On Consistent Migration of Flows in SDNs

Sebastian Brandt
ETH Zurich, Switzerland
brandts@ethz.ch

Klaus-Tycho Foerster*
ETH Zurich, Switzerland
foklaus@ethz.ch

Roger Wattenhofer
ETH Zurich, Switzerland
wattenhofer@ethz.ch

***Abstract*—We study consistent migration of flows, with special focus on software defined networks. Given a current and a desired network flow configuration, we give the first polynomial-time algorithm to decide if a congestion-free migration is possible. However, if all flows must be integer or are unsplittable, this is NP-hard to decide. A similar problem is providing increased bandwidth to an application, while keeping all other flows in the network, but possibly migrating them consistently to other paths. We show that the maximum increase can be approximated arbitrarily well in polynomial time. Current methods as RSVP-TE consider unsplittable flows and remove flows of lesser importance in order to increase bandwidth for an application: We prove that deciding what flows need to be removed is an NP-hard optimization problem with no PTAS possible unless P = NP.**

I. Introduction

In today's wide area networks, traffic demands change all the time. To cope with this, the changing data flows get sent through the network by routing algorithms in a greedy (shortest path) way that favors distributed fault-tolerance over efficiency. While this makes sense in a scenario where the network is controlled by various independent participants (e.g., the Internet), the situation changes when the network is controlled by one entity such as Google [1] or Microsoft [2]. With only one logical central controller, it is possible to actually reclaim the control over the general behavior of the network and just leave the menial task of data forwarding localized in the switches and routers. This is one of the fundamental ideas that gave rise to Software Defined Networks (SDNs) [3], [4], [5], a network paradigm separating control and data plane of a network. Should new data flows appear or old flows change their traffic demands, one can use the established toolkit of multi-commodity flow theory to calculate a new network behavior optimized for efficiency. The new rules can then be distributed to all nodes in the network [1], [2], [6], [7], [8], [9]. The advantages of this method become especially evident when optimizing under-utilized links of high costs, with cable prizes of up to hundreds of millions of dollars [10].

But, while the new network behavior might be optimal, what happens during the migration to the new behavior? Clock synchronization is far from perfect, and even if, some switches will straggle (taking up to $100\times$ longer than average to update in practice [11]) or might not be available to the central controller at all for some time [1]. This inherent asynchrony will lead to over-utilization of links, inducing congestion and thus packet loss. One might just ignore these effects, hoping things will get better after a while, but especially for real-time applications, delays and loss of data are not desirable. Hence, recent development has considered consistent migration [2], [9], [11], [12], i.e., congestion-free and without temporary demand reduction, where the network is migrated according to a precomputed ordering or dynamic network behavior.

However, all these solutions suffer from one central downside: They cannot decide if consistent migration is possible. Similar to the halting problem, a consistent migration for flows might be found, if it exists. But if a consistent migration is not possible, all existing algorithms might be stuck searching indefinitely. All known approaches will resort to breaking consistency by, e.g., dropping flows even if this is not necessary.

We close this gap by showing how to decide fast if a consistent migration is possible (Section V). We also prove that splitting flows into non-integer parts is vital: Else, consistent migration of flows is NP-hard to decide (Section IV).

In a similar line of thought, we investigate the problem of consistently increasing the flow of traffic between a source and a terminal node, while keeping all other traffic flows intact. Current methods such as RSVP-TE consider unsplittable flows and assign weights according to the importance of the flow. Then, less important flows are removed until there is enough capacity, and finally the desired flow is added. We can show that deciding what flows to remove is an NP-hard optimization problem with no PTAS possible unless P = NP (Section VI). If flows are splittable, we show how the maximum traffic increase can be approximated arbitrarily well: We can calculate in polynomial time (independent of the chosen approximation ratio) a viable new flow placement, s.t. i) the demand increase for the desired commodity is within $(1-\epsilon)$ of the maximum, ii) the demand of all other commodities is unchanged, and iii) consistent migration is possible (Section VII).

II. Related Work

Multi-Commodity Flows: While some fundamental results from single-commodity flows no longer hold [13], deciding if a feasible multi-commodity flow exists can be computed in polynomial time as well, e.g., by linear programming. However, in practice, one might want to resort to approximation algorithms due to their much better runtime [14]. We refer to Cormen et al. [15] and Ahuja et al. [16] for a further overview.

Migration of Flows in SDNs: One approach to congestion during migration is to accept it, but to try to minimize its impact. The induced inconsistency is then ignored in general, e.g., data might have to be sent again in case of packet loss. *B4* [1] develops custom hardware and mechanisms to integrate current protocols into SDNs, with the goal of speeding up the migration process. Mizrahi et. al [17], [18] aim at minimizing

*Supported in part by Microsoft Research.

inconsistency with time-based updates. They also show that for optimal traffic engineering, flows need to be swapped.

In specialized environments, as (inter-)datacenter traffic, one can take advantage of a restricted/optimized network topology or the ability to schedule traffic [12], [19], [20], [21].

The case of just one logical source or destination was recently covered in [22] via the method of augmenting flows, allowing to migrate flows in a *linear* number of updates.

Another way to tackle congestion is via game theory (e.g., congestion games [23]) or model checking [24], [25], however the authors of [24], [25] note that properties concerning bandwidth are not yet handled by their work.

The work of Reitblatt et al. [26] introduces a different form of consistency, called *per-packet/flow consistency*: By stamping each packet or flow with a version number, denoting old or new forwarding rules, one can ensure that a packet/flow is always routed according to just one set of rules. While their work prevents many effects that induce congestion, it does not prevent congestion when faced with the task of migrating multiple flows at once. As shown in Subfigure 1b, congestion can occur in such a case. If a mixture of old and new rules is allowed as long as certain policies are respected (e.g., passing firewalls), then the amount of additional memory needed in the switches can be greatly reduced in practice, cf. [27].

Dionysus [11] tries to find a consistent migration ordering by searching through a dependency graph of possible migration steps. In their model, flows are not allowed to be splitted and may only migrate as a whole to the desired path, see the Subfigures 1b,1c. They show the corresponding decision problem to be NP-hard under switch memory constraints. If no solution is found, some flows are rate-limited for congestion-free migration of the remaining flows.

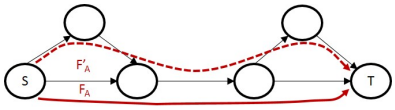
The approach of *SWAN* [2] is twofold: First, if one guarantees that a fraction s of capacity (*slack*) is free on each link for the old and new flow, then one can migrate congestion-free in $\lceil 1/s \rceil - 1$ steps. E.g., if each edge never exceeds 95% capacity in both the current and the desired network state, it takes 19 steps to migrate at most. Second, they use binary search over the number of steps to find the optimal solution, i.e., they check with linear programming if a solution with x steps exists. However, they note that this is costly in practice for a long sequence of LPs. This search also works if there is no slack on some edges, but then the computation is not guaranteed to halt when no consistent migration exists. We refer to the Subfigures 1c,1d for examples.

Our methods extend the work of [2], [11], [12] where the consistent migration of flows in SDNs is implemented and evaluated with multiple production data center networks across three continents with tens of thousands of servers.

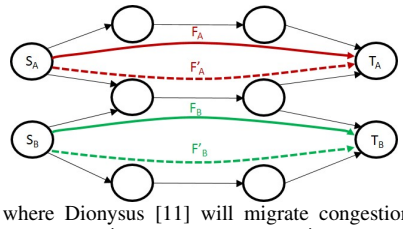
Lastly, for an overview of further abstractions for SDNs, we refer to the recent article of Casado, Foster, and Guha [9].

III. Model

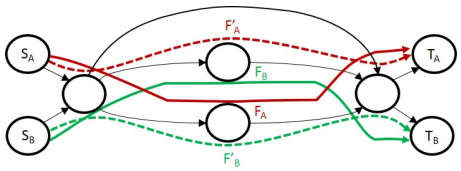
We start by modeling a network as a directed graph, with a flow of a commodity respecting flow conservation, demand satisfaction, and capacity constraints:



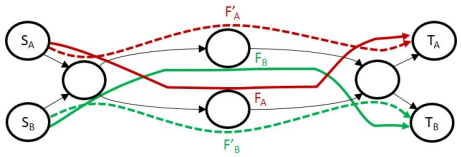
(a) When migrating from F_A to F'_A, the method of packet-stamping [26] will migrate the network in this example congestion-free. This also holds for multiple commodities, as long as only one commodity is considered for migration.



