Routing Stability in Hybrid Software-Defined Networks

Shih-Hao Tseng, Ao Tang, Gagan L. Choudhury, 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 trade-off 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 data from a tier-1 ISP's 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 also discuss how to ensure routing stability under imperfect information. Our preferred approach to dealing with imperfect information is for the centralized controller to wait until information becomes perfect. This works since the imperfect information scenario is usually rare and non-persistent, and the robust distributed routing functions all the time. A second approach is to decouple it to two phases: the information recovery phase and the stability pursuit phase (§V). These two phases can be handled independently. For the first phase, we explore how to leverage the existing redundancy to restore missing information. As such, the restored information can be used in the second phase as perfect information. The redundancy provides robustness, but inconsistent redundancy can be confusing. We resolve the confusion by computing the effective capacity, which estimates the minimum possible capacity.

We then describe in more detail how a centralized controller in a hybrid SDN can be built to achieve stable routing patterns (§VI). The design is implemented through a simulation platform based on modified but realistic wide area network topologies (§VII-B and §VII-C) and traffic data from a tier-

S.-H. Tseng and A. Tang are with the School of Electrical and Computer Engineering, Cornell University, Ithaca, NY 14853, USA (e-mails: {st688, at422}@cornell.edu).

G. L. Choudhury and S. Tse are with AT&T Labs, 200 Laurel Avenue South Middletown, NJ 07748, USA (e-mails: {gc2541, st2196}@att.com).

1 ISP (§VII-C). We utilize the centralized controller with different kernel algorithms to stabilize the network. Through extensive simulations, we demonstrate that the global optimization kernel performs the best in a lightly loaded network. When the loading becomes heavier, the global optimization kernel hinges on the routed flow selection and hence it can suffer performance downgrade. The greedy kernel takes the least time to generate a stable routing pattern, but its routing performance tends to be poor. The local search kernel needs less than twice the time of the greedy kernel in practice, flaps half number of the routes, routes 10% more traffic than the other two kernels when the network loading is heavy, and hence provides the best trade-off among computational complexity, cost performance and route disturbance.

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 the algorithms to achieve stability. Partial 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.

II. Background

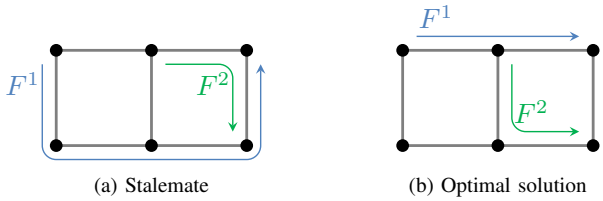
Routing is an essential functionality that a network needs to provide in order for users to send their traffic through, and one key property of a desirable routing mechanism is stability. A stable routing mechanism should not keep changing the route decisions if all the inputs remain the same. Otherwise,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

Nowadays, distributed routing protocols such as Open Shortest Path First (OSPF) and Border Gateway Protocol (BGP) dominate the routing of the network traffic. 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).

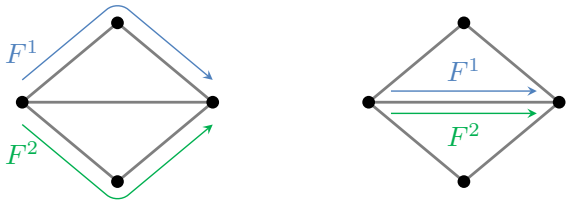
The simple example above explains the motivation for the network operator to introduce a coordinator with global view to help resolve the stalemate and improve the utilization of the network. SDN is a perfect framework to deploy such a coordinator – using the SDN centralized controller.

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.



(a) Stalemate (b) Optimal solution

Fig. 1. A simple example showing that distributed routing can end up in stalemate.



(a) Routing pattern preferred by the centralized controller (b) Routing pattern preferred by the local routers

Fig. 2. A simple example showing that the inconsistency between the centralized controller and the local routers can cause instability.

This dual control framework has significant benefits over pure distributed or pure centralized routing. [20], [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 when failure occurs.

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, the minimum feasible design avoids the need of upgrading the 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.

[1]We refer the reader to Figure 2 in [20] 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.

