# Reroute Backward to Better Break Deadlocks

Lorenzo Maggi, Pierre-Louis Poirion, Jérémie Leguay
Mathematical and Algorithmic Sciences Lab, France Research Center,
Huawei Technologies France, 92100 Boulogne-Billancourt, France
Emails: firstname.lastname@huawei.com

***Abstract*—While rerouting existing demands towards a (better) target configuration, it is sometimes unavoidable to end up in an intermediary *deadlock* state, where some demands need to be rate-limited to be able to successfully reroute all demands to their new path. This is the classic Deadlock Break Problem (DBP), known to be NP-hard. We propose to simplify DBP by first selecting a subset of demands such that, once they have been rerouted via rate-limiting, then all demand rates can be restored to their original value and the normal rerouting process can resume. We call this sub-problem Deadlock Jump Problem (DJP), which is to be solved prior to DBP. We prove that DJP is NP-hard, but we provide an efficient heuristic for it, consisting in rerouting *backward* the demands in deadlock, i.e., from their target to their initial path. Next, DBP can focus only on the demands that have not been successfully rerouted backward. We finally show by experiments that our approach can significantly boost the performance of State-of-the-Art algorithms for DBP, in terms of throughput gain, number of rate-limited demands and number of rerouting epochs.**

***Keywords*—congestion-free rerouting; deadlocks; backward rerouting; rate-limit**

## I. Introduction

As the network conditions change, routing is to be modified so as to dynamically optimize performance. Network changes are of different nature: new (old) connection request (dis)appear, existing demands modify their connection requirements, or the network encounters a failure.

The advent of centrally managed Software Defined Networks (SDN) has recently fostered the implementation of network re-optimization algorithms. However, migrating the current network routing to the new optimized state is still a non-trivial task. In fact, switching all demands to their new path at the same time is not a practical solution to this problem, as data plane updates may fall behind the control plane and the latency is variable across the switches, see [10] and [4]. This may cause transient inconsistent routing policies during the migration process, which are essentially of three different natures [3]: link congestion, black-holes, and loops.

In this paper we focus on the first one, hence looking for congestion-free updates computed in a centralized manner. In this scenario, the rerouting procedure is typically divided into several successive epochs. At each epoch, only a subset of demands is selected for rerouting, such that link congestion is prevented for *any* asynchronous behavior of switches, i.e., for *any* order of update of the demands. The congestion-free property of being robust to asynchronous updates can be ensured via the well-known Make-Before-Break (MBB) mechanism [2]: if a demand is rerouted from its old to its new path at epoch $t$, then the old connection is still kept active during epoch $t$. Such operational constraint makes the congestion-free rerouting problem NP-hard [6]. Hence, researchers recently proposed several heuristics, that can be classified into two main categories, depending on whether flows can be (*unevenly*) split or not between their old and their new, optimized path. In the practice, most of the time, flows can only be split using the classic Equal-Cost Multi-Path (ECMP). On the other hand, the implementation of more general uneven flow splitting has not yet reached a consensus and it is still under study. E.g., hash-based routing rules have been proposed, under the constraint of limited memory at the switch side [11]. In this paper we consider the unsplittable case, first studied in MPLS networks in [7],[9] and then investigated in SDN notably in [6],[1]. In the unsplittable case, during the rerouting process it may happen (in fact, it is unavoidable when the network is highly loaded, see [8]) to end up in a *deadlock* state. A deadlock state is a routing configuration where only a subset of demands are instantiated on their new path, while none of the remaining demands can be individually rerouted without incurring link congestion. In order to break a deadlock and be able to safely reroute all demands to their target new paths, the rate of some demands needs then to be temporarily limited, in order to let the rerouting process resume. Yet, one should not abuse rate-limiting, in order to minimize throughput loss during the rerouting process. The Deadlock Break Problem (DBP) of solving a deadlock situation to reroute all demands is also known to be NP-hard, hence heuristic approaches are proposed in the literature, e.g. [6],[8].

In this paper we propose a method called `UNLOCK` to reduce the dimensionality of DBP, that is able to boost the performance of any heuristic designed for breaking deadlocks via rate-limiting. We formalize it as the solution to the so-called Deadlock Jump Problem (DJP) computing the minimum number of demands that should be simultaneously swapped to their new path to break the deadlock, and let the normal rerouting process resume with restored demand rates.

| | |
|---|---|
| $\mathcal{E}$ | : set of directed edges |
| $\mathcal{K}$ | : demands in deadlock |
| $\mathcal{D}$ | : set of deadlock states |
| $\mathcal{O}_k$ ($\mathcal{N}_k$) | : old (new) path for demand $k$ |
| $d_k$ | : (original: before rate-limiting) size of demand $k$ |
| $b_e$ | : available capacity of edge $e$ |
| $T$ | : n. of epochs used to break the deadlock |
| $T_{BKW}$ | : n. of epochs for Backward Rerouting |
| DBP | : Deadlock Break Problem |
| DJP | : Deadlock Jump Problem, proposed to simplify DBP |
| MBB | : Make-Before-Break |

TABLE I: Table of notation symbols

## II. Deadlock Break Problem

We model the network as a directed graph, where $\mathcal{E}$ is the set of edges. A number of MPLS tunnels (also called *demands*) is instantiated in the network: each demand $k$ is routed on its (old) path $\mathcal{O}_k$, with reserved bandwidth $d_k$.

