Flow Migrations in Software Defined Networks: Consistency, Feasibility, and Optimality

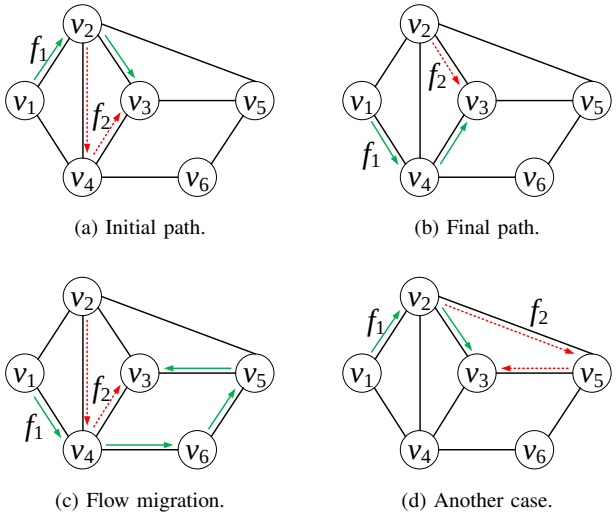
Yang Chen, Huanyang Zheng, and Jie Wu
Department of Computer and Information Sciences, Temple University, USA
Email: {yang.chen, huanyang.zheng, jiewu}@temple.edu

***Abstract*—Software defined networks always need to migrate flows to update the network configuration for a better system performance. However, the existing literature does not take flow path overlapping information into consideration when flows' routes are re-allocated. Consequently, congestion happens, resulting in deadlocks among flows and link resources, which will block the update process and cause severe packet loss. This paper resolves deadlocks with the help of spare resources during the network update. An algorithm is proposed to determine the feasibility of the consistent flow migration. We demonstrate that even if there are multiple consistent migration plans, finding the optimal one that occupies the fewest leisure bandwidth resources is NP-hard. An approximation algorithm is proposed to sequentially migrate flows in a consistent way. Extensive simulations show that our solution achieves a much smaller traffic loss rate at the cost of an affordable spare link resource usage compared to prior methods.**

***Index Terms*—Software Defined Networks (SDNs), spare links, consistent flow migration, feasibility and optimality.**

I. Introduction

Configurations of Software Defined Networks (SDNs) are routinely updated in order to achieve a shorter transmission latency and higher bandwidth utilization [1]. SDN utilizes its centralized controller to configure and execute new update policies [2]. With the rise and development of SDNs, the requirements of high performance networks are becoming more and more intense. For example, in the aspect of the packet loss rate, data centers usually claim it to be around 2% [3], while the requirements of wide area networks (WANs) and carrier-grade networks are much higher [4]. Specifically, the carrier-grade performance is often associated with the term five nines, representing an availability of 99.999%. However, complex network updates are becoming more and more common. For example, Microsoft's SWAN [5] and Google's B4 [6] run updates every few minutes, hundreds of times per day. Key challenges come from the fact that some unexpected events during the update may happen. These unexpected events will disrupt network functionality and cause traffic congestion as well as packet loss, resulting in a Quality of Service (QoS) disqualification. These events consist of unpredicted, long switch update times and abnormal communication delays between the controller and the switch. There are multiple reasons for those chaotic situations, such as imperfect clock synchronization and transient controller-data plane disconnection. In this paper, we study the consistent flow migration problem, which requires moving network flows from their initial routing paths to the target ones in a lossless way [7].



(a) Initial path. (b) Final path. (c) Flow migration. (d) Another case.

Fig. 1: An example to illustrate flow migrations.

To prevent the above anomalies, we explore three properties in network flow migrations: *consistency*, *feasibility*, and *optimality*. Consistency requires that the bandwidth demands of all flows be satisfied during the whole migration process and there be no congestion and packet loss during the update procedure. Because of the demanding packet loss rate, consistency is the most important property of flow migration. However, due to insufficient bandwidth resources, a consistent flow migration may not always exist. Therefore, we use the feasibility to refer to whether or not a consistent flow migration exists. The optimality is raised if there are multiple consistent flow migrations. This means finding the plan that takes the fewest spare resources to finish the update. This paper explores the feasibility and optimality of flow migrations under the consistency constraint in SDN updates.

Fig. 1 shows a toy example with two flows, f_1 and f_2. In each graph, the nodes represent the switches and the edges are the links, all of which are bidirectional. Each directional link has a capacity of 1 Gbps and each flow has a bandwidth of 1 Gbps. Figs. 1a and 1b show the initial and final routing paths of f_1 and f_2. Unfortunately, neither f_1 nor f_2 can be directly migrated to their final paths. This is because the initial routing path of f_1 overlaps the final routing path of f_2, and vice versa. Such a deadlock is challenging in terms of flow migrations. This paper uses *spare resources* to achieve consistent flow migrations. A feasible flow migration is shown in Fig. 1c,

in which f_1 is migrated to an intermediate routing path on three spare links (v_4 to v_6, v_6 to v_5, and v_5 to v_3). Such an intermediate routing path of f_1 releases necessary bandwidth resources for f_2, and thus, breaks the deadlock. However, the flow migration in Fig. 1c may not be the optimal one since, as shown in Fig. 1d, only two spare links can sufficiently fulfill flow migrations. Instead of migrating f_1 to an intermediate routing path on three spare links, we can migrate f_2 to an intermediate routing path on two spare links (v_2 to v_5 and v_5 to v_3). Such a migration breaks the deadlock and uses fewer spare resources than the migration in Fig. 1c, which is more desirable among the feasible consistent migrating plans.