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 imperfect 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 ways of addressing imperfect information situations in other ways as well.

III. Formulation

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. To ensure $x^n(t)$ forms a path, an additional condition $x^n(t) \in \mathcal{P}^n$ is introduced.[3]

A priority class π^n is associated with each flow F^n, and in this work the 3-tuple $< s^n, d^n, \pi^n >$ uniquely defines a flow. We say $\pi^1 \leq \pi^2$ 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 $N_{\leq\pi} = \{n \in N : \pi^n \leq \pi\}$. Similar definitions apply to $N_{=\pi}$. We denote by $\Pi = \{\pi^n : n \in N\}$ the set of all priority classes.

Table I summarizes the major notations for easier notation lookup.

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:

$$
\begin{aligned}
R^n(t) = \min \quad & \sum_{e \in E} m_e x_e^n(t) \\
\text{s.t.} \quad & x^n(t) \in \mathcal{P}^n && \text{(1a)} \\
& x_e^n(t) \in \{0,1\} \qquad \forall e \in E && \text{(1b)} \\
& x_e^n(t) \leq z_e(t) \qquad \forall e \in E && \text{(1c)} \\
& \sum_{n' \in N_{\leq \pi^n}} r^{n'}(t) x_e^{n'}(t) \leq c_e \quad \forall e \in E && \text{(1d)}
\end{aligned}
$$

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. The objective function $\sum_{e \in E} m_e x_e^n(t)$ gives the metric of the selected path, which is the OSPF cost of the path if m_e is the OSPF cost of the edge e. 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.

TABLE I
Major Notations.

for each $e \in E$

c_e	edge capacity
m_e	edge cost metric
$z_e(t)$	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
$x^n(t) \in \mathcal{P}^n$	condition that $x_e^n(t)$ forms a path

for priority class $\pi \in \Pi$

$N_{\leq\pi}$	indices of the higher prioritized flows
$N_{=\pi}$	indices of flows with priority π

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 [25]): 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 [26]): 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 $\sum_{n \in N_{=\pi}} r^n(t) x_e^n(t)$.

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

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

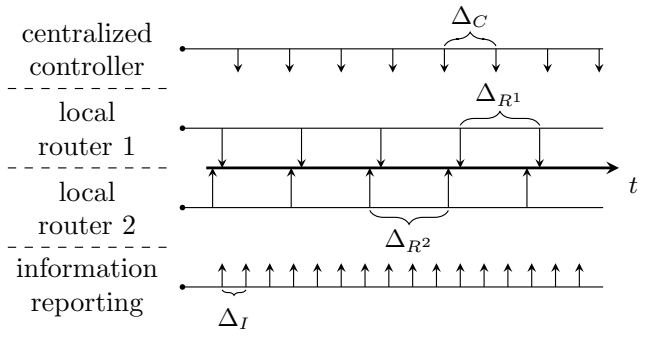


Fig. 3. The system is not necessarily synchronized. Each control unit/reporting protocol has its update interval.

Local routers report the information through these two protocols on a regular basis, but the reporting time is not 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 make an assumption about the behaviour 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: Stable Routing Framework

1: **for** From the highest priority $\pi \in \Pi$ **to** the lowest **do**
2: Invoke the kernel algorithm to get a stable routing pattern of the flows indexed by $N_{=\pi}$ and deploy the corresponding routes.
3: **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], [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. As a result, 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. In the following subsections, we propose different kernel algorithms to obtain a stable routing pattern for a priority class. We defer the stability proofs of Algorithm 1 and the proposed kernels to Appendix.

A. Global Optimization Algorithm

One way to obtain a stable routing pattern for a priority class is by solving the global optimization problem $C_\pi(t)$, given by

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

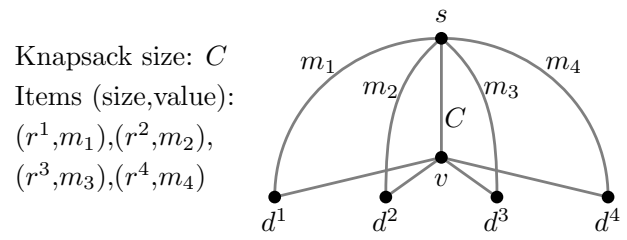


Fig. 4. Polynomial-time reduction from the Knapsack problem.