(b) Example where Dionysus [11] will migrate congestion-free by first migrating flow F_B to F'_B and then F_A to F'_A. However, the method from [26] might not migrate congestion-free, if F_A migrates to F'_A before F_B migrates to F'_B due to asynchrony.



(c) Example where SWAN [2] can migrate congestion-free by temporarily storing a flow on the topmost link. Dionysus [11] will not find a solution without rate-limiting one flow to zero.



(d) Example where the method from SWAN [2] will keep computing forever (unless halted somehow manually or by a threshold of steps), since it is not possible to migrate congestion-free without rate-limiting.

Fig. 1. Example networks where the methods of [2], [11], [26] do not succeed, due to violating congestion-freedom, rate-limiting flows, or by not halting their computation. All edges in the four examples have a capacity of one and all flows have a size of one. Initial flows are drawn solid, the new ones dashed.

Definition 3.1: Let $G = (V, E)$, with $|V| = n$ and $|E| = m$, be a simple directed graph. For every node $v \in V$ denote the set of outgoing edges by $\texttt{out}(v)$ and the set of incoming edges by $\texttt{in}(v)$. A *network* N is a pair (G, c), where $c : E \to \mathbb{R}^+$ is a function that assigns a *capacity* $c(e)$ to each edge $e \in E$. Also, a *commodity* K in N consists of a pair (s, t) where s and t are nodes in G. We call a map $F : E \to \mathbb{R}_{\geq 0}$ a *(single-commodity) flow* for K if the following conditions are satisfied:

- $\forall v \in V \setminus \{s, t\} : \sum_{e \in \texttt{out}(v)} F(e) = \sum_{e \in \texttt{in}(v)} F(e)$,
- $\sum_{e \in \texttt{out}(s)} F(e) = d_F = \sum_{e \in \texttt{in}(t)} F(e)$,
- $\forall e \in E : F(e) \leq c(e)$,

where d_F is called the *demand* of K (w.r.t. F). We also call d_F the *size* of F.

Note that this standard formulation allows cycles to exist where the flow of a commodity is greater than zero on each edge, which can however be easily removed by flow decomposition.

Definition 3.2: We call a single-commodity flow F for commodity $K = (s, t)$ with demand d_F *cycle-free* if there is no cycle C in G, s.t. $\forall e \in C : F(e) > 0$.

For ease of notation, all flows are considered to be cycle-free from now on unless noted otherwise. Since we study the migration of flows between different nodes, we extend the flow definition to multiple commodities.

Definition 3.3: We call a tuple $\mathcal{K} := (K_1, \ldots, K_k)$ of commodities a *multi-commodity*. Let $F_1, \ldots, F_k$ be single-commodity flows for $K_1, \ldots, K_k$, respectively. Then we call the tuple $\mathcal{F} := (F_1, \ldots, F_k)$ a *multi-commodity flow* (for $\mathcal{K}$) if the following condition holds: $\forall e \in E : \sum_{i=1}^{k} F_i(e) \leq c(e)$.

In later sections, we will also study flows that cannot be split up among different paths or are of integer value, i.e.:

Definition 3.4: We call a single-commodity flow F for commodity K *unsplittable* if there is a simple path from s to t such that F assigns a value greater than zero only to edges along the path. Similarly, we call a multi-commodity flow unsplittable if all its single-commodity flows are unsplittable.

Definition 3.5: We call a single-commodity flow F for commodity K *integer* if F assigns only integer values. Similarly, we call a multi-commodity flow integer if all its single-commodity flows are integer.

Lastly, we define the term consistent migration formally. We follow the definition as used in [2], [11], [12]. Due to asynchrony, when (parts of) multiple flows are being simultaneously migrated to other parts of the network, one cannot control in which order the flows migrate: Even if the flows in the network before and after the migration are congestion-free, congestion can occur during migration. Moreover, rate-limiting/dropping is only allowed if it is required by the flow. We refer to Figure 2 for further illustration of Definition 3.6:

Definition 3.6: Let N be a network and let $\mathcal{K}_{all} = (K_1, \ldots, K_k)$ be a multi-commodity in N. Let $\mathcal{F}, \mathcal{F}'$ be multi-commodity flows for $\mathcal{K}, \mathcal{K}' \subseteq \mathcal{K}_{all}$, respectively. We call the tuple $U = (N, \mathcal{F}, \mathcal{F}')$ a *consistent migration update* if the following condition holds:

$$\forall e \in E : \sum\nolimits_{1 \leq i \leq k} \max\left(F_i(e), F_i'(e)\right) \leq c(e) \qquad (1)$$

where we assume $F_i(e) = 0$ (respectively, $F_i'(e) = 0$) if $K_i \notin \mathcal{K}$ (respectively, $K_i \notin \mathcal{K}'$). We call a sequence $((N, \mathcal{F}, \mathcal{F}_1), (N, \mathcal{F}_1, \mathcal{F}_2), \ldots, (N, \mathcal{F}_j, \mathcal{F}'))$ of consistent migration updates a consistent migration if for each commodity the demand is monotonically increasing or decreasing over all elements of the sequence $\mathcal{F}, \mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_j, \mathcal{F}'$.

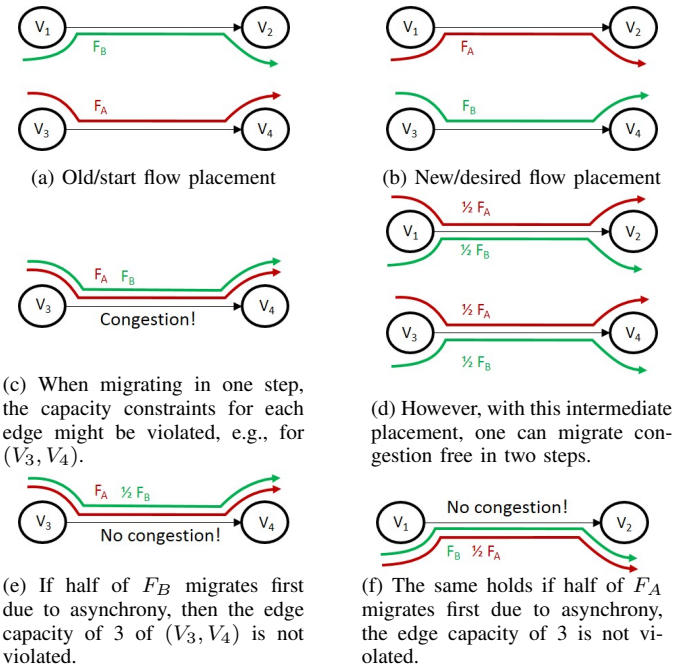
IV. Consistent Congestion-Free Migration for Unsplittable Flows

We begin studying consistent migration of flows by considering unsplittable flows. How hard is it to decide if it is possible to migrate consistently at all with unsplittable flows?

Problem 4.1: Let N be a network and let $\mathcal{F}, \mathcal{F}'$ be unsplittable multi-commodity flows in N for $\mathcal{K}, \mathcal{K}' \subseteq \mathcal{K}_{all}$. Is there a consistent migration $(N, \mathcal{F}, \mathcal{F}_1), \ldots, (N, \mathcal{F}_j, \mathcal{F}')$ s.t. all flows from $\mathcal{F}_1$ to $\mathcal{F}_j$ are unsplittable?

Theorem 4.2: Problem 4.1 is NP-hard.