At a certain time, the network controller decides to re-optimize the network and computes a new target state which is characterized by a set of (new) paths $\{\mathcal{N}_k\}_k$ for each demand $k$. As explained in the Introduction, this decision may respond to a network failure, to the modification of the set of active connection requests, or else to the variation of requirements for existing demands. The new target state $\{\mathcal{N}_k\}_k$ may be more profitable than the old configuration in terms of, e.g., overall routing cost, Quality of Service or link utilization, as in [5].

Once the target state has been computed, each demand $k$ needs to be rerouted from its old path $\mathcal{O}_k$ to its new path $\mathcal{N}_k$ in a congestion-free manner, which is also robust to asynchronous switch updates. Classically, this is achieved via the Make-Before-Break (MBB) rerouting mechanism [2], according to which a new path is instantiated before the old one is terminated. MBB is an efficient congestion-free mechanism, that overbooks network resources in order to ensure that no traffic loss has to be endured during migration. It may happen (in fact, it is sometimes unavoidable [9]) that during network re-optimization a *deadlock* situation is reached, where the rerouting process cannot continue in a congestion-free manner. In other words, no single demand can be rerouted without violating capacity constraints.
Let us now define more formally the concept of deadlock.

**Deadlock**. Assume that a subset of the whole set of demands has already been rerouted and is on their new path, while the remaining demands $\mathcal{K}$ are still on their old path. Let $b_e$ be the available capacity on edge $e$. Then, we say that demands $\mathcal{K}$ are in *deadlock* whenever *no single demand can be rerouted to its new path*, i.e.,

$$\forall \bar{k} \in \mathcal{K}, \quad \exists\, e \in \mathcal{N}_k \setminus \mathcal{O}_k : \; d_{\bar{k}} + \sum_{k \in \mathcal{K}: e \in \mathcal{O}_k} d_k > b_e .$$

□

Let us clarify the concept of deadlock through an example, that we will revisit in Section III-A to illustrate our approach to efficiently solve deadlocks.

**Example 1 [deadlock]** (see also Fig. 1). Let us consider a graph with 5 edges, $e_1, \dots, e_5$. Edges $e_3, e_4$ have capacity $b_3 = b_4 = 2B$, while all other edges $e_1, e_2, e_5$ have capacity $B$. We consider three classes of demands $(\mathcal{K}^+, \mathcal{K}^-, \bar{k})$. We call $\mathcal{K}^+$ the demands having $[e_1, e_3]$ as old path and $[e_2, e_4]$ as new path, and total size $\sum_{k \in K^+} d_k = B$. Then, we call $\mathcal{K}^-$ the demands with $[e_2, e_3]$ as old path and $[e_1, e_4]$ as new path, with the same total size $\sum_{k \in K^-} d_k = B$. Finally, we consider a single demand $\bar{k}$ having $e_5$ as old path, $e_4$ as new path and with (small) size $\bar{d} = \min_{k \in \mathcal{K}^+ \cup \mathcal{K}^-} d_k / 2$. Then, the initial configuration, where each demand is on its old path, is a deadlock state. In fact, none of the demands can be rerouted to its new path without violating capacity constraints on edge $e_1$ (for demands $\mathcal{K}^-$), on edge $e_2$ (for demands $\mathcal{K}^+$) or else on edge $e_3$ (for demand $\bar{k}$). □

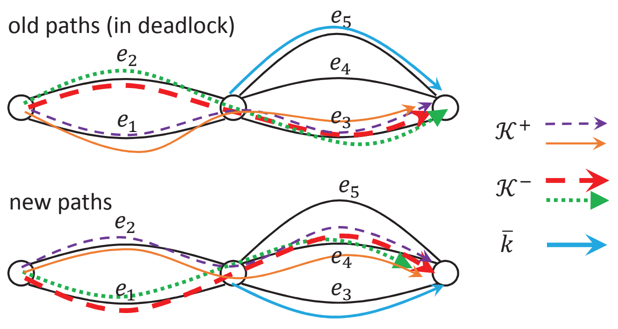

Fig. 1: Illustration of Example 1 with 5 demands, and $|\mathcal{K}^+| = |\mathcal{K}^-| = 2$.

In this paper we work under the (pragmatic, see the Introduction) assumption that demands are unsplittable between old and new path. In this scenario, a deadlock state can be broken and all demands can be successfully rerouted to their new path by *temporarily limiting the rate of certain demands*, as in [6],[8]. At the same time, the temporary rate reduction should be minimized or, equivalently, the total throughput achieved during rerouting should be maximized.
We remark that allowing the traffic to be routed on intermediate paths can help reducing the throughput loss caused by rate-limiting, as in [5]. On the other hand, this would increase the complexity of DBP as the number of constraint becomes exponential. Moreover, instantiating intermediate paths is time-consuming. For simplicity of analysis we will assume that intermediate paths cannot be used, even though our approach described in Section III can be easily adapted to this scenario.

The program that breaks a deadlock with the minimum throughput loss is dubbed Deadlock Break Problem (DBP), and we formalize it below.