The global optimization kernel routes the flows based on the resulted $x_e^n(t)$, and its stability is shown in Appendix A.

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, Proposition 1 suggests that obtaining an optimal solution to $C_\pi(t)$ is computationally intractable because of NP-hardness. Therefore, we propose two other tractable methods in the following subsections.

Proposition 1. *Solving $C_\pi(t)$ is NP-hard.*

Proof. We show the proposition by providing a polynomial-time reduction from the Knapsack problem, which is NP-hard.

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 setup $C_\pi(t)$ by letting Π be a singleton and creating 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, solving $C_\pi(t)$ 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. 4 shows the construction in the proof above.

B. Greedy Algorithm

While NP-hardness prevents us from solving the global optimization problem, 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 Algorithm 2. Again, the stability of the kernel is shown in Appendix A.

Algorithm 2 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, which is beyond the scope of this paper.

Algorithm 2: Greedy Algorithm

1: **for** Choose $n \in N_{=\pi}$ in an arbitrary order **do**
2: Solve $R^n(t)$ and deploy the resulted path.
3: **end for**

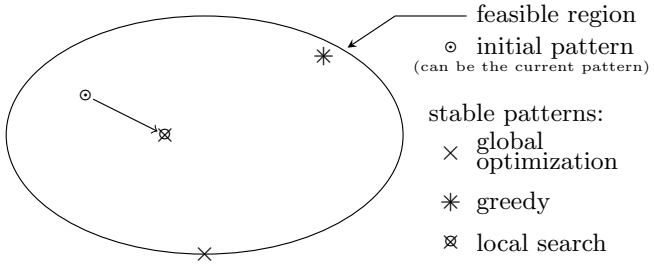


Fig. 5. Local search kernel finds a "nearby" stable routing pattern.

C. Local Search Algorithm

Algorithm 2 builds a stable routing pattern for a priority class from scratch. However, the centralized controller may have derived a feasible routing pattern from some heuristics already, and the only question left is how to shape the existing feasible routing pattern to be a stable one. To deal with the situation, we take a local search approach. A local search algorithm finds a "nearby" stable pattern by "improving" the feasible 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 the local search algorithm implies the stability of the resulted solution.

The 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 feasible solution by using the strictly shorter route given by the optimal solution to $R^n(t)$. The design is summarized in Algorithm 3, and its stability is shown in Appendix A.

An important question about Algorithm 3 is whether the algorithm will terminate in polynomial-time. Proposition 2 confirms that Algorithm 3 terminates in $O(|N_{=\pi}|^2)$ iterations, and hence the Algorithm 1 with Algorithm 3 as its kernel terminates in $O(|N|^2)$.

Proposition 2. *Algorithm 3 terminates in $2^{|E|}|N_{=\pi}|^2$ iterations.*

Proof. We show the proposition by considering the longest execution. Notice that the while loop in Algorithm 3 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

Algorithm 3: Local Search Algorithm

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

most $2^{|E|}$ different costs that can be incurred by a route, the longest execution has at most $|N_{=\pi}|$ (every $|N_{=\pi}|$ iterations) $\times|N_{=\pi}|$ (possible flows) $\times 2^{|E|}$ (times of cost decreasing) $= 2^{|E|}|N_{=\pi}|^2$ iterations before termination. □

D. Comparison amongst the Kernels

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

Since solving the global optimization problem is NP-hard, it will take exponential-time to solve; the greedy algorithm checks each flow only once, and hence it is linear-time solvable; and the time-complexity of the local search kernel has been given by Proposition 2.

We can observe from the table that the global optimization and the greedy are two extreme cases. Solving the global optimization is the most computationally expensive with the optimal cost (per priority class), while the greedy algorithm loses the optimality in exchange for lower computational complexity.

Besides these two extremes, the local search algorithm provides flexibility with a quadratic-time complexity by allowing the specification of an initial feasible routing pattern. The importance of flexibility is argued in the following sense: Unless the centralized controller aims to minimize the same objective as the local routers, it may select a stable routing pattern based on some other criterion, such as route disturbance. In that case, the flexibility allows the controller 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 7).

