Augmenting Flows for the Consistent Migration of Multi-Commodity Single-Destination Flows in SDNs ☆

Sebastian Brandt[a], Klaus-Tycho Foerster[a], Roger Wattenhofer[a]

[a]*ETH Zürich, Gloriastrasse 35, 8092 Zurich, Switzerland*

Abstract

Updating network flows in a real-world setting is a nascent research area, especially with the recent rise of Software Defined Networks. While augmenting *s*-*t* flows of a single commodity is a well-understood concept, we study updating flows in a multi-commodity setting: Given a directed network with flows of different commodities, how can the capacity of some commodities be increased, without reducing capacities of other commodities, when moving flows in the network in an orchestrated order? To this extent, we show how the notion of augmenting flows can be efficiently extended to multiple commodities for applications with a single logical destination. We also show that our methods induce stronger consistency settings than previous work. Lastly, we prove the consistent migration to new demands to be NP-hard for unsplittable flows, and discuss extensions for the case of multiple source-destination pairs.

Keywords: Software Defined Networks, Congestion, Multi-Commodity Flow, Anycast, Flow Augmentation, Consistent Migration

1. Introduction

The rise of *Software Defined Networks (SDNs)* has sparked an increasing interest in applying network flow algorithms. While smart traffic engineering is not only studied or enabled in SDNs, utilizing the available bandwidth almost completely is one of the central topics in SDN research [21]. The algorithmic tool to manage network traffic in an efficient way is provided by flow algorithms.

Since network traffic is highly dynamic, existing SDN solutions [8, 21, 24, 25, 30] frequently re-compute the optimal way to route traffic demands, usually using an approach based on linear programming (LP), often accepting the overhead that a new solution will re-route many existing flows that did not change their demands.

☆A preliminary extended abstract appeared in the Proceedings of the 17th International Conference on Distributed Computing and Networking (ICDCN '16), ACM, New York, NY, USA, `http://dx.doi.org/10.1145/2833312.2833450` [4].

Email addresses: `brandts@ethz.ch` (Sebastian Brandt), `foklaus@ethz.ch` (Klaus-Tycho Foerster), `wattenhofer@ethz.ch` (Roger Wattenhofer)

In this article we propose to abandon LP-based solutions in favor of path augmentation, a technique developed 60 years ago [15]: We believe that SDNs should be managed in an incremental way. If a commodity (a source-destination node pair) wants to reduce its bandwidth, no changes need to be made. If a commodity wants to increase its bandwidth (or a new commodity is introduced to the network), we try to increase its flow by using path augmentation. In the best case, the increased demand fits without changing any of the other flows. However it can be that re-routing (also called migration) of some other flows is necessary, conceivably even recursively.

The problem of flow migration still has multiple open research questions: It is not clear in general (1) how to classify when congestion-free migration is possible, (2) to what solution one should actually attempt to migrate, (3) how to reasonably bound the migration time.

Moreover, there is another problem: Apart from a few exceptions that we discuss in the related work section, path augmentation was only developed for s-t-flow problems with a *single* commodity. In real networks we have *multiple* commodities, so we first need to generalize path augmentation to *flow augmentation*, a path augmentation technique supporting multiple commodities.

It turns out that generalizing path augmentation is not as easy as one may hope. As Hu notes in his influential article [22], "*it is unlikely that similar techniques can be developed for constructing multicommodity flows*".

This is why this article focuses on an important case of multi-commodity flow, the so-called *anycast* problem, cf. [42]. In the anycast problem, we have different commodities, one for each source node. All these commodities must be routed to an arbitrary set T of destination nodes (or from T to the source nodes). In contrast to general multi-commodity flow problems, it does not matter which commodity ends up at which destination, as long as the destination is in the set T. Commodities may route to *any* destination of the set T, hence anycast.

Using popular terminology, think of T as the set of servers of a cloud provider; customers do not care which server gets the (potentially enormous [43]) data, as long as "the data gets to the cloud". An analogous case can be made when, e.g., migrating virtual machines *inside* the cloud (i.e., anycast-based data center intra- [9] or inter-connections [18]). Furthermore, when data is requested from the cloud, the customer does not care either from which anycast-based [2] content delivery networks the (identical) data is obtained, cf. [6, 10]. For example, streaming a video to digital media player, or a mobile phone requesting a web site currently under a distributed denial-of-service (DDoS) attack [37, 38].

Applying our method of flow augmentation, we develop an efficient algorithm for consistently migrating to any desired feasible set of traffic demands. We require only one augmenting flow per commodity, minimizing network overhead. Thus, for the anycast setting, we solve all the three issues (1), (2), (3) mentioned above.

Furthermore, we show that our method can also be extended to stronger notions of consistency, by adding a polynomial number of intermediate updates to the flow migration. Lastly, we also prove that the consistent migration problem to new demands turns out to be NP-hard for unsplittable flows.

1.1. Structure of our Article

After discussing the background and related work in Section 2, we continue with the model Section 3 – where the term consistent migration is defined formally. In Section 4 we develop a technique to use augmenting flows for consistent migration in the anycast setting, before showing in Section 5 how to implement our method efficiently in practice. We also prove that our method works for stronger consistency models in Section 6, but that the corresponding problem for unsplittable flows is NP-hard, cf. Section 7. After discussing the case of general multi-commodity flows in Section 8, we conclude with Section 9.

2. Background and Related Work

To the best of our knowledge, the concept of flow augmentation has not yet been used in the context of consistent migration. Thus, we treat both topics separately, followed by a concise comparison of our work to current flow migration techniques.

2.1. Augmentation & Multi-Commodity Flows

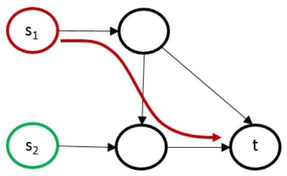
The notion of augmenting paths for single-commodity flows has been introduced in the seminal works of Ford and Fulkerson [15, 16], with their concepts influencing thousands of publications to this day. In the last decades, there has been a great amount of research regarding (multi-commodity) flow problems. We refer to the textbooks by Cormen et al. [11] and Ahuja et al. [1] for an in-depth overview.

Hu [22] studied augmenting paths for a two-commodity setting and generalized the results of Ford and Fulkerson to maximize the simultaneous flow of two commodities. By limiting the problem to just two commodities, he introduced so-called backward and forward paths, which together allow for an augmentation of the network.