Proof: Our proof works by reduction from *3-SAT* [28]. Consider the following setting: Parts of two unsplittable flows F_A, F_B need to be swapped, but the only way to do so is by temporarily storing one on a helper path P. However, that helper path P is blocked by other unsplittable flows, which need to be temporarily stored in other parts of the network as well. Storing these unsplittable flows however requires solving the *3-SAT* problem, as each of them can be stored in up to three variable gadgets (one can think of them as clauses). These variable gagdets can be set to true or false – and if and only if one finds a variable assignment that satisfies each clause, then the path P can be freed up temporarily to allow swapping parts of F_A, F_B. An illustration with a small unsatisfiable instance can be found in Figure 3. ■



(a) Old/start flow placement

(b) New/desired flow placement

(c) When migrating in one step, the capacity constraints for each edge might be violated, e.g., for (V_3, V_4).

(d) However, with this intermediate placement, one can migrate congestion free in two steps.

(e) If half of F_B migrates first due to asynchrony, then the edge capacity of 3 of (V_3, V_4) is not violated.

(f) The same holds if half of F_A migrates first due to asynchrony, the edge capacity of 3 is not violated.

Fig. 2. In this example, we want to migrate consistently from Subfigure 2a to Subfigure 2b. Each edge has a capacity of 3 and the flows F_A, F_B have a size of 2 each. If one migrates in one step, then congestion could occur. E.g., as shown in Subfigure 2c, F_B might migrate before F_A, inducing congestion on (V_3, V_4). However, if we add an intermediate step as shown in Subfigure 2d, congestion cannot occur, see Subfigure 2e and 2f.

We note that the construction only uses capacities and flows of size one. Thus, we can extend Theorem 4.2 to integer flows:

Problem 4.3: Let N be a network and let $\mathcal{F}, \mathcal{F}'$ be integer multi-commodity flows in N. Is there a consistent migration $(N, \mathcal{F}, \mathcal{F}_1), (N, \mathcal{F}_1, \mathcal{F}_2), \ldots, (N, \mathcal{F}_j, \mathcal{F}')$ s.t. all flows from $\mathcal{F}_1$ to $\mathcal{F}_j$ are integer?

Corollary 4.4: Problem 4.3 is NP-hard.

V. Consistent Congestion-Free Migration for Splittable Flows

As we saw in the last section, it is NP-hard to decide if consistent migration is possible if flows have to be integer or unsplittable. Thus, we turn our attention to splittable flows in this section. As we show, this relaxed problem is actually solvable in polynomial time:

Problem 5.1: Let N be a network and let $\mathcal{F}, \mathcal{F}'$ be multi-commodity flows for multi-commodities $\mathcal{K}, \mathcal{K}' \subseteq \mathcal{K}_{all}$. Is there a consistent migration from $\mathcal{F}$ to $\mathcal{F}'$?

Theorem 5.2: Problem 5.1 is in P.

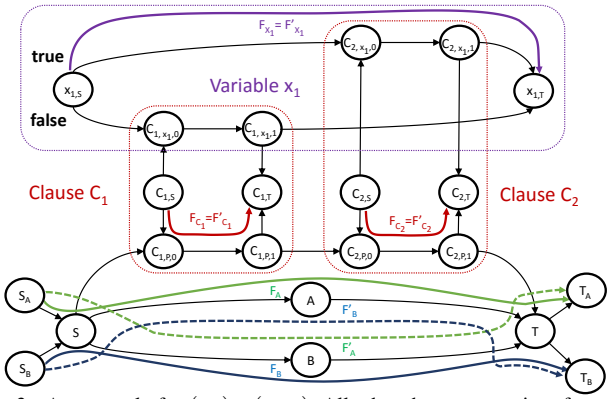


Fig. 3. An example for $(x_1) \wedge (\neg x_1)$. All edges have a capacity of one and all flows have a size of one. Note that the formula is not satisfiable. For the green flow F_A and the blue flow F_B to swap (parts of) their paths, one of them needs to migrate to the path $S, C_{1,P,0}, C_{1,P,1}, C_{2,P,0}, C_{2,P,1}, T$ above them. This requires temporary migration of the two red flows F_{C_1}, F_{C_2} along the variable gadget, since else the path will not be free. However, the violet flow F_{X_1} will always block one of the temporary migration options of one of the red clause flows, no matter where the violet flow migrates to. Thus, it is not possible to swap the green and the blue flow in this instance.

For better readability, we first present some preliminaries before actually proving Theorem 5.2. First, consider a special case of the problem where $\mathcal{K} = \mathcal{K}'$ and the demand of each commodity does not differ between $\mathcal{F}$ and $\mathcal{F}'$. We note that if a commodity only exists in either $\mathcal{K}$ or $\mathcal{K}'$ or has a higher demand in one of the two multi-commodity flows, then one could drop the corresponding excess before migration or insert it afterwards without violating monotonicity.

Problem 5.3: Let N be a network and let $\mathcal{F}, \mathcal{F}'$ be multi-commodity flows for the same multi-commodity $\mathcal{K}$ s.t., for each $K \in \mathcal{K}$, the demand of K is the same w.r.t. $\mathcal{F}$ as w.r.t. $\mathcal{F}'$. Is there a consistent migration from $\mathcal{F}$ to $\mathcal{F}'$?

We now introduce the concept of slack, i.e., an edge is not used at full capacity by a (multi-commodity) flow:

Definition 5.4: Let $\mathcal{F} = (F_1, \ldots, F_k)$ be a multi-commodity flow in N. We say that an edge e in N has *slack* w.r.t. $\mathcal{F}$ if the following condition holds: $\sum_{1 \leq i \leq k} F_i(e) < c(e)$. If for a consistent migration update $U = (N, \mathcal{F}, \mathcal{F}')$ an edge e has slack w.r.t. $\mathcal{F}'$, but not w.r.t. $\mathcal{F}$, then U *induces* slack on e.

Also, note that if every edge is used at full capacity (i.e., without slack), then consistent migration is not possible for the above problem if $\mathcal{F} \neq \mathcal{F}'$, as any consistent migration update to a multi-commodity flow $\mathcal{F}^* \neq \mathcal{F}$ would violate the capacity constraint of some full edge:

Observation 5.5: Let N be a network and let $\mathcal{F} \neq \mathcal{F}'$ be multi-commodity flows in N for the same multi-commodity $\mathcal{K}$ s.t. for each commodity $K \in \mathcal{K}$ the corresponding flows in $\mathcal{F}, \mathcal{F}'$ have the same size. If every edge is used at full capacity in $\mathcal{F}$ and $\mathcal{F}'$, then consistent migration is not possible.

On the other hand, if every edge that needs to change its flows has slack, then one can migrate consistently, cf. [2]:

Observation 5.6: Let N be a network and let $\mathcal{F}, \mathcal{F}'$ be multi-commodity flows for $\mathcal{K}$ s.t. for each commodity $K \in \mathcal{K}$ the corresponding flows in $\mathcal{F}, \mathcal{F}'$ have the same size. If every edge where $\mathcal{F}$ and $\mathcal{F}'$ differ has slack w.r.t. both $\mathcal{F}$ and $\mathcal{F}'$, then consistent migration is possible.

This gives rise to the following question: What happens when an edge does not have slack w.r.t. $\mathcal{F}$ or $\mathcal{F}'$, but not all edges are used at full capacity – is consistent migration possible?

The first step to answering this question is to identify the edges which will never admit slack (after any consistent migration) by Algorithm 5.7. Note that edges which never admit slack do not change their flow assignment in any consistent migration update, this would violate Condition (1).

Algorithm 5.7:

Input: A network N and a multi-commodity flow $\mathcal{F}$ for a multi-commodity $\mathcal{K}$.

Output: A multi-commodity flow $\mathcal{F}^*$ for $\mathcal{K}$ s.t. $a)$ for each commodity $K \in \mathcal{K}$ the corresponding flows in $\mathcal{F}, \mathcal{F}'$ have the same size and $b)$ only the edges, which never admit slack (after any consistent migration from $\mathcal{F}$), have slack w.r.t. $\mathcal{F}^*$.