Due to the complexity of the assistant intermediate routing paths, our problem becomes challenging. We study the consistent flow migration problem which requires that the bandwidth demands of all flows be satisfied during the entire migration process. Moreover, we strive to find the optimal one when several consistent update plans exist. Existing works focus on the resource dependencies between the initial and final routing paths of flows. They aim at finding a specific migration order of flows [8] or using a two-phase tagging scheme [9] to complete the lossless flow migration to its final state. However, when the remaining bandwidth resources are insufficient in the flows' final paths, these methods cannot make any progress except in reducing flow rates. Consequently, such flow rate limitations will lead to packet loss as well as QoS deterioration [10]. In contrast, our paper proposes a generic approach to consistently migrate flows with the help of spare paths. An algorithm is proposed to determine the feasibility of consistent flow migrations. Furthermore, to efficiently resolve deadlocks, we demonstrate that even if there are multiple consistent flow migrations, it is NP-hard to find the optimal one that occupies the fewest bandwidth resources. Therefore, a greedy algorithm is proposed to consistently migrate flows in sequence within a reasonably competitive ratio.

The remainder of this paper is organized as follows. Section 2 surveys related works. Section 3 describes the model and formulates the problem. Section 4 discusses the feasibility and optimality of flow migrations in SDNs. Section 5 includes the experiments. Finally, Section 6 concludes the paper.

II. Related Work

There are two basic mainstream methods for flow migration implementation: ordering [11] and two-phase [9]. The ordering strategy usually updates the forwarding table of the switches one-by-one in a specified order, which is carefully calculated in order to preserve some required properties, like being loop-free and blackhole-free. However, this order might not exist when it needs to guarantee both forwarding and policy demands. The two-phase scheme installs both the initial and final rules on all switches and tags packets with a rule's version number. This method ensures the success of the update, but it doubles the number of rules on every switch, which wastes expensive and power-hungry Ternary Content Addressable Memory (TCAM) memory resources. This paper performs the two-phase commit using version numbers for flow migrations.

The major drawback of the basic ordering and two-phase methods of flow migration is that they cannot guarantee consistency as congestion may exist during flow migrations. To obtain consistent flow migrations, we summarize that there are mainly three kinds of approaches: link capacity reservation [5], intermediate state-involvement [12], and time-awareness update [13]. As a typical link capacity reservation approach, SWAN [5] has two main results. First, if a fraction of the capacity is guaranteed to be free on each link for the old and new flows, SWAN can update the network in constant steps. Second, in order to solve the problem efficiently, they use linear programming to check whether a solution with bounded steps exists. However, when there is no slack on some edges, it is unlikely that this algorithm will halt in certain steps, which will lead to a high computation complexity. A representative of the intermediate state-involvement approach, ZUpdate [12], attempts to compute and execute a sequence of steps to migrate flows in a congestion-free way. However, it stretches the update time, which makes the chaos of traffic migration last longer. A typical time-awareness update approach, Dionysus [14], dynamically schedules the process based on the runtime differences of switches' update speeds, instead of a previous static ordering of rule updates. It is a path-based update method, meaning that a flow is scheduled to be migrated to its final path when all the links along its new path have enough bandwidth resources. If a single link along the path is not available, this method can only wait and waste a lot of link resources. Another kind of time-awareness plan is based on time synchronization technology.

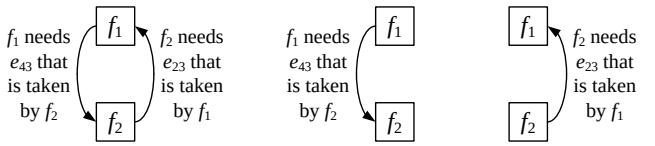
III. Framework

Networking updates in SDNs consist of hardware upgrades, deployment modifications, and configuration changes. It is necessary to routinely update networks for better performances in failure recovery, transmission latency, and bandwidth utilization. However, this planned maintenance always puts the paths of network flows in a state of flux. Flow migration is an important kind of configuration change. It is a common source of instability in SDNs, leading to update deadlocks, broken connectivity, forwarding loops, and access control violations.

A. Model and Formulation

Our flow migration scenario is based on a directed network, $G = (V, E)$, where V is a set of vertices (i.e., switches) and $E \subseteq V^2$ is a set of directed edges (i.e., links). We use v_i to denote the i-th vertex and use e_{ij} to denote the edge from v_i to v_j. Each edge is capacitated, and we use c_{ij} to denote the bandwidth capacity of e_{ij}. The network, G, includes a set, F, of given flows. We use f_k to denote the k-th flow, and its bandwidth demand is d_k. The initial and final routing paths of f_k are denoted by p_k and p_k^*, respectively. A path is an ordered set of edges. For example, in Fig. 1a, we have $p_1 = \{e_{12}, e_{23}\}$ and $p_2 = \{e_{24}, e_{43}\}$.

This paper studies consistent flow migrations which require that the bandwidth demands of all flows be satisfied during the entire migration process. A round-by-round manner is used



(a) Deadlock in Fig. 1a. (b) Flow in Fig. 1c. (c) Flow in Fig. 1d.

Fig. 2: RDG, deadlock, and spare path.

to migrate flows from their initial to final routing paths. In each round, only one flow is migrated to another path. Let p_k^r denote the routing path of f_k at the round r. Let b_{ij}^r denote the bandwidth usage of e_{ij} during round r, which is equal to the total bandwidth demands of its passing flows. We have R rounds in total, i.e., $0 \leq r \leq R$. Our problem is similar to the Klotski game, and is formulated as:

$$\text{minimize} \quad \sum_{e_{ij} \in E} [\max_{0 \leq r \leq R} b_{ij}^r] \tag{1}$$

$$\text{s.t.} \quad b_{ij}^r \leq c_{ij} \qquad \forall 0 \leq r \leq R, e_{ij} \in E \tag{2}$$

$$\left|\{ f_k \,|\, p_k^r \neq p_k^{r+1}\}\right| = 1 \qquad \forall 0 \leq r \leq R \tag{3}$$