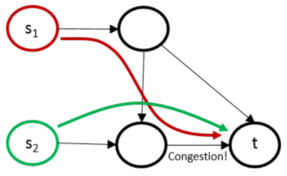
Furthermore, in 1978, shortly before the celebrated publication of the Ellipsoid method [26], Itai [23] published an improved version of Hu's two-commodity flow algorithm and showed that maximizing a two-commodity flow is as difficult as linear programming in the sense that they are polynomially equivalent.

However, while many further results were published for multi-commodity flow problems in general and augmenting path algorithms for single-commodity flow problems in particular, the application of augmenting paths to multi-commodity flow problems has been sparse.

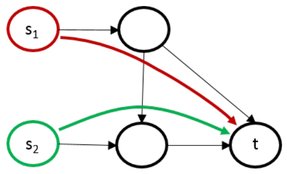
Rothfarb et al. applied augmenting paths in the following way [41]: To maximize multi-commodity flows with just one destination t, they added a logical super-source s, considered all commodities as the same commodity with new source s, and then solved the obtained single-commodity flow problem using the standard augmenting path method. Afterwards, the single-commodity flow is split into a multi-commodity flow again, using arc-chain decomposition. Since the arc-chain decomposition is independent of the initial flow, possibly all single-commodity flows are re-routed completely, even though already a small



(a) Initial network with just one flow from s_1 to t.



(b) If the lower flow is inserted first, there will be congestion.



(c) Desired flow placement with two flows of two commodities.

Figure 1: This figure depicts a small network to introduce the concept of consistency. In the above examples, all flows have a size of one and all edges have a capacity of one as well. If the SDN controller desires to migrate the network from Subfigure 1a to Subfigure 1c in order to add a flow for the second commodity outgoing from s_2, then the commodity outgoing from s_1 has to be moved first. Else, due to asynchrony, s_2 could start a flow before the last edge is free, causing congestion. The concept is defined formally in Condition (5) in Subsection 3.2.

modification might have been sufficient. Furthermore, their algorithm does not deal with the problem of asynchrony in SDNs (which is not surprising, considering the concept of SDNs was still decades away), and as thus, will induce congestion if used for migration.

2.2. SDNs & Multi-Commodity Flows

Unlike multi-commodity flow problems regarding demand satisfaction, the study of migration of flows is still in an emergent state. Perhaps it was not before the rise of Software Defined Networks (SDNs) that moving flows onto other paths became a relevant topic in practice. In SDNs, a central controller can change the behaviour of the switches in the network, allowing for, e.g., arbitrary flow allocations among the links. There is a great amount of interesting checkable properties that the controller might want to verify (for example by model checking, cf. [31, 35]), such as waypointing via firewalls, loop freedom, or network connectivity. We refer to [8] for an overview of SDN abstractions, with more recent important SDN abstractions in, e.g., [27, 36]. We furthermore refer to [14] for a recent comprehensive survey on the network update problem.

We focus our attention on preventing capacity constraint violations caused by asynchrony during the migration of flows, i.e., consistent updates for multi-commodity flows. Imagine a small example network, in which there are two directed edges, each filled to the brim with a different commodity. Due to a planned network optimization, the commodities might have to swap edges [33], i.e., each commodity will be routed along the other edge. Should this swap not be synchronized perfectly, then one commodity will migrate before the other, causing one edge to be over capacity. However, clock synchronization for simultaneous updates in the switches is far from perfect, and even if it was, current industry switches might straggle [24] – taking up to 100x longer than average to implement updates [25]. The recent works of Kuzniar et al. [29] and He et al. [20] give further evidence for the asynchrony of switch updates in practice.

Hong et al. [21] propose to always leave a fraction s of the capacity of each edge unutilized s.t. when a migration of flows has to occur, it can be performed in $\lceil 1/s \rceil$ updates. However, this approach fails when some edges are used at full capacity. Thus, the authors also present a linear program that essentially tries if a migration is possible in $1, 2, 4, 8, \dots$ update steps, possibly temporarily re-routing flows to arbitrary edges in the network. A similar approach is also used in a datacenter setting by Liu et al. [30]. It can be decided in polynomial time if a consistent migration in their model is possible [5], but the number of necessary updates can be arbitrarily large.

The work of Jin et al. [25] follows a different approach: They build a combinatorial graph from the current and desired flow allocations, and try to find an ordering of how to move the flows s.t. the individual updates are consistent. Their search algorithm cannot guarantee to find a solution if one exists, which is why they heuristically opt to break demand constraints temporarily if the flows have been migrated in such a way that no local migration progress is possible.

Temporal breaking of demand constraints and congestion is also an idea implemented by Jain et al. [24]: Instead of considering them at all, they aim to migrate as fast as possible in order to minimize the effects of capacity constraint violations.

A related but algorithmically more intricate technique is introduced by Mizrahi, Rottenstreich, and Moses [32, 33, 34]: They schedule updates at synchronized time slots, instead of just applying them as fast as possible. In their work, they show that time-based updates can be a powerful mechanism in SDNs, and make a case for its inclusion in standard protocols such as OpenFlow.

Finally, Reitblatt et al. [39, 40] deal with asynchrony by inserting version numbers into each packet: With both old and new rules implemented in the network at the same time, a packet will always be handled by one of the two rule sets, but never a mix of them – a property they call *per-packet consistency*. They extend their work to *per-flow consistency* by creating essentially multiple "versioned" distinct flows per commodity. Unlike most other work (including ours), they can guarantee that all packets in the same flow are forwarded according to the same rules, an important element for, e.g., load balancers or waypointing. Even though their method is not aimed at congestion per se, it still prevents many issues causing capacity constraint violations.

2.2.1. Comparison with Current Flow Migration Techniques

The current (congestion-oriented) flow migration techniques for SDNs can thus be roughly classified into two approaches:

1. Minimize congestion periods by applying updates fast [24] and/or synchronized well [32].

2. Minimize congestion by carefully scheduling sequential updates, computed by linear programming approaches, e.g., [21].

Approach 1 can be seen as orthogonal to our work. By developing and optimizing protocols and hardware to be fast and failure resistant [24], and by synchronizing independent node updates well, e.g., [32], transient congestion cannot be removed in a theoretically correct way, but rather avoided in a best-effort approach. While our work avoids congestion in a theoretically sound way, we usually need more than one single network update, making our approach slower as we aim beyond a best-effort goal. However, approach 1 can be combined with our work: We schedule the updates as described in this article, but the practical implementation is enhanced by faster execution times.