1) Pick an edge $(u, v) = e_1 \in E$ without slack.
2) Pick a commodity $K \in \mathcal{K}$ used on edge e_1. We denote the corresponding flow as F. Let s^* be some positive real number s.t. 1) for all edges which have slack, s^* is smaller than the minimal slack of these edges and 2) for all edges e with $F(e) > 0$, $s^* < F(e)$.
3) Outgoing from the endpoint v of e_1, perform a BFS, where a node v'' is a child-node of a node v', if $a)$ there is an edge $e = (v', v'')$ with $F(e) > 0$ or $b)$ there is an edge (v'', v') with slack.
4) If the BFS from step 3 visits the node u, then divide the set of edges traversed in the corresponding node sequence $(v, \ldots, u)$ into 1) the set E_K which contains the edges selected by condition $a)$ in step 3 and 2) the set E_s which contains the edges selected by condition $b)$ from step 3. For all edges in E_K and for e_1, reduce the flow of commodity K by s^*, and for all edges in E_s, increase the flow of commodity K by s^*. Remove any cycling subflows, should they exist.
5) If the BFS from step 3 does not visit the node u, then repeat steps 2 to 4 for the remaining commodities used on the edge e_1, until a BFS from step 3 visits node u or all commodities have been chosen.
6) Repeat steps 1 to 5 for all other edges without slack.
7) Repeat steps 1 to 6 until either all edges have slack or until steps 1 to 6 were performed without inducing any new slack on an edge that had no slack before.

Lemma 5.8: Algorithm 5.7 produces a correct output, i.e., all edges with potential slack after a consistent migration are identified. The runtime is in $O(|\mathcal{K}||E|^3)$, i.e., polynomial.

To prove Lemma 5.8, we need a lemma which establishes a relation between the existence of a consistent migration update which induces slack and step 4 of Algorithm 5.7.

Lemma 5.9: Let N be a network and let $\mathcal{F}$ be a multi-commodity flow for $\mathcal{K}$. Let $e = (u, v)$ be an edge without slack w.r.t. $\mathcal{F}$. Then the following are equivalent:

(i) There exists a consistent migration update $(N, \mathcal{F}, \mathcal{F}')$, where $\mathcal{F}'$ is a multi-commodity flow for the multi-commodity $\mathcal{K}$ s.t. for each commodity $K \in \mathcal{K}$ the corresponding flows in $\mathcal{F}, \mathcal{F}'$ have the same size and e has slack w.r.t. $\mathcal{F}'$.

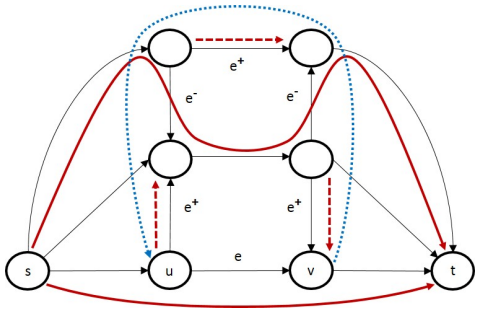


Fig. 4. In this example, all edges have a capacity of one and the solid red flow of the single commodity has a size of one along each of the two paths. For an unobstructed view, other commodities are left out. To create slack on e, parts of the red flow migrate to the edges denoted with e^+, as shown in the dashed red arrows (see (i) of Lemma 5.9 and increasing flow in step 4 of Algorithm 5.7). The dotted blue path from v to u is found via step 3 of Algorithm 5.7 and corresponds to (ii) of Lemma 5.9. We note that the edges in e^- and e^+ do not have to alternate in the path from v to u. E.g., there could be also multiple e^+ or e^- edges in a row.

(ii) There is a commodity $K \in \mathcal{K}$ with a positive flow of K on e w.r.t. $\mathcal{F}$ and a sequence of nodes $(v = v_1, v_2, \ldots, v_j = u)$ s.t. for each $1 \leq i \leq j-1$ there is $a)$ an edge (v_i, v_{i+1}) with a positive flow of commodity K w.r.t. $\mathcal{F}$ or $b)$ an edge (v_{i+1}, v_i) with slack w.r.t. $\mathcal{F}$.

Before proving Lemma 5.9, we first illustrate it (and the steps 3 and 4 of Algorithm 5.7) in Figure 4.

Proof: We begin by showing that (i) implies (ii). Let $K \in \mathcal{K}$ be a commodity where for the corresponding flows $F \in \mathcal{F}$ and $F' \in \mathcal{F}'$ holds: $F(e) > F'(e)$. Such a $K = (s,t)$ exists as the consistent migration update $(N, \mathcal{F}, \mathcal{F}')$ induces slack on e. We denote the set of all edges e_K with $F(e_K) > 0$ by E_K (with $e \in E_K$) and the set of all edges e_s with slack w.r.t. $\mathcal{F}$ by E_s. Furthermore, we denote the set of all edges e^- with $F(e^-) > F'(e^-)$ by E^- and the set of all edges e^+ with $F(e^+) < F'(e^+)$ by E^+. Note that all edges in E^- are contained in E_K and all edges in E^+ are contained in E_s.

Let V^e be the set of all nodes v' for which there exists a sequence of nodes $(v, \ldots, v')$ with the properties specified in (ii). If $u \in V^e$, then (ii) follows. Thus, assume $u \notin V^e$.

Let E_{in} be the set of all edges (x,y) with $x \notin V^e$ and $y \in V^e$. Let E_{out} be the set of all edges (x',y') with $x' \in V^e$ and $y' \notin V^e$. For any $(x,y) \in E_{\text{in}}$, it holds that $e_{\text{in}} \notin E_s$, as otherwise $x \in V^e$ by the definition of V^e. This implies $(x,y) \notin E^+$. Analogously, for all edges $(x',y') \in E_{\text{out}}$, it holds that $(x',y') \notin E_k$. This implies $(x',y') \notin E^-$. Set $\Phi_F := \sum_{e_{\text{in}} \in E_{\text{in}}} F(e_{\text{in}}) - \sum_{e_{\text{out}} \in E_{\text{out}}} F(e_{\text{out}})$. Note that $\Phi_F = \sum_{v^e \in V^e} \left(\sum_{e' \in \text{in}(v^e)} F(e') - \sum_{e' \in \text{out}(v^e)} F(e') \right)$, which implies the following by the defining conditions of a flow: If $s \in V^e, t \notin V^e$, then $\Phi_F = -d_F$. If $s \notin V^e, t \in V^e$, then $\Phi_F = d_F$. Otherwise, $\Phi_F = 0$. We have analogous statements for F' and since $d_F = d_{F'}$, we obtain $\Phi_F = \Phi_{F'}$. On the other hand, $\Phi_{F'} - \Phi_F = \sum_{e_{\text{in}} \in E_{\text{in}}} (F'(e_{\text{in}}) - F(e_{\text{in}}))$ $+ \sum_{e_{\text{out}} \in E_{\text{out}}} (F(e_{\text{out}}) - F'(e_{\text{out}}))$. As shown above, all edges in E_{in} are not in E^+.

Thus, every summand in the first sum is ≤ 0. Analogously, since all edges in E_{out} are not in E^-, every summand in the second sum is ≤ 0. Furthermore, since $e \in E_{\text{in}}$ and $F(e) > F'(e)$, there is a negative summand in the first sum. Hence, $\Phi_{F'} < \Phi_F$, contradicting $\Phi_{F'} = \Phi_F$. Therefore, $u \in V^e$ which shows that (i) implies (ii).

It is left to show that (ii) implies (i). Let s^* be some positive real number s.t. 1) for all edges which have slack w.r.t. $\mathcal{F}$, s^* is smaller than the minimal slack of these edges and 2) for all edges e' with $F(e') > 0$, $s^* < F(e')$. Let E_1 be the union of the set $\{e\}$ and the set of edges described by $a)$ in (ii) and E_2 the set of edges described by $b)$ in (ii).