$$p_k^0 = p_k \text{ and } p_k^R = p_k^* \qquad \forall f_k \in F \tag{4}$$

In Eq. 1, the maximum bandwidth usage of e_{ij} among all rounds is $\max_r b_{ij}^r$. The objective is to minimize the total maximum bandwidth usage among all edges during flow migrations. We aim to use the minimum spare bandwidth resources to migrate flows. Two constraints are involved. Eq. 2 means that the bandwidth usage of e_{ij} is smaller than or equal to its capacity, c_{ij}. Eq. 3 means that only one flow is migrated in each round, in terms of changing its routing path. Here, $\{ f_k \,|\, p_k^r \neq p_k^{r+1}\}$ is the set of flows that change their routing paths. Meanwhile, $|\cdot|$ the denotes set cardinality. Eqs. 2 and 3 represent the consistency requirement during flow migrations. However, this requirement may not be always satisfied, leading to the feasibility problem. Finally, Eq. 4 requires each flow to migrate from its initial path to its final one.

B. Resource Dependency Graph

We start with the concept of resource dependency:

***Definition** 1:* Flow f_k depends on a minimal set of other flows (denoted by F_k) if f_k could be immediately migrated from its current path to its final path after the removal of F_k, but not after the removal of $F_k \setminus \{f\}$ for $\forall f \in F_k$.

F_k may be an empty set, and f_k may depend on different minimal sets. If we map flows to nodes and map dependency relationships to directed edges, a resource dependency graph can be obtained. An example graph is shown in Fig. 2a, which corresponds to Fig. 1a. It can be seen that f_1 and f_2 depend on each other in terms of their initial routing paths.

***Definition** 2:* Let each flow correspond to one of its minimal dependency sets. The Resource Dependency Graph (RDG) is defined by mapping flows and their dependency sets to nodes and directed edges, respectively. Given an RDG, a closed walk without repeated nodes is defined as a deadlock.

Definition 2 reveals resource deadlocks in flow migrations. Let l_i denote the i-th deadlock and let L denote the set of deadlocks. Note that multiple RDGs can be obtained for a given network since a flow may depend on different minimal flow sets. Once the RDG is determined, we have:

***Theorem** 1:* If the RDG does not include a deadlock (i.e., $L = \emptyset$), a feasible solution could use the topological order in the RDG to migrate flows. Each flow is immediately migrated from its initial to final paths.

The major challenge of our problem is to resolve deadlocks while migrating flows. As previously mentioned, our problem aims to use minimum spare bandwidth resources to break deadlocks by migrating flows. As shown in Figs. 1c and 2b, three spare links (e_{46}, e_{65}, and e_{53}) are used to break the dependency from f_2 to f_1. As shown in Figs. 1c and 2c, two spare links (e_{25} and e_{53}) are used to break the dependency from f_1 to f_2. These spare bandwidth resources are formally defined as spare paths with respect to deadlocks:

***Definition** 3:* The spare path is defined for a given flow, f. It is a path that (i) has enough bandwidth to hold f, (ii) has the same source and destination as f, and (iii) is different from the initial and final paths of f.

***Definition** 4:* Spare path collection is a set of spare paths for flows in a given deadlock in an RDG. Once flows in the deadlock are migrated to the corresponding spare paths, then (i) all remaining flows in this deadlock can be migrated to their final paths following the dependency order and (ii) all flows in the spare paths can be migrated back.

Note that a deadlock might have multiple spare collections. For example, the deadlock in Fig. 2b includes two spare path collections: one collection includes one path of $\{e_{46}, e_{65}, e_{53}\}$, and another collection includes one path of $\{e_{25}, e_{53}\}$. The key idea of the spare path collection is that they can resolve a deadlock, without considering intersections among different deadlocks. Let S_i denote the set of spare path collections for deadlock l_i. To reduce the complexity, we limit the size of spare path collections: The hop length of a spare path is no more than H and the cardinality of a spare path collection is no more than C. Meanwhile, H and C are pre-determined parameters. Given an RDG, all spare path collections for each deadlock can be determined through an exhaustive search. Shorter spare paths are preferred over longer spare paths as they use fewer total bandwidth resources. Note that H and C bring a trade-off between accuracy and time complexity. A larger H and C can explore more possible spare path collections at the cost of an exponentially larger time complexity.

IV. Feasibility and Optimality

This section shows that the feasibility of the consistent flow migrations can be determined with the assistance of spare path collections. However, even if multiple consistent flow migrations exist, searching for the optimal solution that occupies the fewest bandwidth resources is NP-hard.

A. Feasibility

This subsection studies the feasibility problem [15], i.e., whether a consistent flow migration exists or not. The idea is

Algorithm 1 Feasibility Determination

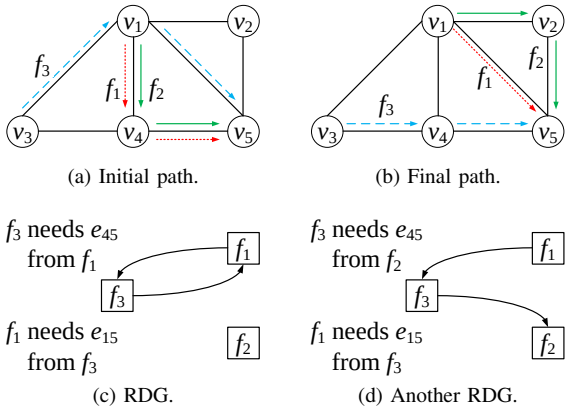
Input: Network G and flow set F;
Output: Feasible solution existence;