Approach 2 is the one most related to our work. The main difference between, e.g., *SWAN* [21], is the number of needed updates in the worst case. While we can bound the number of updates to be polynomial due to only needing one augmenting flow per commodity, the seminal model of *SWAN* [21] has cases where the minimum number of updates is *unbounded*. Furthermore, the number of variables of the LP used in [21] grows with the number of updates needed – meaning that for extreme situations, the (computational) runtime of *SWAN* is no longer polynomial in the input size. Lastly, our work also extends to stronger consistency models, cf. Section 6.

We note that at a fundamental level, the price we have to pay for the listed advantages is that we can only operate in a limited scenario, namely multi-commodity single-destination flows. Furthermore, most of the discussed (practical) literature is heavily fine-tuned in their implementation for real-world use: Our work is more at a conceptual level, proving theoretical advances which can be the building blocks of future works to come.

3. Model

3.1. Network Flows

We consider a network as a directed graph with edge capacities. For the definition of a multi-commodity flow, we first need to define a single-commodity flow via the usual flow constraints:

Definition 1. *Let $G = (V, E)$ be a simple connected directed graph with $|V| = n$ nodes and $|E| = m$ edges. Denote the set of edges outgoing from a node $v \in V$ by $\boldsymbol{out}(v)$ and the set of incoming edges by $\boldsymbol{in}(v)$. A* network *is a pair $N = (G, c)$ where $c : E \to \mathbb{R}_+$ is a map assigning each edge a positive capacity. We call a pair of distinct nodes $s, t \in V$ a* commodity *K. We define a* single-commodity flow *for K as a map $F : E \to \mathbb{R}_{\geq 0}$ s.t.*

$$F(e) \leq c(e) \qquad \text{for all } e \in E, \tag{1}$$

$$\sum_{e \in \boldsymbol{out}(v)} F(e) = \sum_{e \in \boldsymbol{in}(v)} F(e) \qquad \text{for all } v \in V \setminus \{s, t\}, \tag{2}$$

$$\sum_{e \in \boldsymbol{out}(s)} F(e) = d_F = \sum_{e \in \boldsymbol{in}(t)} F(e), \tag{3}$$

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

We now extend the definition of a single flow to multi-commodity anycast, for which we encompass all nodes in T in a single node t. Our results can be applied analogously to the "edge reversed" model variant, where an arbitrary set of source nodes S routes multiple commodities to their assigned distinct destinations.

Definition 2. *Let N be a network and let $K_i = (s_i, t)$ be commodities where $s_1, s_2, \ldots, s_k, t \in V$ are pairwise distinct nodes. Then we call a tuple $\mathcal{K} = (K_1, K_2, \ldots, K_k)$ a* multi-commodity*. Let F_i be a flow for the commodity K_i for all $1 \leq i \leq k$. A tuple $\mathcal{F} = (F_1, \ldots, F_k)$ is called a* multi-commodity flow *for $\mathcal{K}$ if*

$$\sum_{i=1}^{k} F_i(e) \leq c(e) \qquad \text{for all } e \in E. \tag{4}$$

We will assume in the following that all considered flows are cycle-free. In the presented algorithms, cycles may appear temporarily, but will always be explicitly removed. In particular, this implies that $\sum_{e \in \mathtt{in}(s)} F(e) = 0 = \sum_{e \in \mathtt{out}(t)} F(e)$ if $d_F > 0$. For the sake of simplicity, we assume that this equation also holds for the case of $d_F = 0$ (which is a natural assumption from a practical point of view as we want to study the traffic from a source s to a destination t).

Definition 3. *We call a flow F* cycle-free*, if there is no directed cycle C in N s.t. $F(e) > 0$ for all $e \in C$.*

Lastly, we will need the concept of a *partial flow* in the following sections:

Definition 4. *Let F be a single-commodity flow for the commodity $K = (s, t)$ and let $v \in V$. We call a flow F' for the commodity $K' = (v, t)$ a* partial flow of F starting in v *if the following conditions hold:*

$$F'(e) \leq F(e) \qquad \text{for all } e \in E$$
$$F'(e) = F(e) \qquad \text{for all } e \in \mathtt{out}(v)$$

Furthermore, we call a flow F'' for the commodity K a subflow *of F if $F''(e) \leq F(e)$ for all $e \in E$.*

Note that, since we assume all considered flows to be cycle-free, all the traffic leaving v (in the flow F) must finally end up in t which implies that (for each $v \in V$) there exists a partial flow of F starting in v.

3.2. Consistent Migration

We define the term *consistent migration* (i.e., not violating edge capacity constraints due to asynchrony in node updates) as proposed in [21, 25, 30], who implemented and evaluated the consistent migration of flows in SDNs with multiple production data center networks across three continents with tens of thousands of servers. We note that the term consistent updates is sometimes used for different concepts in SDNs, e.g., in [7, 13]. We refer to Figure 1 for an introductory example.

Definition 5. *Let N be a network and let $\mathcal{F} = (F_1, \ldots, F_k)$, $\mathcal{F}' = (F'_1, \ldots, F'_k)$ be multi-commodity flows for the multi-commodity $\mathcal{K}$ s.t. $d_{F_i} \leq d_{F'_i}$, $1 \leq i \leq k$. The tuple $(N, \mathcal{F}, \mathcal{F}')$ is a* consistent migration update *from $\mathcal{F}$ to $\mathcal{F}'$ if*

$$\sum_{i=1}^{k} \max\left(F_i(e), F'_i(e)\right) \leq c(e) \qquad \text{for all } e \in E. \tag{5}$$

A consistent migration *from $\mathcal{F}$ to $\mathcal{F}^*$ is a sequence of consistent migration updates $(N, \mathcal{F}, \mathcal{F}_1), (N, \mathcal{F}_1, \mathcal{F}_2), \ldots, (N, \mathcal{F}_j, \mathcal{F}^*)$.*

Note that for each $K \in \mathcal{K}$ the demand of K w.r.t $\mathcal{F}, \mathcal{F}_1, \ldots, \mathcal{F}_j, \mathcal{F}^*$ is non-decreasing. If the demand of flows was smaller in $\mathcal{F}'$, then one would drop corresponding parts of the flows before migration.

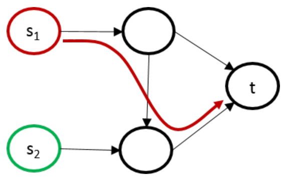
4. Augmenting Flows for Multiple Commodities

In the case of one source and one destination, it is well-known [16] how to use an obtained augmenting path P in order to transform a given flow into a new enhanced flow whose size is increased by the "capacity" of P. When we have multiple sources, the "standard" augmenting path does not account for moving multiple commodities at once, since it is only defined to modify the flow of a single commodity.