Now define $\mathcal{F}'$ as follows: For all commodities in $\mathcal{K}$ except K, take the corresponding flow from $\mathcal{F}$. For K, set

$$F'(\bar{e}) := \begin{cases} F(\bar{e}) - s^* & \text{if } \bar{e} \in E_1, \\ F(\bar{e}) + s^* & \text{if } \bar{e} \in E_2, \\ F(\bar{e}) & \text{otherwise.} \end{cases} \tag{2}$$

By the definition of s^*, no capacity constraints will be violated by $\mathcal{F}'$. For all nodes which are not in the sequence given in (ii), the flow conservation condition holds w.r.t. $\mathcal{F}'$, as no ingoing or outgoing flows have been changed. For each node which is in the sequence given in (ii), the (two) incident edges with changed flow F' (compared to F) cancel each other out regarding flow conservation. If the nodes s and t are not in the sequence given in (ii), then $d_{F'} = d_F$ holds. W.l.o.g., let s be in the sequence given in (ii): Then there could be a cycle of commodity K but increasing the outgoing flow of commodity K for s. In that case, we transform F into a cycle-free flow by subtracting the cycling "subflows" from the affected edges. This does not violate flow conservation or capacity constraints and ensures $d_{F'} = d_F$.

Since we only change the flow of one commodity from $\mathcal{F}$ to $\mathcal{F}'$, Condition 1 is satisfied because no capacity constraints are violated by $\mathcal{F}'$ as shown before. Thus, $(N, \mathcal{F}, \mathcal{F}')$ is a consistent migration update. As $e \in E_1$, $(N, \mathcal{F}, \mathcal{F}')$ induces slack on e, which shows that (ii) implies (i). ■

Now we are able to prove Lemma 5.8.

Proof: We begin with the observation that once we have slack on an edge, we never need to remove the slack on this edge completely in order to induce slack on other edges. Assume that there is a consistent migration update that induces slack on edge e_1 and removes slack on edge e_2. If we change the migration update to just migrate half the size of the migrated flows, we will still have slack on both edge e_1 and edge e_2. This is essential to the proof, as it cannot be done with unsplittable or integer flows. Furthermore, it shows that the size of the slack is not relevant for any further steps of an (appropriately designed) algorithm that tries to induce slack on as many edges as possible. As we consider splittable flows, for any consistent migration update, one can just reduce the size of the moved flow s.t. it still leaves slack on the newly used edges, but still induces slack on the previously more heavily used edges. Thus, we only need to differentiate for each edge whether $i)$ it has slack, or $ii)$ it has no slack.

Recall from Definition 3.6 that a consistent migration update is only possible if there is some slack on the edges where the flow is being moved to. Else Condition (1) will be violated. Consider any step 4 of Algorithm 5.7, where the BFS from step 3 visits the node u. The flow transformations being performed

subsequently correspond to the construction of F' in the proof of Lemma 5.9 (given by (2)). Thus, by Lemma 5.9, all flow transformations performed by Algorithm 5.7 are consistent migration updates.

What is left to show (apart from the polynomial runtime) is that Algorithm 5.7 induces slack on each edge on which slack can be induced by any consistent migration. Thus, assume, for the sake of contradiction, that there is a consistent migration M after which there is slack on some edge e_x which had no slack initially, but Algorithm 5.7 does not induce slack on e_x.

Let E_i be the set of all edges which had no slack initially, on which M induces slack in update i. Let j be the smallest index s.t. E_j contains an edge e_j on which Algorithm 5.7 does not induce slack. Then, Algorithm 5.7 induces slack on all edges from $E_1 \cup E_2 \cup \cdots \cup E_{j-1}$. Let $K \in \mathcal{K}$ be a commodity for which the corresponding flow size on edge e_j gets reduced in the j-th update of M.

Let $\mathcal{F}$ be the initially given flow and $\mathcal{F}'$ the flow after the j-th update of M. Let $\mathcal{F}^*$ be the flow obtained by running Algorithm 5.7 on $\mathcal{F}$. Let $F \in \mathcal{F}$, $F' \in \mathcal{F}'$ and $F^* \in \mathcal{F}^*$ be the flows corresponding to commodity $K = (s,t)$. Let E^- denote the set of edges e with $F'(e) < F(e)$ and E^+ the set of edges e' with $F'(e') > F(e')$. Every edge $e \in E^+$ must have had slack at some point before the j-th update of M as some flow for commodity K has been added during the migration from $\mathcal{F}$ to $\mathcal{F}'$. By the definition of j, this implies that e has slack also w.r.t. $\mathcal{F}^*$. Furthermore, for each edge $e' \in E^-$, we have $F(e') > 0$, and since the design of Algorithm 5.7 ensures that any edge containing some flow for some commodity always keeps some positive flow for this commodity, we obtain $F^*(e') > 0$.

Now define a flow $\overline{F}$ for $\mathcal{K}$ by setting

$$\overline{F}(e) := \begin{cases} F^*(e) - r(F(e) - F'(e)) & \text{if } e \in E^-, \\ F^*(e) + r(F'(e) - F(e)) & \text{if } e \in E^+, \\ F^*(e) & \text{otherwise,} \end{cases}$$

for the flow $\overline{F}$ for commodity K and taking the flows from $\mathcal{F}^*$ for all other commodities in $\mathcal{K}$. The parameter $r \in \mathbb{R}_+$ in the definition of $\overline{F}$ will be determined in the following. Note that the first two terms in the above definition are equal.

We have to show that $\overline{F}$ is indeed a flow (and $\overline{\mathcal{F}}$ indeed a multi-commodity flow). As $\overline{F}$ is a linear combination of the flows F^*, $F(e)$ and $F'(e)$ (which all satisfy the flow conservation condition on the nodes different from s and t), it also satisfies the flow conversation condition. Furthermore, as F and F' cancel each other out regarding the flow (size) outgoing from s and incoming in t, $\overline{F}$ satisfies also the second flow condition and the demand of K w.r.t. F is also the same as w.r.t. F^*, F and F'. By choosing r small enough, we can also ensure that the third flow condition is satisfied (in the general form required for multi-commodity flows, also given in the model section), i.e., that $\overline{\mathcal{F}}$ does not violate the capacity constraints. (This last claim follows from the fact that the only edges e with $\overline{F}(e) > F^*(e)$ are those in E^+ and we already showed that all of these edges have slack w.r.t. F^*.)

In order to avoid having "negative" flows on some edges, we must take care that for the edges in E^- (for which $F'(e) < F(e)$ holds), $F^*(e) - r(F(e) - F'(e))$ is positive. As we showed above, we have $F^*(e) > 0$ for all edges $e \in E^-$. Thus, we can ensure the required positivity by again choosing r small enough. Now fix r s.t. it is small enough for the arguments in the above discussion. Then $\overline{F}$ is a flow for commodity K and $\overline{\mathcal{F}}$ is a multi-commodity flow for $\mathcal{K}$. Furthermore, for all commodities in $\mathcal{K}$ different from K, the corresponding flows are identical in $\overline{\mathcal{F}}$ and $\mathcal{F}^*$. Thus, $(N, \mathcal{F}^*, \overline{\mathcal{F}})$ is a consistent migration update as Condition 1 must be satisfied (since both $\overline{\mathcal{F}}$ and $\mathcal{F}^*$ do not violate the capacity constraints of the edges). Moreover, for each commodity in $\mathcal{K}$, the corresponding flows in $\overline{\mathcal{F}}$ and $\mathcal{F}^*$ have the same size.

Consider the edge e_j. By its definition, we have $F'(e_j) < F(e_j)$ which implies $e_j \in E^-$. Thus, $\overline{F}(e_j) < F^*(e_j)$ and e_j has slack w.r.t. $\overline{F}$. So we have shown that $(N, \mathcal{F}^*, \overline{\mathcal{F}})$ is a consistent migration update as described in statement (i) of Lemma 5.9 whereas e_j is an edge without slack w.r.t. $\mathcal{F}^*$. By applying Lemma 5.9, we obtain the corresponding statement (ii) from Lemma 5.9 (where "e" $= e_j$ and "$\mathcal{F}$" $= \mathcal{F}^*$). It follows that after reaching flow $\mathcal{F}^*$, Algorithm 5.7 will pick the edge e_j in some step 1 and the commodity K in step 2. In the subsequent step 4, Algorithm 5.7 will find a node sequence which ends in the starting node of e_j (the existence of such a sequence is ensured by the above statement (ii) from Lemma 5.9). Thus, Algorithm 5.7 will perform a consistent migration update which induces slack on the edge e_j. This is a contradiction to the assumption and Algorithm 5.7 produces a correct output.