1: **for** each flow $f_k \in F$ **do**
2: Compute its minimal dependency set, F_k.
3: Generate RDGs based on flow dependency.
4: **for** each RDG of network, G **do**
5: Find the set of deadlocks, L, in this RDG.
6: **for** each deadlock $l_i \in L$ **do**
7: Find the set of spare path collections, S_i, for l_i.
8: **if** $S_i = \emptyset$ **then**
9: **continue** to the next RDG;
10: **return** a feasible solution exists;
11: **return** a feasible solution may not exist;

to use spare paths to break deadlocks. As shown in Theorem 1, if the RDG does not include a deadlock, a feasible solution could use the topological order to migrate flows. Algorithm 1 is proposed to determine feasibility. For a given network, lines 1 to 3 construct the RDGs. Note that different RDGs can be obtained for the same network and flows since a flow might depend on different minimal flow sets. In lines 4 to 10, Algorithm 1 checks each possible RDG. If there is an RDG in which all deadlocks can be solved by spare path collections, Algorithm 1 returns that a consistent flow migration exists in line 10. If deadlocks cannot be broken in all RDGs, Algorithm 1 returns infeasibility in line 11.

Fig. 3 shows an example for Algorithm 1. Figs. 3a and 3b show the initial and final flow-routing paths, respectively. Links e_{14}, e_{34}, and e_{45} have bandwidth capacities of 2 Gbps, while all other links have bandwidth capacities of 1 Gbps. Each flow has a bandwidth of 1 Gbps. In this scenario, f_1 depends on f_3, and f_2 does not depend on other flows. f_3 can depend on either f_1 or f_2, leading to two different RDGs as shown in Figs. 3c and 3d. f_3 needs e_{45} to be migrated to its final routing path. Meanwhile, e_{45} can be released for f_3 by either f_1 (in Fig. 3c) or f_2 (in Fig. 3d). Algorithm 1 will traverse each RDG in lines 4 to 10. Let us start with Fig. 3c, which includes only one deadlock among f_1 and f_3. This deadlock includes two spare path collections: one has a spare path of $\{e_{12}, e_{25}\}$ to migrate f_1 and another one has a spare path of $\{e_{31}, e_{12}, e_{25}\}$ to migrate f_3. Therefore, Algorithm 1 terminates.

Algorithm 1 correctly determines the feasibility. If there is an RDG in which all deadlocks have spare path collections, Algorithm 1 returns that a consistent flow migration exists. This is simply because, by definition, each deadlock can be broken by one of its spare path collections. The corresponding flows can be migrated to their spare paths and can be migrated back after breaking the deadlock. Algorithm 1 is a conservative approach: a feasible solution can exist but Algorithm 1 cannot identify its existence. For a given network, the number of RDGs may be exponential with respect to the number of flows. This is because each flow may depend on different minimal



(a) Initial path. (b) Final path. (c) RDG. (d) Another RDG.

Fig. 3: An example to illustrate Algorithm 1.

flow sets. An example is shown in Fig. 3, in which f_3 can depend on either f_1 or f_2. However, the time complexity of Algorithm 1 can be reduced to a polynomial through capping the cardinalities of spare path collections. The number of RDGs is $O(|F|^C)$ and the number of spare paths is $O(|E|^H)$.

B. Optimality

The previous subsection discussed the problem feasibility, i.e., whether a consistent flow migration exists or not. However, even if multiple consistent flow migrations exist, it is NP-hard to find the optimal one that occupies the fewest bandwidth resources. We start with the problem hardness:

***Theorem** 2:* Our flow migration problem is NP-hard.

The detailed proof is omitted since it is a simple reduction of the set cover problem [16]. Given some elements and a collection of sets of elements, the set cover problem aims to select minimum sets to cover all given elements. We reduce sets and elements to sparse path collections and deadlocks in RDGs. Since our problem is NP-hard, Algorithm 2 is proposed as an approximation algorithm. In line 1, Algorithm 2 initializes the set of spare path collections to be $S = \emptyset$. In lines 2 and 3, it determines all deadlocks in the RDG and their corresponding sets of spare path collections. Lines 4 to 6 include a greedy selection until all deadlocks are broken. In line 5, a spare path collection, s, is selected from the unselected spare paths, $\cup_i S_i \backslash S$. We use $|\{\, l_i \,|\, s \in S_i\}|$ to denote the number of deadlocks that can be broken by s. On the other hand, we use $\sum_{e_{ij} \in S} [\max_r b_{ij}^r]$ to denote the total spare resources used by S. Therefore, $\sum_{e_{ij} \in S \cup \{s\}} [\max_r b_{ij}^r] - \sum_{e_{ij} \in S} [\max_r b_{ij}^r]$ is the marginal gain of spare resources after adding s into S. Note that spare path collections in S may overlap with each other. $\frac{|\{\, l_i \,|\, s \in S_i\}|}{\sum_{e_{ij} \in S \cup \{s\}} [\max_r b_{ij}^r] - \sum_{e_{ij} \in S} [\max_r b_{ij}^r]}$ represents the benefit-to-cost ratio of spare path collection s, in which the benefit is the number of broken deadlocks and the cost is the marginal gain of spare resources. Line 6 updates the deadlocks to be $L = L \backslash \{\, l_i \,|\, s \in S_i\}$, i.e., it removes deadlocks broken by s. Lines 5 and 6 are iterated until all deadlocks are broken. Line 7 returns the result. We have:

***Theorem** 3:* Algorithm 2 achieves an approximation ratio of $O(H \cdot C \cdot \ln |L|)$ for the optimal algorithm.

Algorithm 2 Spare Path Computation

Input: A feasible RDG for network G and flow set F;
Output: The set of spare paths if feasible solutions exist;

1: Initialize the set of spare path collections, $S = \emptyset$.
2: **for** each deadlock, $l_i \in L$, in RDG **do**
3: Find the set of spare path collections, S_i, for l_i.
4: **while** $L \neq \emptyset$ **do**
5: Set $s = \max \frac{|\{\, l_i \,|\, s \in S_i\}|}{\sum_{e_{ij} \in S \cup \{s\}} [\max_r b_{ij}^r] - \sum_{e_{ij} \in S} [\max_r b_{ij}^r]}$ from $\cup_i S_i \backslash S$, and add s into S.
6: Update deadlocks to be $L = L \backslash \{\, l_i \,|\, s \in S_i\}$.
7: **return** S as the set of spare path collections;

Proof: The proof is done through an intermediate problem, which is defined as follows: (i) we map each spare path collection to a set, and map each deadlock to an element; (ii) an element is included in a set if its deadlock can be broken by the spare path collection; (iii) the intermediate problem is the traditional set cover problem that selects the minimum sets to cover all elements [17]. The cost of a set is the total spare resources of its spare path collection, i.e., $\sum_{e_{ij} \in s} [\max_r b_{ij}^r]$.