**Deadlock Break Problem** (DBP)**.** *Let $\mathcal{K}$ be the set of demands in deadlock. Assume that $T \ge 2$ rerouting epochs are used to break the deadlock. Then, reroute all demands $\mathcal{K}$ via rate-limit by solving the following program*[1]*:*

$$\max_{d_k^t, \mathcal{P}} \sum_{t=1}^{T} \sum_{k \in \mathcal{K}} d_k^t \tag{1}$$

$$\text{s.t.} \quad \sum_{k: e \in \mathcal{P}_k^{t-1} \cup \mathcal{P}_k^t} d_k^t \le b_e, \quad \forall\, e \in \mathcal{E}, t \in [1;T] \tag{2}$$

$$0 \le d_k^t \le d_k, \quad \forall\, k, t \tag{3}$$

$$d_k^0 = d_k^T = d_k, \quad \forall\, k \tag{4}$$

$$\mathcal{P}_k^t = \{\mathcal{O}_k, \mathcal{N}_k\}, \quad \forall\, k, t \tag{5}$$

$$\mathcal{P}_k^0 = \mathcal{O}_k, \; \mathcal{P}_k^T = \mathcal{N}_k, \quad \forall\, k \in \mathcal{K} \tag{6}$$

*where $\mathcal{P}_k^t$ is the path of demand $k$ at epoch $t$ and $d_k^t < d_k$ denotes that the rate of demand $k$ is limited to $d_k^t$ at epoch $t$.*
□

[1]We observe that breaking a deadlock state is always possible, since the program (1-6) admits at least one feasible solution. In fact, it trivially suffices to rate-limit all demands to 0 at epoch $t = 1$ (i.e., $d_k^1 = 0, \; \forall\, k$) while switching all demands to their new path (i.e., $\mathcal{P}_k^1 = \mathcal{N}_k, \; \forall\, k$) and restore the rate of all demands back to their original size at epoch $t = 2$ (i.e., $d_k^2 = d_k, \; \forall\, k$). Clearly, this naive strategy is undesirable since it entails a considerable throughput loss.

The objective function in (1) is throughput maximization over the $T$ epochs, that amounts to minimizing the throughput being lost due to rate-limit. Equation (2) describes the MBB constraint: if demand $k$ is rerouted at time $t$ (thus, $\mathcal{P}^{t-1} = \mathcal{O}_k$ and $\mathcal{P}^t = \mathcal{N}_k$) then a tunnel of size $d_k^t$ is reserved on all edges belonging to the union of old and new path, i.e., $\mathcal{O}_k \cup \mathcal{N}_k$. According to (5), a demand can be routed either on its old path or on its new path.

We observe that, as DBP generalizes the NP-hard unsplittable rerouting problem in [7], then it is also NP-hard.

## A. Previous work

The congestion-free traffic migration problem has been studied extensively in the last few years. It is known to be NP-hard, even in the easiest case where demands cannot be rate-limited and intermediate paths cannot be used [7]. For this reason, several rerouting heuristics have been developed over the last few years (see [3] for a recent survey). The bulk of the existing approaches rely on the computation of an opportunistic metric for each demand, according to which demands are prioritized for rerouting.

Under our assumption that flows cannot be split, the seminal work in [8] prioritizes demands with respect to metrics inspired by discrepancy theory, based on the level of congestion caused by rerouting a demand. A different approach, scheduling rerouting according to an inter-dependence graph, has been taken by [6] and its follow-up [12]. Such graph describes the dependency between rule updates and the resources freed by any rerouting decision. The work in [9] tackles the even harder problem of jointly computing the new target routing state and the rerouting plan, so as to maximize a generic fairness function of the achievable routing state. In the splittable flow scenario, the work in [5] is the first to show how to exploit the free slack capacity to reroute paths via successive rate update. From a more theoretical perspective, [1] decides in polynomial time if consistent migration is possible at all.

While the congestion-free rerouting problem has received considerable attention over the last few years, the sub-problem of solving a deadlock situation has seldom been investigated carefully. The seminal work [8] only mentions that the rerouting algorithm should be run until the target configuration is attained, and then demands should be rate-limited *a posteriori*, to avoid violating capacity constraints at any epoch. A second, more detailed approach illustrated in [6] prescribes to 1) select $k^*$ demands, 2) limit their rate to successfully reroute all of them and 3) finally attempt to resume the standard rerouting process, until a new deadlock state is reached (see Alg. 2).

## B. Main contributions

In this paper we propose an approach to boost the performance of any method designed to solve the Deadlock Break Problem (DBP), being NP-hard. More specifically, we start by simplifying DBP by first solving *Deadlock Jump Problem* (DJP). DJP selects a small subset of demands $\mathcal{K}_J$ (out of the whole set of demands in deadlock $\mathcal{K}$) for resizing, such that once $\mathcal{K}_J$ has been rerouted via rate-limiting then *i)* all demands can be restored to their original size and *ii)* the standard rerouting procedure can resume, *without* the need to resort to rate-limiting.

We then prove in Theorem 1 that DJP itself is a NP-hard problem, but we propose an efficient Backward Rerouting approach to tackle it. Backward Rerouting exploits the intuition that the traffic migration problem has an intrinsic symmetry: *rerouting demands from old to new path via MBB is logically equivalent to rerouting them from new to old path, in reverse time order*. Hence, we first reroute the demands in deadlock in reverse time order, without resorting to rate-limiting. Then, we select $\mathcal{K}_J$ as the set demands that could not be rerouted backward, and which are finally rerouted via a generic `Rate-Limit` heuristic. We dub our overall approach to solve DBP "`UNLOCK`" (Alg. 1).