It is left to show that Algorithm 5.7 runs in polynomial time: For the analysis, we first ignore the runtime contributed by step 4. Steps 1 to 5, excluding 4, can be performed in $O(|\mathcal{K}||E|)$ time – with step 6 iterating this process $O(|E|)$ times. Step 7 will repeat steps 1 to 6, excluding 4, again $O(|E|)$ times, resulting in a total runtime of $O(|\mathcal{K}||E|^3)$. Observe that step 4 will be executed at most $O(|E|)$ times, as it is only run when slack is generated for the first time on an edge. Increasing and decreasing the flows in step 4 can be performed in $O(|E|)$ time, leaving the cycle removal: By selecting an edge e with smallest flow size $F_K(e)$ of commodity K, we can determine if there is a cycle for the commodity K containing e, and if this is the case, remove such a cycle, setting $F_K(e) = 0$ in the process. Such a cycle removal with a runtime of $O(|E|)$ needs to be performed at most $|E|$ times, yielding a total runtime of $O(|E|^3)$ for all executions of step 4. Hence, the total runtime of Algorithm 5.7 is $O(|\mathcal{K}||E|^3)$. ∎

Furthermore, Problem 5.1 can be reduced to Problem 5.3 by essentially i) first dropping (parts of) commodities that do not need to migrate, and ii) adding new (parts of) commodities at the end. Then, Algorithm 5.7 can be applied to both $\mathcal{F}$ and $\mathcal{F}'$. Should there be edges in N, on which i) no slack can be induced starting from $\mathcal{F}$ or $\mathcal{F}'$ by means of consistent migration, and that ii) differ in their flow assignment in $\mathcal{F}$ and $\mathcal{F}'$, then consistent migration is not possible. Else, one can use the approach of a) applying Algorithm 5.7 to the old and new placement to ensure slack s on each edge and then

migrate in at most $\lceil 1/s \rceil - 1$ consistent migration updates, or b) use the method of binary search via LPs from [2] to find a consistent migration. The formal proof of Theorem 5.2 is given in the following.

Proof of Theorem 5.2: We start to address this problem by first recalling Observation 5.5: I.e., if all edges are at full capacity, the commodities and demands stay the same, and some flows have to change their placement in the network, any consistent migration step would violate the capacity constraint of some edge. Furthermore, assume that a commodity only exists in the initial starting state, i.e., in $\mathcal{K}$, but not in $\mathcal{K}'$. Then, one can just remove the corresponding flow/commodity and consider this reduced problem. The same holds if the commodity just exists in the desired state, i.e., in $\mathcal{K}'$, but not in $\mathcal{K}$. One can ignore this commodity when migrating – and just add its flow at the end, as there will be enough space on all of its used edges. Similarly, if there is some slack on each edge for both $\mathcal{F}$ and $\mathcal{F}'$, then it is possible to migrate consistently, see Observation 5.6.

Thus, we want to identify the edges whose usage by flows cannot be changed by consistent migration. We refer to Subfigure 1d for an example: As all of the edges are used to full capacity in both the old starting state with $\mathcal{F}$ and the new desired state with $\mathcal{F}'$, the flow assignment of any edge cannot be changed, unless one violates congestion-freedom or rate-limits some flow. Note that a necessary requirement for such an edge is that it is used at full capacity in both the old starting state $\mathcal{F}$ and the new desired state $\mathcal{F}'$.

Hence, when a commodity with the same source and sink has different demands $d_{F_{old}}, d_{F_{new}}$ in the old and new state with flows F_{old}, F_{new}, we only need to look at the minimum of $d_{F_{old}}, d_{F_{new}}$: Assume $d_{F_{old}} < d_{F_{new}}$ with $d_{F_{old}} = y * d_{F_{new}}$, $y > 1$. Then, we can reduce the flow of F_{new} by a factor of y on all of its used edges, and after a consistent migration, add the missing demand over all edges. The same holds if $d_{F_{old}} > d_{F_{new}}$ with $y * d_{F_{old}} = d_{F_{new}}$, $y > 1$: We reduce the flow of F_{old} by a factor of y on all of its used edges, and then consider consistent migration. In both cases, we will only lower the used capacity of edges by consistent migration steps. Therefore, we can simplify our original problem to one where the demands and commodities are equal for $\mathcal{F}$ and $\mathcal{F}'$, i.e., $\mathcal{K} = \mathcal{K}' = \mathcal{K}_{all}$. Note that the problem is symmetric in the sense that consistent migration from $\mathcal{F}$ to $\mathcal{F}'$ is possible if and only if consistent migration from $\mathcal{F}'$ to $\mathcal{F}$ is possible.

This means that if there is an edge whose usage by flow $\mathcal{F}$ or $\mathcal{F}'$ cannot be changed by any sequence of consistent migration steps, and it is not used in the same fashion by $\mathcal{F}$ and $\mathcal{F}'$, then consistent migration from $\mathcal{F}$ to $\mathcal{F}'$ is not possible! Conversely, if no such edge exists, we can migrate consistently: If each edge that needs to be changed in its usage of flows has some slack, we can migrate consistently by always changing a small part of the network in accordance with the slack. I.e., if the minimum slack is 10%, we can migrate in 9 steps.

Therefore, we consider all edges that are i) at full capacity without slack for $\mathcal{F}$ or $\mathcal{F}'$, and ii) not used the same by $\mathcal{F}$ or $\mathcal{F}'$, and try to figure out if we can find some sequence of consistent migration steps that induces slack on them. Thus, we only need to look at an even more restricted problem:

Let N be a network and $\mathcal{F}^*$ be a flow in N with commodities $\mathcal{K}^*$. On what edges can we induce slack by a sequence of consistent migration steps?

However, this is exactly the problem solved by Algorithm 5.7 in polynomial time. As all steps mentioned before can be performed in polynomial time as well, Problem 5.1 is in the complexity class P. ∎

VI. Insertion of Unsplittable flows

A natural method to insert/increase a flow in a network is to check first if there is enough space. In that case, the solution is straightforward – one just increases/adds the flow in question. However, things get more difficult if there is currently not enough capacity. Is it necessary to remove some flows, or is it enough to consistently migrate the existing flows?

A common method (cf. *RSVP-TE* [29]) is to assign levels of importance to all flows in the network, and then remove those of lesser importance if they block the new flow. We show that from a theoretical standpoint, this approach is problematic for unsplittable flows, as we prove the following corresponding decision problem to be NP-hard:

Problem 6.1: Let N be a network and let $\mathcal{F}$ be an unsplittable multi-commodity flow in N for a multi-commodity $\mathcal{K}$. Let K_{new} be a commodity not contained in $\mathcal{K}$ and let M be a map of each commodity in $\mathcal{K}$ to a level of importance, i.e., $M : \mathcal{K} \to \mathbb{N}$. Let r be some integer. Is there a set $\mathcal{K}_r \subseteq \mathcal{K}$ of commodities with summed up importance at most r, an unsplittable multi-commodity flow $\mathcal{F}'$ in N for the multi-commodity $\{\mathcal{K} \setminus \mathcal{K}_r\} \cup K_{new}$, and a consistent migration $(N, \mathcal{F}, \mathcal{F}_1), (N, \mathcal{F}_1, \mathcal{F}_2), \ldots, (N, \mathcal{F}_j, \mathcal{F}')$ s.t. all flows $\mathcal{F}_i$ are unsplittable?

Theorem 6.2: Problem 6.1 is NP-hard.

Proof: The proof follows directly from the proof of Theorem 4.2 if one considers the following case: Importance $r = 0$, $\mathcal{F}$ and $\mathcal{K}$ as given in the proof of Theorem 4.2, and K_{new} with source S and sink T, cf. Figure 3. ∎

This implies that the corresponding optimization version of Problem 6.1 can also not be approximated well: Any (constant) approximation ratio for r would mean that one could decide if the problem is satisfiable or not.

Corollary 6.3: The optimization version of Problem 6.1, i.e, minimizing r, does not admit a PTAS unless P = NP.