V. Imperfect 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, the preferred approach for it is to do nothing until complete information is available. This approach works in the hybrid framework since the distributed routing continues to work. In this section, we also consider a second approach where centralized controller takes action even if the information is incomplete.

TABLE II
Comparison of Algorithm 1 with different Kernels.

	Global Optimization	Greedy	Local Search
Cost Effectiveness	optimal	depending on the order	depending on the order and the initial pattern
Time Complexity	in general $O(2^{\lvert N\rvert})$	$O(\lvert N\rvert)$	$O(\lvert N\rvert^2)$
Initialization Allowance	no	no	yes

As to how the centralized controller can pursue a stable routing pattern under imperfect 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 perfect 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 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 [27] 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.

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.

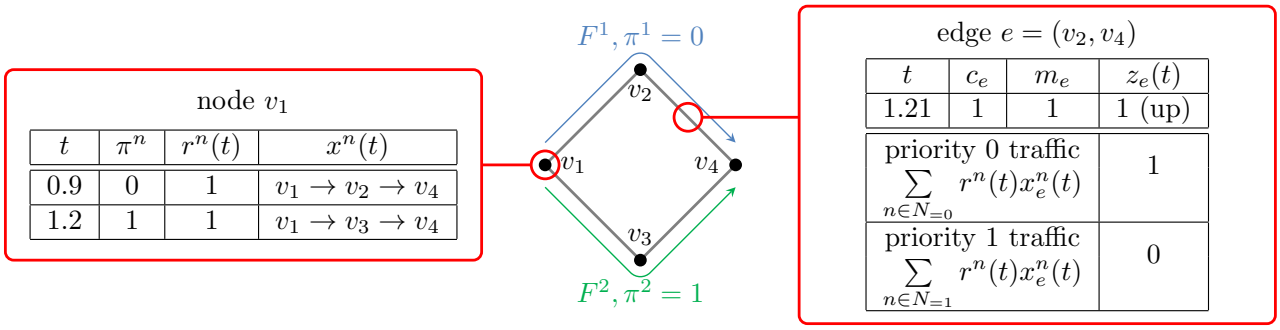


Fig. 6. The PCEP and BGP-LS information is marked with time stamps and stored as a graph.

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.

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 $e \in E$, $x^n(t) \leq z_e(t)$ 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. 6 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 feasible solution based on the effective capacities remains feasible 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 $2 - 0.6 = 1.4$.

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. 6 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, Algorithm 1 is implemented as well as the three kernel algorithms: global optimization, greedy, and local search. Dijkstra algorithm is programmed to solve shortest path problems, which is a critical component not only for the distributed routing, but also for the greedy and the local search kernels. In order to deal with the $C_\pi(t)$ in the optimization kernel, we express it as an integer program with the decision variables $x_e^n(t)$, and utilize COIN-OR Branch and Cut (CBC) [28] as the integer program solver.

The global optimization problem $C_\pi(t)$ 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 the greedy kernel to select a subset of the flows that admits a feasible solution, and route those flows by the global optimization kernel when $C_\pi(t)$ is infeasible.

We assign each flow an id number and the greedy kernel 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 the local search kernel when adding flows into the circular buffer.

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

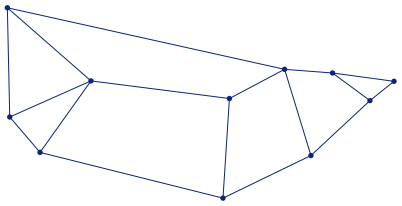


Fig. 7. The Internet2 Network topology (previously known as *Abilene Network*).

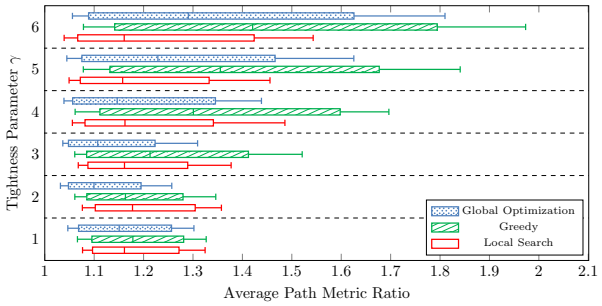


Fig. 8. The 1^{st}-5^{th}-50^{th}-95^{th}-99^{th} percentiles of the average path metric ratio of the routed flows.