`UNLOCK` is shown to have several advantages. First, by construction, it reduces the complexity of the main Deadlock Break Problem DBP. Then, it is capable to boost the performance of the State-of-the-Art heuristic to solve DBP in terms of total throughput during rerouting, number of rate-limited demands and number of used rerouting epochs.

## III. Deadlock Jump: A problem reduction for Deadlock Break

Breaking the deadlock by solving DBP optimally is not a viable solution since, as already observed, DBP is NP-hard. For this reason, some heuristics have been proposed in the literature to approximate DBP (see Section II-A).

In this paper we propose a boosting technique for a generic heuristic approximating DBP, which *reduces* the dimensionality of DBP by first tackling what we call *Deadlock Jump Problem* (DJP). DJP reduces the *potential* pool of demands that are to be rate-limited by DBP in order to successfully reroute all demands. More specifically, DJP selects a (small) subset of demands $\mathcal{K}_J \subset \mathcal{K}$ such that, once they have been rerouted via rate-limiting, then all demands can be restored to their original size and the normal rerouting procedure can resume. Finally, the task of effectively rerouting the selected demands $\mathcal{K}_J$ is delegated to a generic rate-limit heuristic, such as the one in [6].

Technically speaking, the Deadlock Jump Problem (DJP) can be formalized as follows.

**Deadlock Jump Problem** (DJP)**.** *Find $x^*$ defined as the solution to the following program:*

$$\min_{x \in \{0,1\}^{\mathcal{K}}} \sum_{k \in \mathcal{K}} x_k \tag{7}$$

$$\text{s.t.} \sum_{k \in \mathcal{K}: e \in \mathcal{N}_k} x_k d_k - \sum_{k \in \mathcal{K}: e \in \mathcal{O}_k} x_k d_k \le b_e, \quad \forall\, e \in \mathcal{E} \tag{8}$$

$$\mathcal{K} \setminus \{k : x_k = 1\} \notin \mathcal{D} \tag{9}$$

*where $\mathcal{D}$ is the set of deadlock states. Then, select the demands to be jumped as $\mathcal{K}_J = \{k : x_k^* = 1\}$.* □

Equations (8),(9) claim that, after demands $\{k : x_k = 1\}$ have been rerouted, then they can all be restored to their original size and the network is no longer in deadlock, respectively.

By construction, the first clear advantage of DJP is reducing the complexity of DBP. Moreover, as it will become apparent in Section IV, it also allows to boost the performance of any heuristic designed for DBP, in terms of throughput, and number of rate-limited demands and number of used epochs.

Unfortunately, the problem reduction DJP itself is NP-hard, as we show below.

**Theorem 1.** DJP *is* NP*-hard.*

We defer the proof of Theorem 1 to the Appendix. It relies on a polynomial reduction from a modified version of the classic subset problem, that given a set of numbers asks whether there exists a subset of them summing up to zero.

On the other hand, as a positive result, we next propose a Backward Rerouting approach that jointly allows to heuristically approximate DJP and, as a by-product, to compute the rerouting schedule after the deadlock has been broken.

## A. UNLOCK*: Break Deadlocks via Backward Rerouting*

After introducing Deadlock Jump Problem (DJP) as a method to reduce and simplify the NP-hard Deadlock Break Problem (DBP) and having proved that DJP is also NP-hard, we now develop a heuristic for DJP.

We build upon the intuition that the rerouting problem has an inherent symmetric structure: the classic, *forward* problem of rerouting demands from an initial (old) configuration to a target (new) one is essentially equivalent to the respective *backward* problem, where new and old paths are swapped and time order is reversed. We can then apply this concept to our scenario where demands $\mathcal{K}$ are in deadlock. By definition, none of the demands $\mathcal{K}$ can be rerouted forward. Yet, some demands may still be – virtually – rerouted *backward* (see Figure 2). In other words, we bet on the fact that *the target state*, where all demands are on their new path, *is not in deadlock if we reverse the time order*.

We can now exploit this intuition to build a straightforward but effective heuristic for DJP. We first (artificially) reroute *backward* the demands in deadlock $\mathcal{K}$ over a – possibly fixed *a priori* – number of epochs $T_{BKW}$. This can be achieved via any rerouting heuristic from the literature, e.g., [6],[7],[8],[12], with the trick of *swapping the roles of new and old paths*. We highlight that Backward Rerouting is performed without resorting to rate-limiting.

Then, we can finally tackle the solution of the main Deadlock Break Problem (DBP). We first select the demands $\mathcal{K}_J$ as the ones that were *not* successfully rerouted by Backward Rerouting. Next, we feed the selected demands $\mathcal{K}_J$ to a generic Rate-Limit routine that approximates the original Deadlock Break Problem DBP, as the one in [6] that we recall in Alg. 2. Remarkably, as a by-product, our approach not only allows to jump a deadlock, but it also outputs at the same time a feasible rerouting schedule *after* the deadlock has been resolved, read in reverse time order.
We call UNLOCK our procedure to break deadlocks, that uses Rate-Limit as sub-routine and that we formalize below in Alg. 1 (see also Fig. 2).

In order to help the reader grasp the idea behind UNLOCK, let us apply it to Example 1, described in Section II and Figure 1.