A greedy algorithm, which iteratively selects the set with the maximum ratio of marginal element coverage to set cost, has an approximation ratio of $O(\ln |L|)$ for the traditional set cover problem. $|L|$ is the number of deadlocks (i.e., elements). However, the set costs should be independent from each other in the traditional set cover problem. In contrast, spare paths can overlap with each other in our problem, meaning that:

$$\sum_{e_{ij} \in S \cup \{s\}} [\max_r b_{ij}^r] - \sum_{e_{ij} \in S} [\max_r b_{ij}^r] \leq \sum_{e_{ij} \in \{s\}} [\max_r b_{ij}^r] \quad (5)$$

The key observation is:

$$\sum_{e_{ij} \in S \cup \{s\}} [\max_r b_{ij}^r] - \sum_{e_{ij} \in S} [\max_r b_{ij}^r] \geq \frac{1}{HC} \sum_{e_{ij} \in \{s\}} [\max_r b_{ij}^r] \quad (6)$$

This is because the spare path collection, s, has at least $H \cdot C$ edges. Each spare path has at most H edges and s has at most C spare paths. As a result, Eq. 6 shows that Algorithm 2 achieves a ratio of $O(H \cdot C \cdot \ln |L|)$. ■

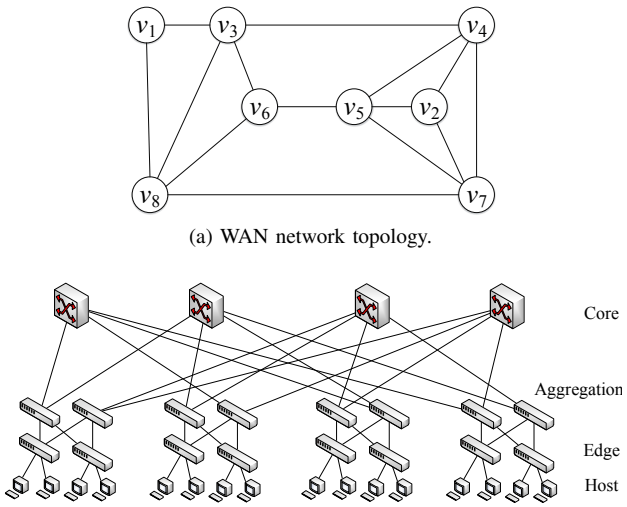
Note that our algorithms do not guarantee a consistent flow migration. This because a consistent flow migration may not exist or be found by Algorithm 1. In this case, flow migrations can be conducted through other well-known approaches such as rate-limiting and buffering [18].

V. Experiment

Experiments are conducted to evaluate the performances of our proposed algorithms. After presenting the topologies and basic settings, the evaluation results are shown from different perspectives to provide insightful conclusions.

A. Experimental Settings

We conduct simulations in two real topologies. The first one is Microsoft's inter-data center WAN topology [14, 19], consisting of 8 switches that are connected as shown in Figure 4a. Each link is two-way and has a capacity of 1-Gbps. The second one is a fat-tree topology [20] for the data centers, shown in Figure 4b. There are 4 core switches, 8 aggregation switches, and 8 edge switches in this network. Each edge switch connects 2 hosts. Each switch has two 10-Gbps ports, resulting in a network with a full bisection bandwidth.



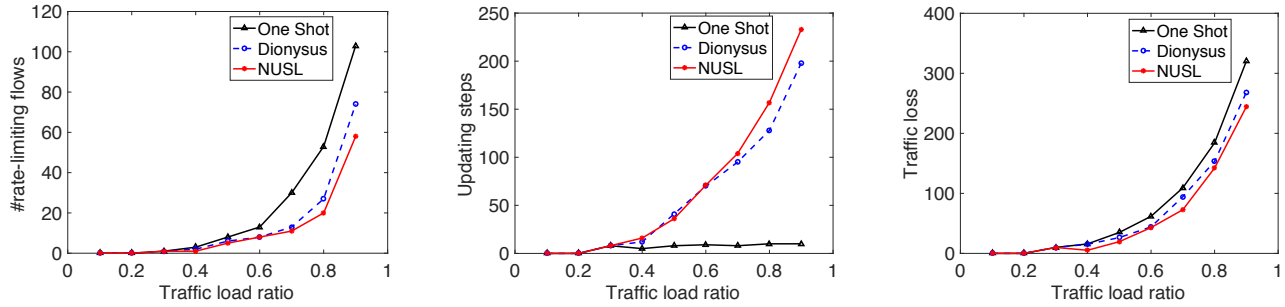
(a) WAN network topology.

(b) Fat-tree topology of data center.

Fig. 4: Network topologies for simulations.