We can take this problem even one step further and ask about the hardness of approximation regarding the additive error. Maybe one could always migrate in the construction from our proof if one just removes just a constant amount of the flows of the lowest importance? If we assign the clause flows the importance 1 and the remaining flows the importance # *of clauses*, then this problem reduces to finding a variable assignment that satisfies as many clauses as possible – i.e., solving *MAX 3-SAT*. However, as shown by Håstad [30], this is NP-hard to approximate better than $7/8 + \epsilon$.

Maximize $\sum_{i:e_i \in \mathtt{out}(s_1)} x_{i1}$
subject to
1) $\forall 1 \leq j \leq k \forall v \in V \setminus \{s_j, t_j\}$:
$\sum_{i:e_i \in \mathtt{out}(v)} x_{ij} = \sum_{i:e_i \in \mathtt{in}(v)} x_{ij}$,
2) $\forall 2 \leq j \leq k : \sum_{i:e_i \in \mathtt{out}(s_j)} x_{ij} = d_j = \sum_{i:e_i \in \mathtt{in}(t_j)} x_{ij}$,
3) $\forall 1 \leq j \leq k : \sum_{i=1}^{k} x_{ij} \leq c(e_j)$,
4) $\forall 1 \leq i \leq m$ s.t. $e_i \in E_{\mathtt{fix}} \forall 1 \leq j \leq k : x_{ij} = F_j(e_i)$,
5) $\sum_{i:e_i \in \mathtt{in}(s_1)} x_{i1} = 0$.

Fig. 5. The LP doesn't alter the flow on the edges from $E_{\mathtt{fix}}$ due to 4).

VII. Increasing Splittable Flows

As shown in Section VI, the consistent increase/insertion of flows is an NP-hard problem if flows are unsplittable. This leads to the natural question of *how much* the demand of a commodity can be increased under the condition of consistency and splittable flows. Then, one can make an informed decision if it is worth it to violate consistency or not: Maybe the new demand is for a critical application that absolutely needs the bandwidth – or maybe it is just some background data that is not time critical.

In Section V we showed that it is decidable in polynomial time if consistent migration is possible. However, this does not answer the question of to what unknown new desired flow placement one should migrate. One could just solve an LP maximizing the demand of one commodity while keeping the demand of the other commodities fixed (cf. the LP in Figure 5 without restriction 4)). But it can be the case that consistent migration is not possible for the resulting multi-commodity flow of the LP. We thus formulate the following problem:

Problem 7.1: Let N be a network and let $\mathcal{F}$ be a multi-commodity flow in N for the multi-commodity $\mathcal{K}$. Let $K \in \mathcal{K}$. Find a multi-commodity flow $\mathcal{F}'$ for the multi-commodity $\mathcal{K}$ that $i)$ maximizes the demand of commodity K, $ii)$ leaves the demand of all other commodities unchanged, and $iii)$ can be migrated to consistently from $\mathcal{F}$.

For ease of notation, we assume that if one wants to maximize a new commodity, it is denoted as $K \in \mathcal{K}$ with a current demand of 0. As it turns out, we can approximate the maximum consistent increase of the demand of commodity K in Problem 7.1 arbitrarily well:

Theorem 7.2: Let $\epsilon > 0$. Finding a multi-commodity flow for Problem 7.1 that satisfies the conditions $ii)$ and $iii)$ and approximates the maximum demand of commodity K in condition $i)$ with an approximation ratio of $(1 - \epsilon)$ can be done in polynomial time (independent of the chosen ϵ).

Proof: Let $\mathcal{F} = (F_1, ..., F_k)$ be the given (initial) multi-commodity flow for the multi-commodity $(K_1, ..., K_k)$ where commodity $K_j = (s_j, t_j)$ with demand d_j w.r.t. $\mathcal{F}$. Let $K = K_1$. Using Algorithm 5.7, we can determine all the edges which do not admit slack after any consistent migration starting from $\mathcal{F}$. Let $E_{\mathtt{fix}}$ denote the set of these edges. Due to Condition (1), consistent migration updates cannot change the flow assignments on $E_{\mathtt{fix}}$, so we would like to fix them for the (approximate) maximum flow we are looking for.

By solving the LP given in Figure 5, we can find a multi-commodity flow $\mathcal{F}^*$ that maximizes the demand of commodity K. Any found optimal solution of the LP represents a multi-commodity flow $\mathcal{F}^*$ with the above-mentioned properties by setting $F_j(e_i) := x_{ij}$, $1 \leq i \leq m$, $1 \leq j \leq k$. The first three constraints represent the usual flow prerequisites. The fourth constraint guarantees that nothing changes (from $\mathcal{F}$ to $\mathcal{F}^*$) on the edges in $E_{\mathtt{fix}}$. As the LP does not check if the solution(s) represent cycle-free flows, we need the last constraint: It ensures that no part of the flow leaving s_j returns to s_j (such a part would not contribute to the flow F_j from s_j to t_j, but to the sum we are trying to maximize). If we obtain a solution to the LP with a cycle, then we can transform it into a cycle-free flow by subtracting the cycling "subflows" from the affected edges. Thus, we may still assume cycle-freedom.

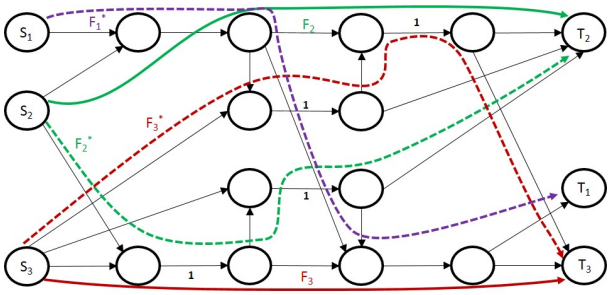


Fig. 6. In this network, all edges have a capacity of two – except for the ones denoted with a capacity of 1. The commodities K_2 and K_3 have a demand of one. In the initial flow $\mathcal{F}$, consisting of F_2 and F_3, one can achieve slack on all edges by rerouting some parts of F_2 and F_3 along alternate paths. In the optimal solution $\mathcal{F}^* = (F_1^*, F_2^*, F_3^*)$ of the LP from Figure 5 that maximizes the demand for K_1 (which is then two), the flows F_2 and F_3 of size one have to be rerouted along the paths denoted by F_2^* and F_3^*. However, by means of consistent migration it is not possible to induce slack on the 1-edges starting from (F_2^*, F_3^*), even if F_1^* is not inserted yet: F_2^* would need to use at least one of the 1-edges fully occupied by F_3^* and vice versa. Thus, it is not possible to consistently migrate between $\mathcal{F}^*$ and $\mathcal{F}$.

While it is not ensured that it is possible to migrate consistently from $\mathcal{F}$ to $\mathcal{F}^*$, the obtained maximized demand $d' := \sum_{i:e_i \in \mathtt{out}(s_1)} x_{i1}$ gives us an upper bound for the demand of commodity K w.r.t. flow $\mathcal{F}^*$, subject to the condition that all other demands remain as they are. If we demand consistent migration, then the demand d' cannot necessarily be achieved, but we come arbitrarily close. We refer to Figure 6 for an example where d' cannot be achieved.

However, the initial flow $\mathcal{F}$ and an arbitrary optimal solution $\mathcal{F}^*$ of the LP can be combined to a multi-commodity flow $\mathcal{F}'$ which can be migrated to consistently from $\mathcal{F}$. The combination is parametrized by some $0 < r < 1$ which determines the demand of the commodity whose demand we are trying to maximize, and thus also determines the approximation ratio. We define $\mathcal{F}'$ by $F'_j(e) := rF_j^*(e) + (1 - r)F_j(e)$ for all $1 \leq j \leq k$ and $e \in E$. It follows directly from the definition that $\mathcal{F}'$ is a multi-commodity flow for which the demands $K_2, ..., K_k$ did not change from $\mathcal{F}$. What is left to show is that we can migrate consistently from $\mathcal{F}$ to $\mathcal{F}'$. Recall that starting from $\mathcal{F}$, slack can be achieved on exactly the edges which are not in $E_{\mathtt{fix}}$. We show that the same is true for $\mathcal{F}'$:

We can "divide" the given network N into two networks N_1 and N_2 which have the same underlying graph as N but have less capacity – for N_1 each edge capacity in N is multiplied by r, for N_2 by $(1 - r)$. We can imagine the flow $r\mathcal{F}^*$ as

living only on the N_1 part of N and $(1-r)\mathcal{F}$ as living only on the N_2 part. Now any consistent migration of $\mathcal{F}$ in N represents a consistent migration of $(1-r)\mathcal{F}$ in N_2 and thus a consistent migration of $\mathcal{F}'$ in N. It follows that slack (starting from $\mathcal{F}'$) can be achieved on all edges except possibly on those in $E_{\texttt{fix}}$. We will show now that slack cannot be induced on any $e \in E_{\texttt{fix}}$ starting from $\mathcal{F}'$. By the above discussion, there is a consistent migration from $\mathcal{F}'$ to a flow $\overline{\mathcal{F}}$ (for the same multi-commodity) for which exactly the edges not in $E_{\texttt{fix}}$ have slack (just apply only updates which do not affect N_1).