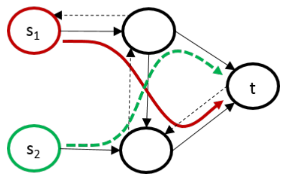
In the following Definition 6, we define an augmenting flow for the case of a multi-commodity flow where we have multiple sources (but only one destination). The augmenting flow may use edges from the residual network, which is created by adding a back-edge in the reverse direction for every edge with some flow on it, cf. the dashed edges in Subfigure 2b. Note that while the augmenting flow may use these back-edges, there will be never any "real" flow routed over these edges, as they are not part of the physical network and just used for our algorithms.

We further introduce the notion of a *farthest back-edge* which is a back-edge used by the augmenting flow "after which" the augmenting flow only uses forward edges.

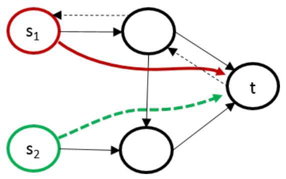
Definition 6. *Let N be a network and let $\mathcal{F}$ be a multi-commodity flow for the multi-commodity $\mathcal{K}$. We denote by $\overline{G}$ the graph obtained from G by adding an edge $e^* = (v, u)$ to G for any edge $e = (u, v) \in E$. Let E^* be the set of all newly added edges. If an edge $e^* \in E^*$ starts and ends in the same vertices as some edge in E, we still consider them as distinct edges. Set $\overline{N} := \left(\overline{G}, \overline{c}\right)$ where $\overline{c}(e^*) := \overline{c}(e) := c(e)$ for all $e \in E$. Let $K \in \mathcal{K}$. We call a cycle-free (single-commodity) flow F_A for K in $\overline{N}$ an* augmenting flow *w.r.t. $\mathcal{F}$ if $F_A(e) \leq c(e) - \sum_{F \in \mathcal{F}} F(e)$ and $F_A(e^*) \leq \sum_{F \in \mathcal{F}} F(e)$ for all $e \in E$. Set $E^*_{F_A} := \{e^* \in E^* | F_A(e^*) > 0\}$. We call an edge $(u, v) \in E^*_{F_A}$ a* farthest back-edge *if there is no path P from v to t s.t. for all edges $e \in P$ we have $F_A(e) > 0$ and there is an edge $e^* \in P$ with $e^* \in E^*_{F_A}$. Since F_A is cycle-free, such a farthest back-edge exists if $E^*_{F_A} \neq \emptyset$.*



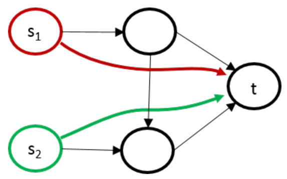
(a) Initial network with just one flow from s_1 to t. Currently, there is no space for a flow from s_2 to t, the red flow needs to be moved first.



(b) An augmenting flow for s_2 is found from s_2 to t, using a dashed edge in $\overline{G}$ that pushes the red flow back to the top.



(c) After the red flow has been pushed to the top, the resulting green augmenting flow uses only "real" edges from the network. Thus, in a next step, it can be replaced with a proper "real" flow.



(d) The resulting flows are feasible and use only edges in the "real" network, none of the (hidden) dashed ones from $\overline{G}$.

Figure 2: In this small introductory example network to illustrate augmenting flows, all edges have a capacity of one and all flows have a size of one. The solid edges are the "real" edges in the network N, while the dashed edges in Subfigure 2b exist just in $\overline{G}$: A reverse edge for every edge with some flow on it. Dashed edges from $\overline{G}$ are never used for routing, they are just used to find augmenting flows. If the task is to add a flow from s_2 to t, then one searches for an (augmenting) flow from s_2 to t – but not just in N, the dashed edges from $\overline{G}$ are allowed as well.

Note that in this article, such an augmenting flow always "belongs" to a specific commodity K contained in the respective multi-commodity.

We develop a technique in the following Algorithm 1 to transform the given multi-commodity flow step by step into a multi-com-modity flow where the flow size for K is increased by the size of the augmenting flow, see Theorem 2. A very small introductory example is given in Figure 2. We show that the transformation steps correspond to consistent migration updates, thus proving that the new (multi-commodity) flow can be obtained from the old one by a consistent migration.

The general idea is as follows: Given a multi-commodity flow and an augmenting flow, Algorithm 1 will perform a consistent migration update (Lemma 6) in the network.[1] Essentially, one execution of Algorithm 1 will process one

[1]The calculation of the corresponding augmenting flow for Algorithm 1 is discussed in Section 5. Essentially, we will calculate one augmenting flow per commodity that needs to be augmented, and apply Algorithm 1 sequentially.

edge of the augmenting flow. As the augmenting flow can have at most $m = |E|$ edges, the augmenting flow will be inserted consistently after a linear number of iterations of the algorithm (Theorem 2). We refer to Figure 3 for an advanced illustration of Algorithm 1.

Algorithm 1. *Let N be a network and $\mathcal{F} = (F_1, \dots, F_k)$ be a multi-commodity flow for the multi-commodity $\mathcal{K} = (K_1, \dots, K_k)$. Let F_A be an augmenting flow w.r.t. $\mathcal{F}$ for some commodity $(s,t) = K \in \mathcal{K}$. Let $E^*_{F_A}$ be non-empty and let $(u_0, v_0) = e^*_0 \in E^*_{F_A}$ be a farthest back-edge. Let $F_{k_1}, ..., F_{k_q} \in \mathcal{F}, k_1 < \dots < k_q$ be the flows[2] which assign the edge e_0 (i.e., the edge which induced the adding of e^*_0 to G) a non-zero value, i.e., the flows which are present on this edge. Let r be the smallest index such that $\sum_{z=1}^{r} F_{k_z}(e_0) \geq F_A(e^*_0)$. Set $U := F_A(e^*_0) - \sum_{z=1}^{r-1} F_{k_z}(e_0)$. We migrate to a new multi-commodity flow $\mathcal{F}' = (F'_1, \dots, F'_k)$ for $\mathcal{K}$ and a new augmenting flow F'_A w.r.t. $\mathcal{F}'$ as follows:*