B. Statistical Performance

We first compare the performance of the kernels based on the layer-3 topology of the Internet2 Network [29], which has 11 nodes and 17 edges (Fig. 7). 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 distributed routing. 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

$$\text{path metric ratio of } F^n \text{ at time } t = \frac{\sum_{e \in E} m_e x_e^n(t)}{\sum_{e \in p^n(t)} m_e}.$$

According to the definition, the path metric ratio is always greater than or equal to one. We compare the average of the

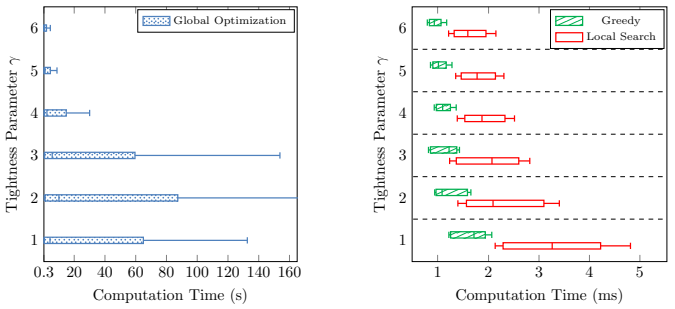


Fig. 9. The 1^{st}-5^{th}-50^{th}-95^{th}-99^{th} percentiles of the computation time.

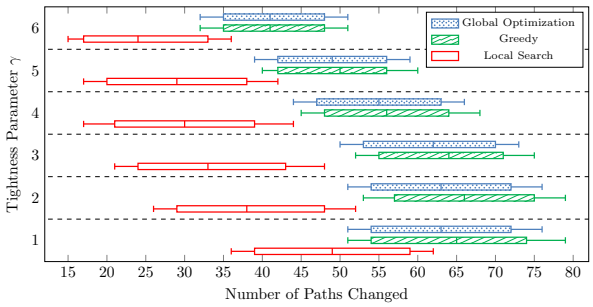


Fig. 10. The 1^{st}-5^{th}-50^{th}-95^{th}-99^{th} percentiles of the number of paths changed.

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.

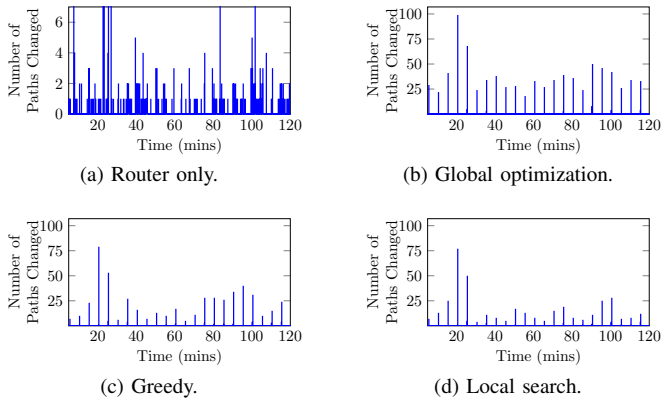
In Fig. 8, the local search kernel performs similar to the greedy kernel under small γ. When the network becomes highly utilized, the local search kernel beats the greedy kernel. The global optimization outperforms the other two kernels when γ is small, as we would expect. As γ growing larger, not all the flows can be routed, and the greedy kernel is introduced to select the routed flows, which causes the performance downgrade of the global optimization kernel. While the local search algorithm still performs well in that case.

2) Computational Efficiency: Even though the global optimization kernel 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.

In Fig. 9, the global optimization kernel requires up to 5 orders of magnitude longer time than the other two kernels. The greedy kernel 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 the local search kernel is merely around twice the time needed by the greedy kernel 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. 10, the global optimization kernel and the greedy kernel both change around 1.5 to 2 times more routes than the local search kernel does when stabilizing the network. The reason is that the heuristic in the local search kernel applies the current routing pattern as the initial feasible pattern, and the local search kernel will terminate once a stable pattern is found by changing one path at a time to search locally, which usually changes only small fraction of the routes.