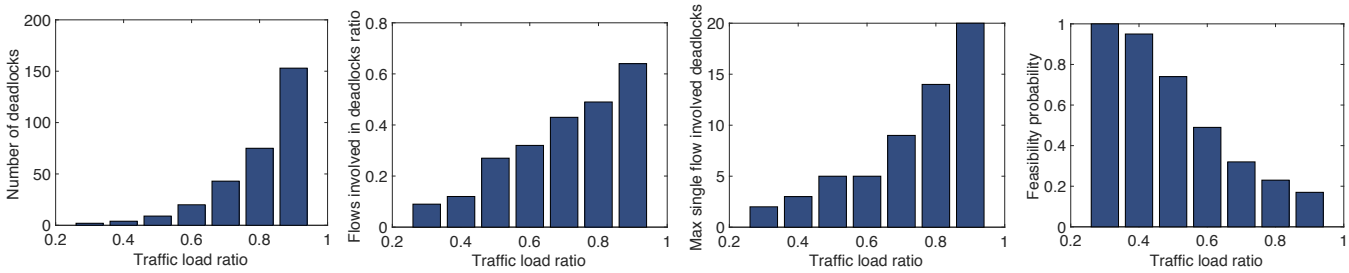
The proposed algorithm Network Update through Spare Links is denoted as NUSL. There are two comparison algorithms in our simulations: One-shot and Dionysus [14]. The One-shot scheme updates the network directly from the initial to the final stage by cutting off all the current flows and allowing new ones in after the network is vacant. It causes severe packet loss and significantly reduces the QoS. The One-shot is not able to meet the demanding loss rate requirement in realistic networks even though it only takes one step to update the network. Dionysus has been specifically introduced in the related work PART. It builds the dependency graph to describe the relationships among different flow states and uses a topological order to migrate flows. As for deadlocks, it opportunistically rates limit flows as zero until all deadlocks are resolved.

We evaluate various aspects under two different topologies in Fig. 4a and Fig. 4b. Our experiments study the relationships between the traffic load and three metrics: the number of rate-limiting flows (when a consistent migration plan does not exist), update steps (time), and traffic loss (the total number of lost packets). We change the traffic load ratio from 10% to 90% to simulate independent variables. Metrics related to deadlocks are also evaluated, including the number of deadlocks, the number of involved flows, the maximum number of deadlocks that a single flow can become involved in, and the probability of finding a feasible update plan with spare paths. Then, we test the spare bandwidth resource usage conditions and the flows involved-in-deadlocks situations. Flows in the network are generated randomly at the granularity of 1Mbps.



(a) Rate limiting flows. (b) Updating time. (c) Traffic loss.

Fig. 5: Performance in the WAN topology.



(a) The number of deadlocks. (b) Flows involved in deadlocks. (c) Max single flow in deadlocks. (d) Feasibility probability.

Fig. 6: WAN deadlock involvement.

We assume the initial and final states of the network are all valid. There are no more new flows coming into the network during the update. No flow paths have any loops, and each link load is within its capacity.

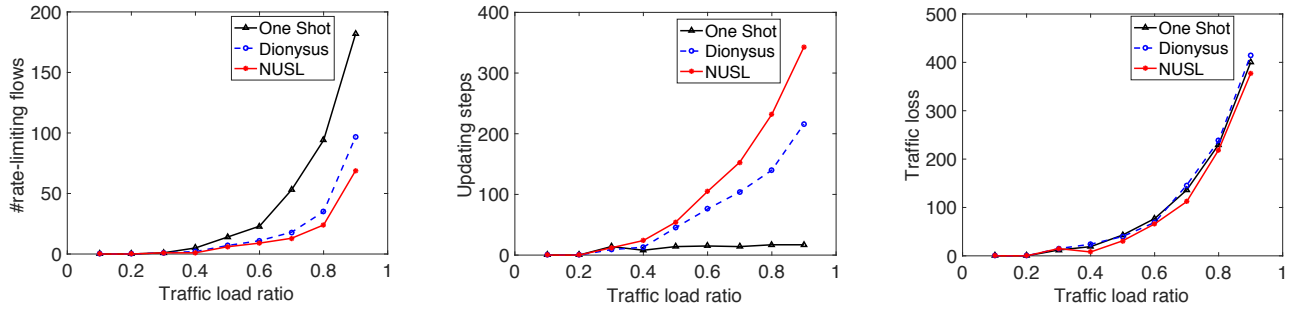
B. Evaluation Results for the WAN

The experimental results in the WAN topology are shown in Fig. 5, Fig. 6, and Tab. I. Algorithm performances are compared with respect to different traffic load ratios. Fig. 5 shows that our algorithm, NUSL, achieves a satisfying result compared to One-Shot and Dionysus. NUSL limits the fewest flows in Fig. 5a and maintains the highest throughput with the lowest traffic loss in Fig. 5c. With a load ratio of 80%, NUSL limits only 47% of the flows in One-Shot and 78% in Dionysus. NUSL does, however, have a longer update time . It takes about 33% more steps than Dionysus with a ratio of 70%. This is because NUSL introduces the intermediate state of the update flows. It utilizes the leisure bandwidth resources to reduce traffic loss during the update process by migrating some flows to their alternate paths. In this way, it vacates some competing link resources to break deadlocks among flows. At the same time, as shown in Fig. 5b, it leads in extra update time. As long as there is no chaos during the update, it is acceptable to stretch the time when the controller orderly operates flows one by one. Dionysus performs just fine. It does not have a good strategy for breaking deadlocks, thus, it experiences more traffic loss than NUSL. Even in update time, Dionysus has no obvious advantages over our approach. One-Shot acts as an ideal baseline under both topologies in our simulation because it represents the most naive solution for updating the network without any strategies and does not have the limitations of the underneath network situations. When One-Shot is applied, there is no new traffic coming into the network during the update. In other words, all flows must be cut off to vacate the link resources they are occupying. As a result, One-Shot takes the fewest steps to accomplish the network update at the expense of the throughput.