1. *Begin by setting $F'_y(e) := F_y(e)$ for all $e \in E$ and all $1 \leq y \leq k$, and $F'_A(e) := F_A(e)$ for all $e \in E \cup E^*$.*
2. *Redefine $\mathcal{F}'$ on e_0 and F'_A on e^*_0: Set $F'_{k_z}(e_0) := 0$ for all $1 \leq z \leq r-1$, $F'_{k_r}(e_0) := F'_{k_r}(e_0) - U$, and $F'_A(e^*_0) := 0$.*
3. *Choose a partial flow of F_A starting in v_0 and choose a subflow F_a (of this partial flow) of size $F_A(e^*_0)$. (Note that $F_a(e^*) = 0$ for all $e^* \in E^*$ because e^*_0 is a farthest back-edge.) Decompose F_a in r subflows $F_a^{(1)}, \dots, F_a^{(r)}$ of sizes $F_{k_1}(e_0), \dots, F_{k_{r-1}}(e_0), U$ such that, for each edge $e \in E$, we have $\sum_{z=1}^{r} F_a^{(z)}(e) = F_a(e)$. Now set $F'_{k_z}(e) := F'_{k_z}(e) + F_a^{(z)}(e)$ for all $1 \leq z \leq r$ and all $e \in E$, and set $F'_A(e) := F'_A(e) - F_a(e)$ for all $e \in E$.*
4. *For all $1 \leq z \leq r$, choose a partial flow of F_{k_z} starting in u_0 and choose a subflow $F^{(z)}$ (of this partial flow) of size $F_{k_z}(e_0)$ if $z \neq r$ and of size U if $z = r$. Then replace these subflows by the augmenting flow, i.e., set $F'_{k_z}(e) := F'_{k_z}(e) - F^{(z)}(e)$ for all $1 \leq z \leq r$ and all $e \in E$, and set $F'_A(e) := F'_A(e) + \sum_{z=1}^{r} F^{(z)}(e)$ for all $e \in E$.*
5. *Replace possible cycles for flows in $\mathcal{F}'$ by cycles for F'_A: If there is some flow $F' \in \mathcal{F}'$ which is not cycle-free, then find a (directed) cycle C s.t. $F'(e) > 0$ for all $e \in C$. Set $F'(e) := F'(e) - \min_{e' \in C} F'(e')$ for all $e \in C$, thus "removing" the cycle, and set $F'_A(e) := F'_A(e) + \min_{e' \in C} F'(e')$ for all $e \in C$. Continue removing (and replacing) cycles in this way (for all flows in $\mathcal{F}'$) until there are no cycles left in $\mathcal{F}'$. (Note that the removal of a cycle implies that there is some edge e which changes in the process from $F'(e) > 0$ to $F'(e) = 0$. Thus, all flows contained in $\mathcal{F}'$ are cycle-free after removing at most $O(mk)$ cycles.)*

[2]We note that one could order the flows by decreasing size to further decrease the amount of flows being re-routed.

6. *Remove possible cycles for F'_A: First remove cycles for the flow F'_A which consist only of an edge $e \in E$ and its corresponding edge $e^* \in E^*$, until no such cycles remain. Subsequently, remove arbitrarily chosen cycles for F'_A iteratively until F'_A is cycle-free. Analogously to the above, at most $O(m)$ cycles need to be removed.*

In the following, we will state and prove various lemmas to lastly prove Theorem 2 in this section. We begin with Lemma 1 and Lemma 2, which state that the new augmenting flow F'_A will not violate the capacity constraints set in Definition 6:

Lemma 1. *$F'_A(e) + \sum_{F' \in \mathcal{F}'} F'(e) \leq c(e)$ for all $e \in E$.*

Proof. Since F_A is an augmenting flow w.r.t. $\mathcal{F}$, it holds that $F_A(e) + \sum_{F \in \mathcal{F}} F(e) \leq c(e)$ for all $e \in E$. Thus, after step 1 we have $F'_A(e) + \sum_{F' \in \mathcal{F}'} F'(e) \leq c(e)$ for all $e \in E$. In step 2, $F'_A(e) + \sum_{F' \in \mathcal{F}'} F'(e)$ remains unchanged for all $e \in E$. In steps 3, 4, and 5 this is also the case since for each $e \in E$, $F'_A(e)$ is diminished by the same amount by which $\sum_{F' \in \mathcal{F}'} F'(e)$ grows larger, resp. vice versa. As the cycle removals in step 6 can only diminish the above sum, we obtain Lemma 1. □

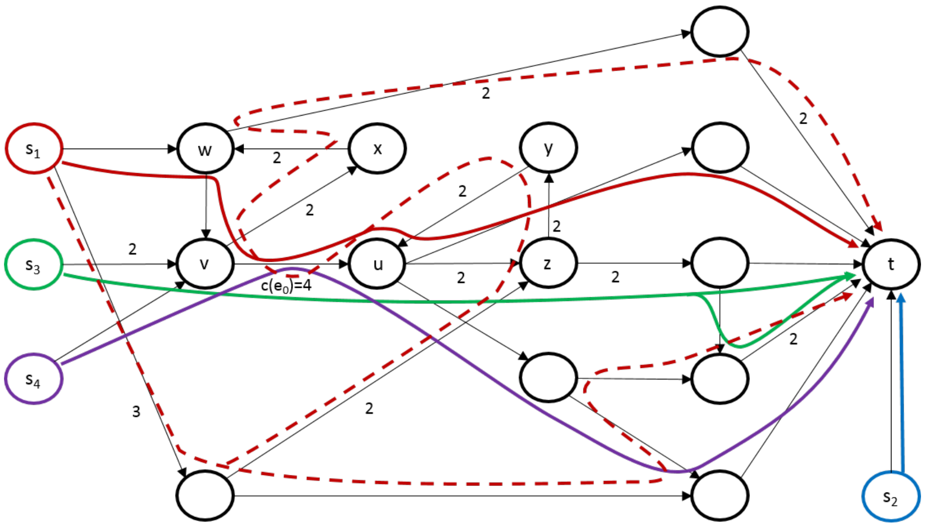
Lemma 2. *$F'_A(e^*) \leq \sum_{F' \in \mathcal{F}'} F'(e)$ for all $e \in E$.*