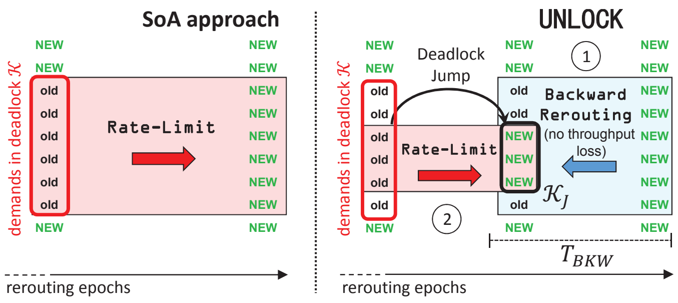

Fig. 2: (**left**) Classic approach to break deadlock via a Rate-Limit procedure, as in [8] and [6]; old/NEW stand for old/new path. To break a deadlock state, all demands can be *possibly* rate-limited to reach the target state. (**right**) UNLOCK prescribes to reduce the number of *potential* demands that can be rate-limited by first Backward Rerouting (without rate-limiting), from the target state back to the deadlock state. Then, a generic Rate-Limit procedure is applied to the remaining demands $\mathcal{K}_J$.

**Algorithm 1:** UNLOCK for the Deadlock Break Problem

```
Data: - Demands in deadlock $\mathcal{K}$;
      - N. of epochs $T_{BKW}$ for Backward Rerouting;
      - Generic routine Reroute(initial paths, target paths,
        $T_{BKW}$) computing congestion-free rerouting sequence
        $\mathcal{R}_1, \dots, \mathcal{R}_{T_{BKW}}$ without rate-limiting over the $T_{BKW}$
        epochs (as, e.g., [6],[7],[8],[12]);
      - Generic routine Rate-Limit to reroute demands in
        deadlock by limiting their rate (as, e.g., Alg. 2 [6])
1 Solve Deadlock Jump Problem (DJP) via Backward
  Rerouting:
  Compute the backward rerouting sequence for the demands in
  deadlock $\mathcal{K}$:
  $\mathcal{R}_1, \dots, \mathcal{R}_{T_{BKW}}$ = Reroute($\{\mathcal{N}_k\}_{k\in\mathcal{K}}$, $\{\mathcal{O}_k\}_{k\in\mathcal{K}}$, $T_{BKW}$)
  (Observe that old and new paths are swapped)
2 Compute the set $\mathcal{K}_J$ of demands that could not be rerouted
  backward: $\mathcal{K}_J = \mathcal{K} \setminus \cup_t \mathcal{R}_t$
3 Solve a smaller Deadlock Break Problem (DBP):
  Call Rate-Limit routine on the reduced set of demands $\mathcal{K}_J$
4 Reroute rate-limited demands $\mathcal{K}_J$ as computed in step 3
5 Restore each demand $k \in \mathcal{K}_J$ to its nominal rate $d_k$
6 Reverse the time order of the the rerouting sequence $\mathcal{R}$
  computed in step 1, i.e., $\mathcal{R}_t \leftarrow \mathcal{R}_{T_{BKW}-t+1}$, $\forall\, t$
7 Reroute all remaining demands $\mathcal{K} \setminus \mathcal{K}_J$ according to $\mathcal{R}$ over
  the last $T_{BKW}$ epochs
8 END: All demands are successfully rerouted.
```

**Example 1 (continued)**. As already observed, all demands $\mathcal{K} = \mathcal{K}^+ \cup \mathcal{K}^- \cup \bar{k}$ are in deadlock. We observe that the only way to jump the deadlock is to jointly move to their new path a subset of demands $\widetilde{\mathcal{K}}^+ \subset \mathcal{K}^+$ and a subset of demands $\widetilde{\mathcal{K}}^- \subset \mathcal{K}^-$. In this way, at least demand $\bar{k}$ can be safely rerouted. We notice that, since no slack capacity is available at edges $e_1$ and $e_2$, we must impose the constraint that *the total size* of demands $\widetilde{\mathcal{K}}^+$ and $\widetilde{\mathcal{K}}^-$ *is equal*. This requires to solve a form of subset sum problem, that given a set of numbers (in our case, the size of demands) asks whether there exists a subset of them having equal sum. Unfortunately, this is a NP-hard problem, that we do not want to tackle. For further details, please refer to the proof of Theorem 1 in the Appendix.

A less ambitious goal would be to just *identify* the whole bundle of "hard" demands $\mathcal{K}^+ \cup \mathcal{K}^-$ and jump the deadlock by simply setting $\mathcal{K}_J = \mathcal{K}^+ \cup \mathcal{K}^-$. In such a way, we *avoid* tackling the NP-hard subset problem, while still managing to reduce the Deadlock Break Problem DBP by one unit, as the demand $\bar{k} \notin \mathcal{K}_J$. In fact, once the bundle of demands $\mathcal{K}_J = \mathcal{K}^+ \cup \mathcal{K}^-$ has been jumped, then there exists enough capacity on edge $e_3$ to reroute demand $\overline{k}$ to its new path.