Deadlock involvement situations with different metrics are described in Fig. 6. As shown in Fig. 6a, more flows in the network can increase the traffic load, which makes more deadlocks emerge. For example, when the load ratio is 90%, the deadlocks are more than two times of those with a ratio of 80%. The percentage of flows involved in deadlocks also increases quickly, as shown in Fig. 6b. In Fig. 6c, we use the maximum number of deadlocks in which a single flow can get involved, to measure the complexity of the resource dependency graph. The heavier the traffic is, the more deadlocks a single flow can become involved in. Fig. 6d shows the feasibility property of our scheme. From the results, we know that with the assistance of spare paths, it is highly possible to find a lossless update plan when the traffic is light. For example, when the load ratio is less than 50%, its probability is as high as 85%. From these four figures and the analysis, we can conclude that deadlocks are not negligible.

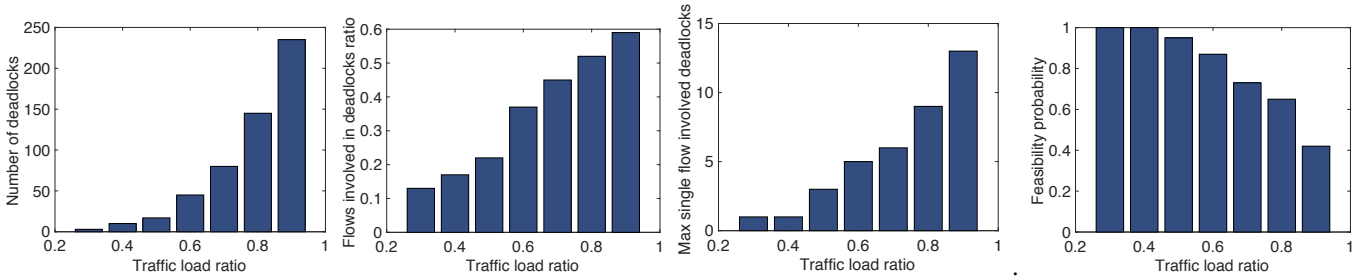
C. Experimental Results for the Fat-tree Topology

The experimental results for the Fat-tree topology in the data centers are shown in Fig. 7. The one-shot approach also acts as an ideal baseline in our simulation. From the figures, we can see that the basic tendency and relationships are almost the same as those of the WAN topology. We will focus on the differences between Fat-tree and WAN. It should be first noted



(a) Rate limiting flows. (b) Updating time. (c) Traffic loss.

Fig. 7: Performance in the Fat-tree topology



(a) The number of deadlocks. (b) Flows involved in deadlocks. (c) Max single flow in deadlocks. (d) Feasibility probability.

Fig. 8: Fat-tree topology deadlock involvement

that Fat-tree can hold more than five times the flows WAN topology can under the same traffic load ratio. By contrast, the number of rate limiting flows and throughput maintenance with NUSL is better than in the WAN. This is due to Fat-tree's excellent load balancing property. The performance of our algorithms is better than the other two. In the update steps, the difference between NUSL and Dionysus is smaller because the data center is able to achieve a relatively faster update as a result of its almost fixed path length. In Fig. 7c, there is actually little difference among these three approaches. NUSL less frequently uses spare paths under the Fat-tree topology than in WAN because this topology always distributes in a traffic more balanced way and we can update the network consistently without the help of spare paths when the traffic is light. Moreover, when the traffic is heavy, it is almost impossible to find spare paths under the balancing network. However, the ratio of the number of flows that need to be migrated to their backup paths is also much smaller, which indirectly proves the advantage of a regular topology. The ability to predict the performance in the data center is essential for providing a satisfying service. Consequently, it is worth applying the Fat-tree topology in data centers.

Fig. 8 shows that there are comparatively fewer deadlocks in the data centers because of the regular topology and balanced traffic, which proves the sensibility of applying a regular topology in the data centers. The number of flows involved in the deadlocks is greater than WAN's 30%. This is because if one link is busy, it will affect a large number of flows that would compete for the bandwidth resources. The largest number a single flow involved in the deadlocks is smaller than the WAN topology because it is harder to be intertwined with other flows that are routed in a tidy pattern. It is also more likely to find a feasible update plan with the Fat-tree topology. With the same load ratio, the probability is about 24% higher. Thus, NUSL is extremely suitable for the patterned topology. Fat-tree is better than WAN in terms of deadlocks.

D. Spare bandwidth resources cost comparison

We also study the spare link resource utilization condition under those two topologies in Tab. I. For both of the topologies, it is intuitively easy to find spare resources within the light traffic network. As traffic becomes heavier, it becomes more and more difficult to find a spare path for a flow, and a helper path may not even exist. From the table, we can see that the possibility of finding a spare path decreases significantly by more than $\frac{3}{4}$ as the traffic load ratio increases from 30% to 90%. This explains the reason that NUSL needs to limit much more flows and suffer more severe losses under heavy traffic than light traffic. Moreover, in the spare resource cost rows of the table, we notice that the spare resource usage cost first increases then decreases. This is because we are not able to obtain enough spare paths in the congested network. There is little leisure bandwidth available for us to resolve all the deadlocks. Even if there is one, the cost of the path is so high that the loss outweighs the gain. As a result, it is necessary to limit the length of helper paths in order to control the cost and the achievement. We can attribute this phenomenon to the trade-off between spare resource usage and deadlock resolution possibilities.

A comparison between the WAN and Fat-tree topologies is shown in Tab. I, and it is usually more likely to find a spare path in the Fat-tree infrastructure. Moreover, because all paths

TABLE I: Spare link resource usage

Topology setting	Comparison metrics	The given traffic load			
		φ=0.3	φ=0.5	φ=0.7	φ=0.9
WAN topology	Flow with int. state	84%	23%	17%	5%
	Spare resource cost	9	37	87	52
Fat-tree topology	Flow with int. state	79%	45%	22%	13%
	Spare resource cost	11	21	40	31

have fixed lengths in the Fat-tree, the cost of each flow with intermediate state ρ is either 2 or 5. In data centers, most of the traffic is between different pods [20]. It costs less to use the spare resources, because the Fat-tree topology naturally limits the forwarding path length. The fixed path length also reduces the number of the available spare paths from an exponential to a polynomial. This implies that it is better for the data center to apply the spare-path assisted strategy.