Proof. Since F_A is an augmenting flow w.r.t. the multi-commodity flow $\mathcal{F}$, it holds for $F'_A(e^*)$ that $F'_A(e^*) \leq \sum_{F' \in \mathcal{F}'} F'(e)$ for all $e \in E$ after step 1. In step 2, both $\sum_{F' \in \mathcal{F}'} F'(e_0)$ and $F'_A(e_0^*)$ are diminished by $F'_A(e_0^*)$, while nothing changes for the edges $e \neq e_0$. In step 3, $\sum_{F' \in \mathcal{F}'} F'(e)$ cannot decrease, while $F'_A(e^*)$ cannot be increased. Thus, at this point, $F'_A(e^*) \leq \sum_{F' \in \mathcal{F}'} F'(e)$ still holds for all $e \in E$. In step 4, the left hand side of the inequality is not increased, since $e^* \notin E$. As in this step $F'_A(e)$ is increased by the same amount by which $\sum_{F' \in \mathcal{F}'} F'(e)$ is diminished, we obtain $F'_A(e^*) \leq \sum_{F' \in \mathcal{F}'} F'(e) + F'_A(e)$, for all $e \in E$ after step 4. By an analogous argument, this new inequality holds also after step 5. After removing the "small" cycles in the first part of step 6 we have $F'_A(e^*) = 0$ or $F'_A(e) = 0$ for all $e \in E$, while the new inequality still holds. As the subsequent cycle removals in the second part of step 6 cannot decrease $\sum_{F' \in \mathcal{F}'} F'(e)$ to less than 0, the inequality given in Lemma 2 holds for all $e \in E$ with $F'_A(e^*) = 0$. Thus, consider the edges $e \in E$ with $F'_A(e) = 0$. For these edges, $F'_A(e^*) \leq \sum_{F' \in \mathcal{F}'} F'(e) + F'_A(e)$ implies $F'_A(e^*) \leq \sum_{F' \in \mathcal{F}'} F'(e)$ which yields the desired statement of Lemma 2 since the cycle removals in the second part of step 6 cannot decrease $\sum_{F' \in \mathcal{F}'} F'(e)$. □

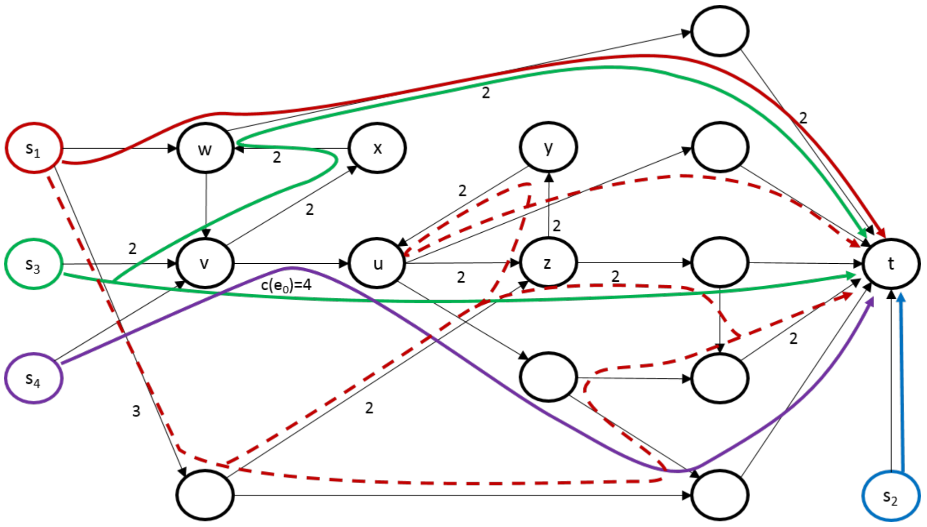
Next, in Lemma 3 and 4, we show that the flows adhere to the flow conditions and keep their demand unchanged.

Lemma 3. *F'_A is a (single-commodity) flow for the commodity K and $d_{F'_A} = d_{F_A}$.*

Proof. We first show that F'_A is non-negative on all edges in $E \cup E^*$ and then that F'_A satisfies Conditions (1)–(3) from Definition 1.



(a) Multi-commodity flow $\mathcal{F} = (F_1, F_2, F_3, F_4)$ and augmenting flow F_A, *before* applying Algorithm 1 w.r.t. e_0^*.



(b) Multi-commodity flow $\mathcal{F}' = (F_1', F_2', F_3', F_4')$ and augmenting flow F_A, *after* applying Algorithm 1.

Figure 3: In this example network, all unmarked edges have a capacity of one. The green flow F_3 starting in s_3 has a size of two, all other solid flows have a size of one. The dashed augmenting flow F_A for the commodity K_1 starting in s_1 has a size of three. In Subfigure 3a, the (not drawn) edge $(u, v) = e_0^*$ in $\overline{N}$ is a farthest back-edge. When executing Algorithm 1, we obtain $q = 3$, $k_1 = 1, k_2 = 3, k_3 = 4$, $F_A(e_0^*) = 2$, $r = 2$, and $U = 1$. A part of the augmenting flow is re-routed in the node u via former paths of F_1 and F_3, whereas F_1 and half of F_3 are re-routed in the node v via a former path of the augmenting flow. The occurring cycles wvx and zyu are removed afterwards by Algorithm 1.

The only step where F'_A can switch to a negative value on some edge $e \in E \cup E^*$ is step 3 and this can only be the case if $e \in E$. But since F_a is a subflow of a partial flow of F_A, we have $F_a(e) \leq F'_A(e)$ for all $e \in E$ and $F'_A(e)$ remains non-negative in step 3. Note that at the beginning of step 3, it holds that $F'_A(e) = F_A(e)$ for all $e \in E$.

By an analogous argument, $F'_y(e)$ is non-negative for all $1 \leq y \leq k$ and all $e \in E$. Since, in addition, $F'_A(e^*)$ is never increased in steps 2 to 6 for all $e^* \in E^*$ (which implies $F'_A(e^*) \leq F_A(e^*)$), Condition (1) holds due to Lemma 1 and Definition 6.

We will now show that Conditions (2) and (3), i.e., flow conservation and demand satisfaction, are maintained. Consider $D_A(v) := \sum_{e \in \mathtt{in}(v)} F'_A(e) - \sum_{e \in \mathtt{out}(v)} F'_A(e)$ for all $v \in V$. After step 1, $D_A(v) = 0$ for all $v \in V \setminus \{s,t\}$, and $-D_A(s) = D_A(t) = d_{F_A}$. However, in step 2, $D_A(u_0)$ is increased by $F'_A(e^*_0)$ and $D_A(v_0)$ is diminished by $F'_A(e^*_0)$. In step 3, $D_A(v_0)$ is increased by $F'_A(e^*_0)$, $D_A(t)$ gets diminished by $F'_A(e^*_0)$, and $D_A(v)$ remains unchanged for all other nodes v. In step 4, $D_A(u_0)$ is diminished by $F'_A(e^*_0)$, $D_A(t)$ gets increased by $F'_A(e^*_0)$, and $D_A(v)$ remains unchanged for all other nodes v. Lastly, the replacement of cycles in step 5 and the removal of cycles in step 6 do not change any $D_A(v)$. Thus, $D_A(v) = 0$ for all $v \in V \setminus \{s,t\}$, and $-D_A(s) = D_A(t) = d_{F_A}$. Since F'_A is cycle-free, this implies $\sum_{e \in \mathtt{out}(s)} F'_A(e) = d_{F_A} = \sum_{e \in \mathtt{in}(t)} F'_A(e)$, i.e., $d_{F'_A} = d_{F_A}$. □