As discussed in the proof of Lemma 5.8, the result of Algorithm 5.7 in terms of which edges have slack, does not depend on any slack size, but just on whether an edge has slack or not. The set of edges which have slack is the same w.r.t. $\overline{\mathcal{F}}$ as w.r.t. the flow Algorithm 5.7 returns after running on $\mathcal{F}$ (namely, $E \backslash E_{\texttt{fix}}$). Thus, running Algorithm 5.7 (again) for those two flows will yield the same set of edges with slack. As the latter has already achieved slack on all edges where this is possible, slack cannot be induced on any $e \in E_{\texttt{fix}}$, starting from $\overline{\mathcal{F}}$. Thus, the same is true for $\mathcal{F}'$.

So slack can be achieved on the edges not in $E_{\texttt{fix}}$, both starting from $\mathcal{F}$ and $\mathcal{F}'$. Furthermore, as $\mathcal{F}^*$ is a solution to the above LP, we have $F_j^*(e) = F_j(e)$ for all $1 \leq j \leq k$, $e \in E_{\texttt{fix}}$. Thus, the flow assignment on the edges in $E_{\texttt{fix}}$ is the same for $\mathcal{F}$ and $\mathcal{F}'$. Ignoring these edges (whose flow assignments do not need to be changed), we can use the technique provided by [2] to obtain a consistent migration between $\mathcal{F}$ and $\mathcal{F}'$ as all edges that need to change their flow assignments have slack w.r.t both $\mathcal{F}$ and $\mathcal{F}'$ (after applying Algorithm 5.7). Hence, we obtain that there is a consistent migration between the multi-commodity flows $\mathcal{F}$ and $\mathcal{F}'$.

It is left to show that the runtime is polynomial (independent of the chosen ϵ): Solving a linear program as the LP in Figure 5 can be done in polynomial runtime (e.g., with the *interior point method*). Furthermore, Algorithm 5.7 has a polynomial runtime as well, see Lemma 5.8. ϵ comes only into play when "dividing" the network into two networks – but there it is only used for multiplication to achieve the desired approximation ratio for the demand of commodity $K = K_1$. ∎

VIII. Conclusion

We studied the problem of migrating flows consistently, i.e., without congestion or rate-limiting, with special focus on software defined networks. We were able to show that for splittable flows, it is possible to decide in polynomial time if consistent migration is possible. All previous approaches could not decide if consistent migration is possible or applied only to specific subsets of the consistent migration problem. We proved that splittable flows are essential for this result, as the decision problem is NP-hard for unsplittable or integer flows.

Furthermore, we also studied the problem of consistently increasing or inserting new flows into the network. A current practice is to drop obstructing flows: We showed that optimizing this technique is NP-hard to as well. However, we proved that one can approximate the maximal consistent increase for the size of a (new) flow arbitrarily well in polynomial time.

References

[1] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hoelzle, S. Stuart, and A. Vahdat, "B4: Experience with a globally-deployed software defined WAN," in *SIGCOMM*, 2013.

[2] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer, "Achieving high utilization with software-driven WAN," in *SIGCOMM*, 2013.

[3] N. Feamster, H. Balakrishnan, J. Rexford, A. Shaikh, and K. van der Merwe, "The Case for Separating Routing from Routers," in *FDNA*, 2004.

[4] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. Mckeown, and S. Shenker, "ETHANE: taking control of the enterprise," in *SIGCOMM*, 2007.

[5] A. Greenberg, G. Hjalmtysson, D. A. Maltz, A. Myers, J. Rexford, G. Xie, H. Yan, J. Zhan, and H. Zhang, "A clean slate 4D approach to network control and management," in *SIGCOMM CCR*, 2005.

[6] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: dynamic flow scheduling for data center networks," in *NSDI*, 2010.

[7] T. Benson, A. Anand, A. Akella, and M. Zhang, "MicroTE: Fine grained traffic engineering for data centers," in *CoNEXT*, 2011.

[8] M. Borokhovich and S. Schmid, "How (Not) to Shoot in Your Foot with SDN Local Fast Failover," in *OPODIS*, 2013.

[9] M. Casado, N. Foster, and A. Guha, "Abstractions for software-defined networks," *Commun. ACM*, vol. 57, no. 10, pp. 86–95, 2014.

[10] B. Gardiner, "Google's Submarine Cable Plans Get Official. February 2008." http://www.wired.com/2008/02/googles-submari.

[11] X. Jin, H. Liu, R. Gandhi, S. Kandula, R. Mahajan, J. Rexford, R. Wattenhofer, and M. Zhang, "Dionysus: Dynamic Scheduling of Network Updates," in *SIGCOMM*, 2014.

[12] H. H. Liu, X. Wu, M. Zhang, L. Yuan, R. Wattenhofer, and D. A. Maltz, "zUpdate: updating data center networks with zero loss," in *SIGCOMM*, 2013.

[13] F. T. Leighton and S. Rao, "Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms," *J. ACM*, vol. 46, no. 6, pp. 787–832, 1999.

[14] A. V. Goldberg, J. D. Oldham, S. A. Plotkin, and C. Stein, "An implementation of a combinatorial approximation algorithm for minimum-cost multicommodity flow," in *IPCO*, 1998.

[15] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms (3. ed.)*. MIT Press, 2009.

[16] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network flows - theory, algorithms and applications*. Prentice Hall, 1993.

[17] T. Mizrahi, E. Saat, and Y. Moses, "Timed consistent network updates," in *SOSR*, 2015.

[18] T. Mizrahi, O. Rottenstreich, and Y. Moses, "TimeFlip: Scheduling network updates with timestamp-based TCAM ranges," in *INFOCOM*, 2015.

[19] S. Ghorbani and M. Caesar, "Walk the line: Consistent network updates with bandwidth guarantees," in *HotSDN*, 2012.

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

[21] J. Perry, A. Ousterhout, H. Balakrishnan, D. Shah, and H. Fugal, "Fastpass," in *SIGCOMM*, 2014.

[22] S. Brandt, K.-T. Foerster, and R. Wattenhofer, "Augmenting Anycast Flows," in *ICDCN*, 2016.

[23] M. Hoefer, V. S. Mirrokni, H. Röglin, and S. Teng, "Competitive routing over time," *Theor. Comput. Sci.*, vol. 412, no. 39, pp. 5420–5432, 2011.

[24] A. Noyes, T. Warszawski, P. Cerný, and N. Foster, "Toward synthesis of network updates," in *SYNT*, 2013.

[25] J. McClurg, H. Hojjat, P. Cerný, and N. Foster, "Efficient Synthesis of Network Updates," in *PLDI*, 2015.

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

[27] S. Vissicchio and L. Cittadini, "FLIP the (Flow) Table: Fast LIghtweight Policy-preserving SDN Updates," in *INFOCOM*, 2016.

[28] M. R. Garey and D. S. Johnson, *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., 1990.

[29] D. Awduche, L. Berger, D. Gan, T. Li, V. Srinivasan, and G. Swallow, "RFC 3209, RSVP-TE: Extensions to RSVP for LSP Tunnels," 2001.

[30] J. Håstad, "Some optimal inapproximability results," *J. ACM*, vol. 48, no. 4, pp. 798–859, 2001.