(a) Router only. (b) Global optimization.

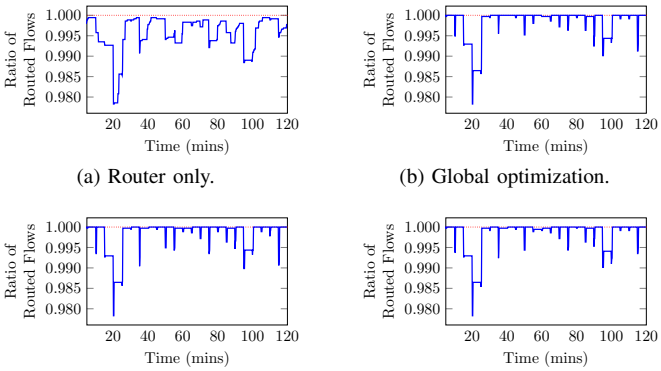
(c) Greedy. (d) Local search.

Fig. 11. Number of paths changed during $5 - 120$ mins after the system starts.

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.

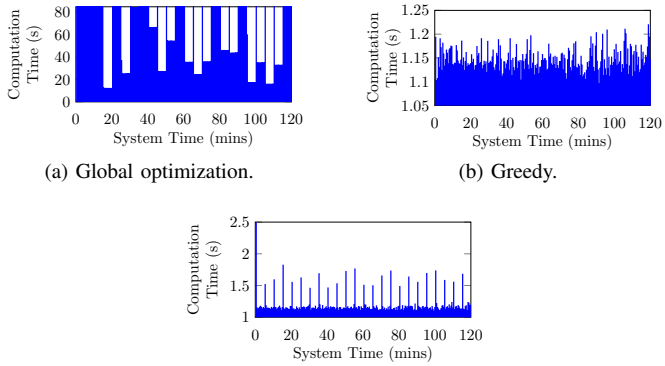
1) Routing Stability: Firstly, we inspect how well our methods can stabilize the routes. Four different scenarios are considered. In the router only scenario, each flow is routed through the shortest path at its source router. The other



(a) Router only. (b) Global optimization.

(c) Greedy. (d) Local search.

Fig. 12. Ratio of routed flows.



(a) Global optimization. (b) Greedy.

(c) Local search.

Fig. 13. Computation time of the controller.

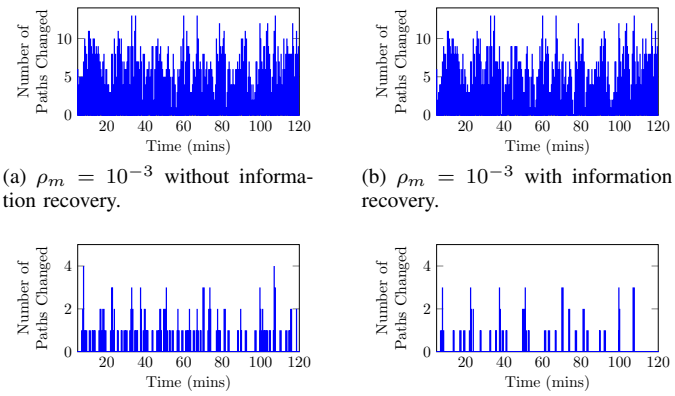
three scenarios correspond to the Algorithm 1 with global optimization, greedy, and local search kernels.

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. 11 the number of changed path from 5 to 120 minutes after the system starts.

Starting with a stable routing pattern, distributed routing still needs several cycles before reaching a stable routing pattern as shown in Fig. 11a. All proposed kernels can 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-C4.

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 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. 12 under the four different scenarios. We can observe that local routers (Fig. 12a) mostly support fewer flows than those cases with controller intervention (Fig. 12b-12d). 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. 13. The global optimization kernel requires considerable computation time (Fig. 13a). 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 the global optimization kernel's ability to deal with the quick variation of the sending rate. The greedy kernel, as expected, takes almost constant time to route the traffic (Fig. 13b), while the local search kernel 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. 13c).



(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.

Fig. 14. Number of paths changed under partial information.

TABLE III
TOTAL NUMBER OF PATHS CHANGED.