In order to identify the bundle of "hard" demands $\mathcal{K}^+ \cup \mathcal{K}^-$ we can apply Backward Rerouting. Thus, demand $\overline{k}$ is rerouted backward from its new path to its old path, since in the the target state there is free capacity on edge $e_5$. Then, a new (backward) deadlock state is reached; Backward Rerouting terminates after $T_{BKW} = 1$ epoch and it outputs the demands $\mathcal{K}_J = \mathcal{K}^+ \cup \mathcal{K}^-$, which according to `UNLOCK` are then fed to a `Rate-Limit` heuristic for the original problem DBP. □

**Algorithm 2:** `Rate-Limit`, as in [6]

```
Data: - Number $k^*$ of demands to be rate-limited;
      - Demands in deadlock $\mathcal{K}$;
      - Rerouting (without rate-limiting) routine Reroute (as,
        e.g., [6],[7],[8],[12])
1 Select demands $\mathcal{K}^*$ ($|\mathcal{K}^*| = k^*$) among the $|\mathcal{K}|$ demands in
  deadlock
2 For each $k \in \mathcal{K}^*$, limit the rate of demand $k$ to the maximum
  $d'_k$ such that demand $k$ can be rerouted, i.e.:
  $d'_k = \min(d_k, \min_{e \in \mathcal{N}_k \setminus \mathcal{O}_k}$ left capacity on $e)$
3 Reroute demands $\mathcal{K}^*$ over a single epoch
4 Resume the standard rerouting procedure over the remaining
  demands $\mathcal{K} \setminus \mathcal{K}^*$ via sub-routine Reroute, until a new
  deadlock is reached, and iterate.
```

## IV. Numerical results

We now demonstrate numerically that `UNLOCK` (Alg. 1) outperforms SoA `Rate-Limit` heuristic (Alg. 2) in solving the original Deadlock Break Problem (DBP). We recall that `UNLOCK` can be actually considered as a *boosting* method for any rate-limiting procedures, as it employs `Rate-Limit` as a sub-routine (see line 3, Alg. 1). More specifically, `UNLOCK` prescribes to first reduce the NP-hard problem DBP via Backward Rerouting, and then to feed `Rate-Limit` with the simpler resulting problem.

Before commenting upon our results, we provide two simple intuitions on the reason why our `UNLOCK` approach is able to boost the performance of `Rate-Limit`.

**Intuition 1.** Via Backward Rerouting, `UNLOCK` ensures the existence of a (final) phase where no demand is rate limited. This clearly decreases the throughput loss during rerouting. □

**Intuition 2.** By shrinking the potential number of demands that can be potentially rate-limited, `UNLOCK` feeds `Rate-Limit` with a smaller, and essentially *simpler*, problem. This helps `Rate-Limit` reducing the number of demands that are rate-limited unnecessarily, which also reduces the overhead traffic required for handling rate limiting. □

We now back up our intuitions via numerical experiments. For the sake of fairness, in our simulations `UNLOCK` uses the very same `Rate-Limit` algorithm (Alg. 2) as a sub-routine to reroute the demands $\mathcal{K}_J$. For completeness, we still have to specify the `Reroute` routine to reroute demands without rate-limiting, used in `UNLOCK` (line 1) and in `Rate-Limit` (line 4). Demands are first sorted in decreasing order of rate. Then, there is an attempt to reroute all demands via MBB, in sorted order. The same procedure iterates over the demands not yet rerouted, until either all demands are rerouted, or a deadlock or else the maximum number of epochs is attained. We point out that any algorithm among [6],[7],[8],[12] can replace the `Reroute` routine we used, but we do not insist on this point as `Reroute` showed little impact on our simulation results, and moreover it is not the focus of this paper.

We considered large size topologies, with $> 10^4$ demands, $4.10^4$ links and $10^4$ nodes. In order to test our approach under stress, we considered highly intricate rerouting scenarios: for all topologies, the inter-dependency graph as defined in [6] contains one single connected component at a deadlock state.

We then compare `UNLOCK` (Alg. 1) against State-of-the-Art `Rate-Limit` routine (Alg. 2) in terms of

a) total throughput during rerouting: $= \sum_{t=1}^{T} \sum_{k \in \mathcal{K}} d_k^t$;
b) number of rate-limited demands:
   $= |\{k : d_k^t < d_k \text{ for some epoch } t\}|$;
c) total number $T$ of epochs used to reroute all demands.

We remark that b) has a direct impact on the amount of signaling traffic required to handle rate-limiting, while c) measures the time required to attain the target routing configuration. We show in Fig. 3.a,b,c) the boosting factor of `UNLOCK` with respect to `Rate-Limit` for the three aforementioned metrics, respectively, versus the number of rerouting epochs $T_{BKW}$ allocated to Backward Rerouting. As a complement, in Fig.3.d) we present the proportion of demands rerouted backward, and hence not treated by the `Rate-Limit` heuristic.

Clearly, the number of demands rerouted backward increases monotonically in $T_{BKW}$, see Fig.3.d). As Intuitions 1 and 2 suggested, this has a beneficial impact on the performance boost of `UNLOCK` with respect to a pure `Rate-Limit` procedure for metrics a,b), see Fig.3.a,b). On the other hand, Fig.3.c) shows that allocating too many epochs to Backward Rerouting procedure may lead to an overall inefficient use of rerouting epochs $T$. This stems from the diminishing marginal return of Backward Rerouting, whose ability to reroute demands vanishes when $T_{BKW}$ increases, as shown in Fig.3.d).