Lemma 4. *$\mathcal{F}'$ is a multi-commodity flow for $\mathcal{K}$ and $d_{F'_y} = d_{F_y}$ for all $1 \leq y \leq k$.*

Proof. Lemma 4 follows by a proof analogous to the proof of Lemma 3. Note that Condition (4) holds due to Lemma 1. □

Combining Lemmas 1 to 4, we obtain the following corollary:

Corollary 1. *F'_A is an augmenting flow w.r.t. $\mathcal{F}'$.*

Furthermore, we need to prove that Algorithm 1 actually makes progress, i.e., at least one of the edges e^* has an augmenting flow of zero afterwards.

Lemma 5. *The number of edges $e^* \in E^*$ with $F'_A(e^*) > 0$ is strictly smaller than the number of edges $e^* \in E^*$ with $F_A(e^*) > 0$.*

Proof. As observed in the proof of Lemma 3, we have $F'_A(e^*) \leq F_A(e^*)$ for all $e^* \in E^*$. Thus, $F'_A(e^*) > 0$ implies $F_A(e^*) > 0$. Moreover, $F_A(e^*_0) > 0$, but $F'_A(e^*_0) = 0$ (due to step 2). The result follows. □

Lastly, we show that the update performed by Algorithm 1 is actually consistent:

Lemma 6. *$(N, \mathcal{F}, \mathcal{F}')$ is a consistent migration update.*

Proof. By Lemma 4, we only have to show that Condition 5 holds, i.e., that for all $e \in E$: $\sum_{y=1}^{k} \max\left(F_y(e), F'_y(e)\right) \leq c(e)$. Let e be an arbitrary edge in E. We observe that, for all $1 \leq y \leq k$, the only step (after setting $F'_y(e) := F_y(e)$)

where $F'_y(e)$ gets possibly increased is step 3. Moreover, a positive increase in step 3 is only possible if $y = k_z$ for some $1 \leq z \leq r$. More specifically, we have $F'_{k_z}(e) \leq F_{k_z}(e) + F_a^{(z)}(e)$ after step 6 for all $1 \leq z \leq r$. Since $F_a^{(z)}(e) \geq 0$ for all $1 \leq z \leq r$, we obtain

$$\sum_{y=1}^{k} \max\left(F_y(e), F'_y(e)\right) \leq \sum_{y=1}^{k} F_y(e) + \sum_{z=1}^{r} F_a^{(z)}(e) = F_a(e) + \sum_{y=1}^{k} F_y(e) \enspace .$$

Since F_a is a subflow of a partial flow of F_A (compare step 3), we have $F_a(e) \leq F_A(e)$. Thus,

$$\sum_{y=1}^{k} \max\left(F_y(e), F'_y(e)\right) \leq F_A(e) + \sum_{y=1}^{k} F_y(e) \leq c(e) \enspace .$$

The last inequality follows since F_A is an augmenting flow w.r.t. $\mathcal{F}$. □

We can now prove Theorem 2, which states that we can update consistently to the new augmented flow in just a linear number of updates[3], resulting from applying the augmenting flow:

Theorem 2. *Let N be a network and let $\mathcal{F} = (F_1, \ldots, F_k)$ be a multi-commodity flow for the multi-commodity $\mathcal{K} = (K_1, \ldots, K_k)$. Let F_A be an augmenting flow w.r.t. $\mathcal{F}$ for the commodity $(s,t) = K_x \in \mathcal{K}$ where $1 \leq x \leq k$. Then there is a multi-commodity flow $\mathcal{F}^*$ for $\mathcal{K}$ s.t. $d_{F_x^*} = d_{F_x} + d_{F_A}$ and $d_{F_y^*} = d_{F_y}$ for all $1 \leq y \leq k$ with $y \neq x$. Moreover, there is a consistent migration from $\mathcal{F}$ to $\mathcal{F}^*$, consisting of at most $m+1$ consistent migration updates.*

Proof. W.l.o.g., let $h \in \mathbb{N}$ be the number of times that the steps 1 to 6 of Algorithm 1 are performed until, for the resulting augmenting flow F_A^h w.r.t. the resulting multi-commodity flow $\mathcal{F}^h$, there is no edge $e^* \in E^*$ with $F_A^h(e^*) > 0$. Note that, due to Lemma 5, it holds that $h \leq m$. Thus, by Lemmas 4 and 6, every one of the h iterations of the steps 1 to 6 corresponds to a consistent migration update. Moreover, $d_{F_y^h} = d_{F_y}$ for all $1 \leq y \leq k$, by Lemma 4, and $F_A^h(e) + \sum_{F^h \in \mathcal{F}^h} F^h(e) \leq c(e)$ for all $e \in E$, by Lemma 1. Therefore, increasing $F_x^h(e)$ by $F_A^h(e)$ for all $e \in E$ corresponds to a consistent migration update and the resulting multi-commodity flow $\mathcal{F}^*$ satisfies the conditions given in Theorem 2. □

5. Augmenting the Network in Practice with Algorithm 1

A standard approach in the single-commodity case for increasing the size of a flow is to compute augmenting paths, apply them to the network, and

[3]We note that other mechanisms such as, e.g., *SWAN* [21] or *zUpdate* [30], do not give *any* bound on the number of updates needed for consistent migration.

iterate this process until the desired demand is reached, if possible. However, this method requires a lot of updates in the network itself, as the number of augmenting paths needed can be linear in the number of edges. Due to the fact that we augment our multi-commodity flow in Section 4 with a *flow* instead of a *path*, in our framework just one augmenting flow per commodity suffices to satisfy any possible new demands, as we show in this section.

While a linear programming solution does not show how to migrate the network consistently, we can use LPs to compute the augmenting flows needed for the consistent migration. For our method we will first need the notion of *difference flows*, which are flows obtained by "subtracting" a multi-commodity flow from another.