γ	Global Optimization	Greedy	Local Search
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

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. Constrained by the computational power, we let the integer solver run for at most 150 seconds per priority class, and stabilize the resulted pattern by the local search kernel to form the global optimization solution in this section.

In Table III, the global optimization kernel and the greedy kernel change up to 2 times more routes than the local search kernel. 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. The greedy

TABLE IV
AVERAGE RATIO OF ROUTED FLOWS.

γ	Global Optimization	Greedy	Local Search
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	0.981571	0.987322
4		0.900165	0.932775
6	intractable	0.802259	0.878906
8		0.685143	0.843711

kernel 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. The local search kernel changes the least paths to obtain a stable routing pattern, which is consistent with the result in Section VII-B3.

Table IV shows that the local search algorithm outperforms the greedy kernel. Besides the lightest loading condition – under which all the kernels can route 99.5% of the flows – the local search kernel can route up to 10% more flows than the greedy kernel under those heavier loading conditions.

5) Partial Information: To realize how well the information recovery phase in the system design can help alleviate the impact of information missing, we let the PCEP information be missing with *information missing rate* ρ_m and run the local search algorithm 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. 14a and 14b). 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. 14c and 14d).

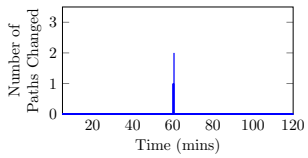
6) Inconsistent Information: We also consider the issue of inconsistent information. Similar to the partial information simulation, we fix the sending rate and make each router send a PCEP message which is inconsistent with the BGP-LS information with *information inconsistent rate* ρ_i. The inconsistent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

The simulation result is a landslide: In all the 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 the information recovery phase change paths, and the network is less stable as ρ_i increases.

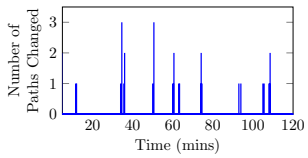
VIII. RELATED WORK

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



(a) $\rho_i = 10^{-3}$ without information recovery.



(b) $\rho_i = 10^{-2}$ without information recovery.

Fig. 15. Number of paths changed under inconsistent information. No path is changed if information recovery phase is adopted.

[12]. [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, in which the SDN controller instructs the local routers to route, but still lets the local routers compute the distributed routing decisions as the failure backup plan. By doing so, the system can react to failure quickly without the centralized controller's intervention. However, the design of such system requires two local router states: the primary state when the local routers listen to the centralized controller, and the backup state when a failure is detected. Our hybrid SDN does not require the routers to differentiate these two states. 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.

Stability: The network stability issue has been discovered for a long time [30] and scrutinized for distributed routing mechanisms, including OSPF [14], BGP [19], [31], [32], and some other path-vector protocols [16], [18]. Unlike those well-defined objectives such as routing table size or cost metric, stability is a rather fuzzy idea and several different definitions have been proposed to describe it in the literature. [14] evaluates stability through network convergence time, the routing load on processors, and route flaps. [15] defines stability as a consensus state among the different autonomous systems (ASs) and provides a set of guidelines for the ASs to route stably without coordination. [16] studies the stable paths problem, in which the path assignment is stable if the next hop of every node in the path is the best possible next hop solution at the node. [17] takes into account not only the routing but also the rate control. A stability condition based on convergence to equilibrium points is presented for such a joint control. [18] defines the stability metric for path-vector routing protocols (such as BGP) in terms of the number of changes made to the routing table. [33] proposes the ideas "prevalence" and "persistence" to describe routing stability when tracing the route changes in the Internet. Prevalence is how likely the route can be observed in the future, and persistence is how long a route can stay unchanged.

Our idea of stability is similar to "persistence" in [33]. A stable state is achieved through a consensus between the

centralized controller and the local routers as in [15], i.e., the centralized controller's decision is the best possible one for the local routers as in [16].

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.

Appendix

We first show that Algorithm 1 generates a stable routing pattern. 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, the priority of the flow. Therefore, at the iteration that considers the flows indexed by $N_{=\pi^n}$, 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.

Next, we show that the proposed kernel algorithms give stable routing patterns.