In conclusion, there are two main reasons for the above results. First, the Fat-tree topology is more regular than the WAN topology. It can be divided into three kinds of switches: core, aggregation, and edge. Aggregation and edge switches can serve as a pod, shown in the Fig. 4b, the dotted line square. All the forwarding paths between servers in different pods follow the same pattern: edge-aggregation-core-aggregation-core, whose length is 5. If the servers are in the same pod but have different edge switches, the pattern is edge-aggregation-edge. The forwarding paths between two servers belong to the same edge switch pass through this switch. With such forwarding paths, the workload is able to be distributed more evenly, which reduces the possibility of deadlocks. Second, the Fat-tree topology has a hierarchical structure, but there is no such specific construction for the WAN topology. The bandwidths in Fat-tree vary from the rank where the links reside. Upper links with more traffic have larger bandwidths. However, all links in the WAN have almost the same capacities. Unbalanced flow distribution is more likely to be blocked in WANs.

VI. Conclusion

This paper focuses on the network update problem in the background of SDNs. When network administrators reallocate bandwidth resources without taking flow path information into consideration, deadlocks among flows and link resources unavoidably block the update process and cause congestion as well as deadlocks. While current methods neglect deadlocks, we come up with an efficient approach to consistently update networks with the help of spare paths. We utilize a resource dependency graph to describe the relationships among different flow states and link resources. An algorithm is proposed to determine the feasibility of consistent flow migrations. We demonstrate that it is NP-hard to find the optimal scheme using the fewest spare link resources, even when there are several consistent update plans. An efficient algorithm, NUSL, is proposed to achieve a reasonably good competitive ratio. Our algorithms are evaluated in various aspects under different network scenarios. The evaluation results demonstrate the effectiveness and efficiency of our approach.

VII. Acknowledgment

This research was supported in part by NSF grants CNS1629746, CNS 1564128, CNS 1449860, CNS 1461932, CNS 1460971, CNS 1439672, CNS 1301774, and ECCS 1231461.

References

[1] D. Katabi, M. Handley, and C. Rohrs, "Congestion control for high bandwidth-delay product networks," *SIGCOMM Computer Communication Review*, vol. 32, no. 4, pp. 89–102, 2002.

[2] M. Canini, P. Kuznetsov, D. Levin, and S. Schmid, "A distributed and robust sdn control plane for transactional network updates," in *IEEE INFOCOM*, 2015.

[3] Y. Zhu, N. Kang, J. Cao, A. Greenberg, G. Lu, R. Mahajan, D. Maltz, L. Yuan, M. Zhang, B. Zhao, and H. Zheng, "Packet-level telemetry in large datacenter networks," in *ACM SIGCOMM*, 2015.

[4] T. Mizrahi and Y. Moses, "Time4: Time for SDN," *IEEE Transactions on Network and Service Management*, vol. 13, no. 3, pp. 433–446, 2016.

[5] C. Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer, "Achieving high utilization with software-driven wan," in *ACM SIGCOMM*, 2014.

[6] 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," *SIGCOMM Computer Communication Review*, vol. 43, no. 4, 2013.

[7] S. Raza, Y. Zhu, and C.-N. Chuah, "Graceful network state migrations," *IEEE/ACM Transaction Network*, vol. 19, no. 4, pp. 1097–1110, 2011.

[8] K. Bu, X. Wen, B. Yang, Y. Chen, L. E. Li, and X. Chen, "Is every flow on the right track? Inspect SDN forwarding with RuleScope," in *IEEE INFOCOM*, 2016.

[9] M. Reitblatt, N. Foster, J. Rexford, C. Schlesinger, and D. Walker, "Abstractions for network update," in *ACM SIGCOMM*, 2012.

[10] H. Zheng, W. Chang, and J. Wu, "Coverage and distinguishability requirements for traffic flow monitoring systems," in *IEEE IWQoS*, 2016.

[11] A. Ludwig, M. Rost, D. Foucard, and S. Schmid, "Good network updates for bad packets: Waypoint enforcement beyond destination-based routing policies," in *ACM HotSDN*, 2014.

[12] H. H. Liu, X. Wu, M. Zhang, L. Yuan, R. Wattenhofer, and D. Maltz, "zupdate: Updating data center networks with zero loss," *SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 411–422, 2013.

[13] S. Tseng, C. Lim, N. Wu, and A. Tang, "Time-aware congestion-free routing reconfiguration," in *IFIP NETWORKING*, 2016.

[14] X. Jin, H. H. Liu, R. Gandhi, S. Kandula, R. Mahajan, M. Zhang, J. Rexford, and R. Wattenhofer, "Dynamic scheduling of network updates," in *ACM SIGCOMM*, 2014.

[15] S. Brandt, K. T. Frster, and R. Wattenhofer, "On consistent migration of flows in sdns," in *IEEE INFOCOM*, 2016.

[16] U. Feige, "A threshold of ln n for approximating set cover," *Journal of the ACM*, vol. 45, no. 4, pp. 634–652, 1998.

[17] D. P. Williamson and D. B. Shmoys, *The Design of Approximation Algorithms*, 2011.

[18] Y. Chen and J. Wu, "Max progressive network update," in *IEEE ICC*, 2017.

[19] S. Kandula, I. Menache, R. Schwartz, and S. R. Babbula, "Calendaring for wide area networks," in *ACM SIGCOMM*, 2014.

[20] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *ACM SIGCOMM*, 2014.