## V. Conclusions

While rerouting the network traffic towards a better configuration, it is sometimes unavoidable to end up in a deadlock state, where demands must be rate-limited in order to attain the target configuration. In this paper we exploit the inherent time symmetry of the congestion-free rerouting problem to break the deadlocks in a more efficient manner, via Backward Rerouting. We show that our approach to break deadlocks, called `UNLOCK`, allows to boost the performance of State-of-the-Art `Rate-Limit` heuristics. In fact, by restricting the number of demands on which `Rate-Limit` can operate, `UNLOCK` also naturally reduces the number of myopic rate-limiting decisions taken by `Rate-Limit` itself. Moreover, it ensures via Backward Rerouting the presence of a final phase where all demands are restored to their original size. This allows to reduce the throughput loss incurred during rerouting, the number of rate-limited demands and, as a by-product, the time required to effectively attain the target routing state.

We envision that the concept of Backward Rerouting can be further generalized, and be successfully exploited for solving the rerouting problem on its own. We believe that alternating between forward and backward rerouting may allow to further reduce the number of demands in deadlock.

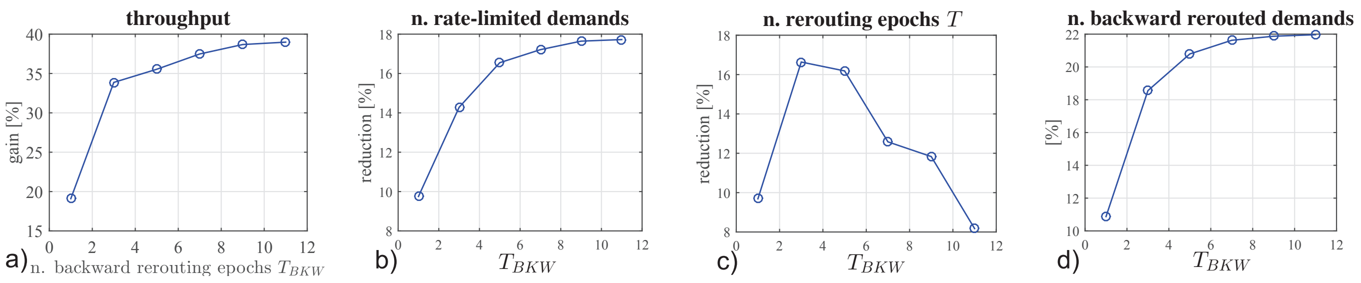

Fig. 3: Performance boost of UNLOCK w.r.t. Rate-Limit ([6], Alg. 2), both solving the Deadlock Break Problem (DBP). The performance metrics are a) throughput ($=\sum_{t=1}^{T}\sum_{k\in\mathcal{K}} d_k^t$), b) number of rate-limited demands ($=|\{k: d_k^t < d_k \text{ for some } t\}|$) and c) number of epochs $T$ used to reroute all the demands in deadlock $\mathcal{K}$. In d) we show the corresponding problem reduction, i.e., the number of demands $|\mathcal{K}_J|$ successfully rerouted by Backward Rerouting w.r.t. the total number of demands in deadlock $|\mathcal{K}|$. On the x-axis, the number of epochs $T_{BKW}$ used by Backward Rerouting sub-routine in UNLOCK (Alg. 1, line 1).

## Appendix

### A. Proof of Theorem 1

We will prove the NP-completeness of the decision problem associated to DJP, called Deadlock Jump Decision Problem (DJDP), aiming at selecting a subset of the demands in deadlock $\mathcal{K}' \subset \mathcal{K}$ of size smaller than $h > 0$ such that allocating $\mathcal{K}'$ to their new path allows to break the deadlock. Before proving the NP-completeness of DJDP, we first introduce an auxiliary problem, being a variation of the classic subset sum problem and that we prove to be NP-complete.

**Modified Subset Sum Problem** (MSSP)**.** *Given a set of numbers $\mathcal{I} = \{i_1, \dots, i_n\}$ summing up to 0, is there a strict subset of $\mathcal{I}$ also summing up to 0?*

We start by proving that MSSP is NP-complete. Clearly, MSSP is in NP. We prove the thesis via a polynomial reduction of from the classic Subset Sum Problem (SSP), that we here recall: "Given a set of numbers $\mathcal{I}'$, does there exist a subset of $\mathcal{I}'$ summing up to 0?". Given $\mathcal{I}'$, set $a = -\sum_{i\in\mathcal{I}'}$ and construct the instance $\mathcal{I} = \mathcal{I}' \cup \{a\}$, which can be clearly done in polynomial time. $\mathcal{I}$ sums up to 0 and it represents a suitable instance for MSSP. If SSP answers YES on the instance $\mathcal{I}'$ then there exists $\mathcal{I}'' \subseteq \mathcal{I}' \subset \mathcal{I}$ that sums up to 0, then MSSP also answers YES on $\mathcal{I}$. Conversely, if MSSP answers YES on $\mathcal{I}$ then there exists $\mathcal{I}'' \subset \mathcal{I}$ summing up to 0. Notice that its complement $\overline{\mathcal{I}''}$ also sums up to 0, as $\mathcal{I}$ itself sums up to 0 by construction. Then, if $a \in \mathcal{I}''$ then $\overline{\mathcal{I}''} \subseteq \mathcal{I}'$ is a YES instance for SSP; else, if $a \in \overline{\mathcal{I}''}$ then $\mathcal{I}'' \subseteq \mathcal{I}'$ is a YES instance for SSP. In either case, SSP answers YES on $\mathcal{I}'$. Hence, MSSP is NP-complete.