- Global Optimization Algorithm (The optimal solution to $C_\pi(t)$): 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 $x_e^n(t)$ 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.
- Greedy Algorithm (Algorithm 2): 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.
- Local Search Algorithm (Algorithm 3): Again, 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 Algorithm 3 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 Algorithm 3 will not terminate. Thus, the resulted routing pattern must be stable.

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.

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

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

[4] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer, "Achieving high utilization with software-driven WAN," *ACM SIGCOMM CCR*, vol. 43, no. 4, pp. 15–26, 2013.

[5] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, and A. Vahdat, "B4: Experience with a globally-deployed software defined WAN," *ACM SIGCOMM CCR*, 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, 2012.

[7] M. F. Bari, R. Boutaba, R. Esteves, L. Z. Granville, M. Podlesny, M. G. Rabbani, Q. Zhang, and M. F. Zhani, "Data center network virtualization: A survey," *IEEE Commun. Surveys & Tutorials*, vol. 15, no. 2, pp. 909–928, 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, 2013.

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

[10] M. Yu, J. Rexford, M. J. Freedman, and J. Wang, "Scalable flow-based networking with DIFANE," *ACM SIGCOMM CCR*, vol. 40, no. 4, pp. 351–362, 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," *ACM SIGCOMM CCR*, vol. 41, no. 4, pp. 254–265, 2011.

[12] S. Sezer, S. Scott-Hayward, P. K. Chouhan, B. Fraser, D. Lake, J. Finnegan, N. Viljoen, M. Miller, and N. Rao, "Are we ready for SDN? implementation challenges for software-defined networks," *IEEE Commun. Mag.*, vol. 51, no. 7, pp. 36–43, 2013.

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

[14] A. Basu and J. Riecke, "Stability issues in OSPF routing," *ACM SIGCOMM CCR*, 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, 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, 2002.

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

[18] D. Papadimitriou, F. Coras, and A. Cabellos, "Path-vector routing stability analysis," *SIGMETRICS Perform. Eval. Rev.*, 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, G. Choudhury, B. Cortez, A. Goddard, N. Padi, A. Raghuram, K. Tse, S. Tse, A. Wallace, and K. Xi, "Evolving to an SDN-enabled ISP backbone: Key technologies and applications," *IEEE Commun. Mag.*, vol. 54, no. 10, pp. 129–135, 2016.

[21] G. Choudhury, B. Cortez, A. Goddard, N. Padi, A. Raghuram, S. Tse, A. Wallace, and K. Xi, "Centralized optimization of traffic engineering tunnels in a large ISP backbone using an SDN controller," *INFORMS Optimization Society Conference*, Mar. 2016.

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

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

[24] D. P. Williamson and D. B. Shmoys, *The Design of Approximation Algorithms*. Cambridge university press, 2011.

[25] J. Vasseur and J. Le Roux, "RFC 5440: Path computation element (PCE) communication protocol (PCEP)," 2009.

[26] H. Gredler, J. Medved, S. Previdi, A. Farrel, and S. Ray, "RFC 7752: North-bound distribution of link-state and traffic engineering (TE) information using BGP," 2016.
[27] "NERC reliability concepts," http://www.nerc.com/files/concepts_v1.0.2.pdf.
[28] "CBC (COIN-OR branch and cut)," https://projects.coin-or.org/Cbc.
[29] "The Internet2 network," http://www.internet2.edu/.
[30] C. Labovitz, G. R. Malan, and F. Jahanian, "Internet routing instability," *ACM SIGCOMM CCR*, vol. 27, no. 4, pp. 115–126, Oct. 1997. [Online]. Available: http://doi.acm.org/10.1145/263109.263151
[31] R. Govindan and A. Reddy, "An analysis of Internet inter-domain topology and route stability," in *Proc. IEEE INFOCOM*, vol. 2. IEEE, 1997, pp. 850–857.
[32] K. Varadhan, R. Govindan, and D. Estrin, "Persistent route oscillations in inter-domain routing," *Computer networks*, vol. 32, no. 1, pp. 1–16, 2000.
[33] V. Paxson, "End-to-end routing behavior in the Internet," *ACM SIGCOMM CCR*, vol. 26, no. 4, pp. 25–38, 1996.