for "outstanding contributions to performance analysis and robust design and their application to improving the performance, reliability and scalability of AT&T's networks" in 2009.

Simon Tse received the B.S. degree in engineering from Brown University, the M.S. and Ph.D. degrees in applied sciences from Harvard University, and the M.B.A. degree from the Wharton School, University of Pennsylvania. He began his career with AT&T Bell Laboratories, Middletown, NJ, USA, in 1985, where he is currently the Director of Inventive Science. He manages a group of technical professionals in network topology designs, network traffic management, and software defined network controllers for multilayer network resource optimization.