Now we are finally ready to prove Theorem 1. Given a certificate $\mathcal{K}' \subseteq$ we can verify in polynomial time whether equations (9) hold by setting $x_k = \mathbb{I}(k \in \mathcal{K}')$. Then, DJDP is in NP. We will prove the NP-completeness of DJDP via a polynomial reduction from the modified subset sum problem (MSSP), being NP-complete as we previously proved. Let $\mathcal{I}$ be an instance of MSSP, and we define for convenience $\mathcal{I}^- = \{i \in \mathcal{I} : i < 0\}$, $\mathcal{I}^+ = \{i \in \mathcal{I} : i > 0\}$ and $B = \sum_{i\in\mathcal{I}^+} i = -\sum_{i\in\mathcal{I}^-} i$. To conceive a suitable instance of DJDP, we refer to Example 1 (see also Fig. 1). We say that demands $\mathcal{K}^+$ have size $\{d_k\}_{k\in K^+} = \mathcal{I}^+$. Similarly, demands $\mathcal{K}^-$ have size $\{d_k\}_{k\in K^-} = -\mathcal{I}^-$.

Let $f : \mathcal{I} \to \mathcal{K}$ be a mapping between the instances of the two problems. We also observe that this instance of DJDP can be constructed in polynomial time. Next, we set $h = |\mathcal{K}| - 2$ and we show that solving MSSP on $\mathcal{I}$ is equivalent to solving DJDP on $f(\mathcal{I})$. If there exists $\widetilde{\mathcal{I}} \subset \mathcal{I}$ summing up to 0 then the demands $f(\widetilde{\mathcal{I}}) \cap \mathcal{K}^+$ and $f(\widetilde{\mathcal{I}}) \cap \mathcal{K}^-$ have the same total size, hence they can switch from their old to their new path while fulfilling the capacity constraints. Thus, enough capacity is freed up on edge $e_3$ for demand $\bar{k}$ to switch to its new path $e_3$. Thus, $\mathcal{K}' = f(\widetilde{\mathcal{I}})$ is not in deadlock, i.e., $\mathcal{K}' \notin \mathcal{D}$. Also, $\mathcal{K}' \leq h$. Then, $\mathcal{K}'$ is a solution to DJDP. Conversely, let $\mathcal{K}'$ be a feasible solution to DJDP. Assume by absurd that $\bar{k} \in \mathcal{K}'$. Then there exists a demand $k \in \mathcal{K}^+$ (or $\mathcal{K}^-$) such that there exists some scratch capacity on $e_2$ (or $e_1$) being at least $d_k$, which implies that the load on $e_1$ or $e_2$ exceeds the capacity. Then, $\bar{k} \notin \mathcal{K}'$. Now, consider $\widetilde{\mathcal{I}}^+ = \{d_k\}_{k\in\mathcal{K}^+\cap\mathcal{K}'}$ and $\widetilde{\mathcal{I}}^- = \{-d_k\}_{k\in\mathcal{K}^-\cap\mathcal{K}'}$. Then, $\widetilde{\mathcal{I}}^+ \cup \widetilde{\mathcal{I}}^- \subseteq \mathcal{I}$ sums up to 0 and MSSP answers YES on the instance $\mathcal{I}$, Q.E.D.. $\square$

## References

[1] S. Brandt, K.-T. Förster, and R. Wattenhofer. On consistent migration of flows in SDNs. In *Computer Communications, IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on*. IEEE, 2016.
[2] E. Crabbe, I. Minei, J. Medved, and R. Varga. PCEP extensions for stateful PCE. *draft-ietf-pce-stateful-pce-18*, 2016.
[3] K.-T. Foerster, S. Schmid, and S. Vissicchio. Survey of consistent network updates. *arXiv preprint arXiv:1609.02305*, 2016.
[4] K. He, J. Khalid, A. Gember-Jacobson, S. Das, C. Prakash, A. Akella, L. E. Li, and M. Thottan. Measuring control plane latency in sdn-enabled switches. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, page 25. ACM, 2015.
[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 Computer Communication Review*, volume 43, pages 15–26. ACM, 2013.
[6] 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 Computer Communication Review*, volume 44, pages 539–550. ACM, 2014.
[7] B. G. Józsa. Reroute sequence planning for protected traffic flows in GMPLS networks. In *Communications, 2002. ICC 2002. IEEE International Conference on*, volume 5, pages 2702–2706. IEEE, 2002.
[8] B. G. Józsa and M. Makai. On the solution of reroute sequence planning problem in MPLS networks. *Computer Networks*, 42(2):199–210, 2003.
[9] O. Klopfenstein. Rerouting tunnels for MPLS network resource optimization. *European Journal of Operational Research*, 188(1):293–312, 2008.
[10] M. Kuźniar, P. Perešíni, and D. Kostić. What you need to know about SDN flow tables. In *International Conference on Passive and Active Network Measurement*, pages 347–359. Springer, 2015.
[11] P. Medagliani, J. Leguay, M. Abdullah, M. Leconte, and S. Paris. Global Optimization for Hash-based Splitting. In *Global Communications Conference (GLOBECOM), 2016 IEEE*. IEEE, 2016.
[12] W. Wang, W. He, J. Su, and Y. Chen. Cupid: Congestion-free consistent data plane update in software defined networks. In *Computer Communications, IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on*, pages 1–9. IEEE, 2016.