Definition 7. *Let N be a network, let $\mathcal{K} = (K_1, \dots, K_k)$ be a multi-commodity, and fix some $i \in \mathbb{N}$ with $1 \leq i \leq k$. Let $\mathcal{F} = (F_1, \dots, F_k), \mathcal{F}' = (F'_1, \dots, F'_k)$ be multi-commodity flows for $\mathcal{K}$ with $d_{F_i} < d_{F'_i}$ and $d_{F_j} = d_{F'_j}$ for all $1 \leq j \leq k$, $j \neq i$. We define a* difference flow *$Z^{\mathcal{F},\mathcal{F}'}$ for $\mathcal{F}$ and $\mathcal{F}'$ in $\overline{N}$ as follows: First, for all $e \in E$: i) If $\sum_{y=1}^{k} F'_y(e) - \sum_{y=1}^{k} F_y(e) \geq 0$, then $Z^{\mathcal{F},\mathcal{F}'}(e) := \sum_{y=1}^{k} F'_y(e) - \sum_{y=1}^{k} F_y(e)$ and $Z^{\mathcal{F},\mathcal{F}'}(e^*) := 0$. ii) If $\sum_{y=1}^{k} F'_y(e) - \sum_{y=1}^{k} F_y(e) < 0$, then set $Z^{\mathcal{F},\mathcal{F}'}(e^*) := -\left(\sum_{y=1}^{k} F'_y(e) - \sum_{y=1}^{k} F_y(e)\right)$ and $Z^{\mathcal{F},\mathcal{F}'}(e) := 0$. Second, remove cycles until $Z^{\mathcal{F},\mathcal{F}'}$ is cycle-free.*

A difference flow is also an augmenting flow:

Lemma 7. *Let $Z^{\mathcal{F},\mathcal{F}'}$ be a difference flow for $\mathcal{F}$ and $\mathcal{F}'$ with $d_{F_i} < d_{F'_i}$. Then, $Z^{\mathcal{F},\mathcal{F}'}$ is an augmenting flow w.r.t. $\mathcal{F}$ for the commodity K_i of size $d_{F'_i} - d_{F_i} > 0$.*

Proof. Recall that $\mathcal{F}$ and $\mathcal{F}'$ are multi-commodity flows in N.

We start by checking the conditions for an augmenting flow given in Definition 6. For all $e \in E$, we have $Z^{\mathcal{F},\mathcal{F}'}(e) \leq \sum_{y=1}^{k} F'_y(e) - \sum_{y=1}^{k} F_y(e) \leq c(e) - \sum_{y=1}^{k} F_y(e)$ in case i), and $Z^{\mathcal{F},\mathcal{F}'}(e) = 0 \leq c(e) - \sum_{y=1}^{k} F_y(e)$ in case ii). For all $e^* \in E^*$, we have $Z^{\mathcal{F},\mathcal{F}'}(e^*) = 0 \leq \sum_{y=1}^{k} F_y(e)$ in case i), and $Z^{\mathcal{F},\mathcal{F}'}(e^*) \leq -\left(\sum_{y=1}^{k} F'_y(e) - \sum_{y=1}^{k} F_y(e)\right) \leq \sum_{y=1}^{k} F_y(e)$ in case ii). Note that removing cycles can never increase the flow on any edge.

As $Z^{\mathcal{F},\mathcal{F}'}$ is cycle-free, it is only left to show that $Z^{\mathcal{F},\mathcal{F}'}$ is a single-commodity flow for K_i in $\overline{N}$ of size $d_{F'_i} - d_{F_i}$. Condition (1) follows directly from the previous considerations. The definition of $Z^{\mathcal{F},\mathcal{F}'}$ ensures that Condition (2) is satisfied for all nodes except s_i and t. Note that removing cycles does not change the difference between the amount of outgoing and incoming flow for a node.

As $d_{F'_i} - d_{F_i} > 0$ holds due to the construction of $Z^{\mathcal{F},\mathcal{F}'}$ and as all cycles were removed from $Z^{\mathcal{F},\mathcal{F}'}$, we obtain that $\sum_{e \in \mathtt{out}(s_i)} Z^{\mathcal{F},\mathcal{F}'}(e) = d_{F'_i} - d_{F_i} = \sum_{e \in \mathtt{in}(t)} Z^{\mathcal{F},\mathcal{F}'}(e)$, i.e., Condition (3) holds and $Z^{\mathcal{F},\mathcal{F}'}$ is an augmenting flow for the commodity K_i of size $d_{F'_i} - d_{F_i} > 0$. □

In the following Algorithm 2, we will show how any desired demands can be obtained by consistent migration, if there is a multi-commodity flow satisfying these demands.

Algorithm 2. *Let N be a network and let $\mathcal{F}$ be a multi-commodity flow for the multi-commodity $\mathcal{K}$. Let $(d_1, \ldots, d_k)$ be a vector of demands s.t. i) there exists a multi-commodity flow for $\mathcal{K}$ satisfying these demands, and ii) $d_1 \geq d_{F_1}, \ldots, d_k \geq d_{F_k}$.*

1. *Compute a multi-commodity flow $\mathcal{F}_1'$ with a demand vector of $(d_1, d_{F_2}, \ldots, d_{F_k})$ using an LP.*
2. *Compute the difference flow $Z^{\mathcal{F},\mathcal{F}_1'}$.*
3. *Augment $\mathcal{F}$ with $Z^{\mathcal{F},\mathcal{F}_1'}$ by using Algorithm 1 iteratively until no more back-edges exist for the resulting augmenting flow F_A. Then, replace F_A by a flow of commodity K_1 and eliminate all cycles for commodity K_1, thereby obtaining some flow $\mathcal{F}_1$ with a demand vector of $(d_1, d_{F_2}, \ldots, d_{F_k})$.*
4. *Iterate the above three steps for the remaining commodities $K_2, \ldots, K_k$, thereby obtaining the flows $\mathcal{F}_2, \ldots, \mathcal{F}_k$ with the demand vectors of $(d_1, d_2, d_{F_3}, \ldots, d_{F_k})$, $\ldots$, $(d_1, d_2, d_3, \ldots, d_{k-1}, d_{F_k})$, $(d_1, \ldots, d_k)$.*

Corollary 3. *Algorithm 2 performs a consistent migration from $\mathcal{F}$ to some multi-commodity flow with a demand vector of $(d_1, \ldots, d_k)$, using only k augmenting flows.*

We note that Algorithm 2 can be used for any imaginable purpose, as long as the respective desired demand vector (for which some flow exists) can be computed. Common examples in practice are maximizing the sum of all commodities or reaching max-min fairness. The respective desired demand vectors can be computed with an LP, cf., e.g., [1][11]. If the computation time is an issue as well, one can also resort to approximation algorithms with a better runtime [19].

Furthermore, the actual updates performed in the network itself are expensive, while "off-line" computations are cheap regarding the execution time in SDNs, rendering the computation overhead induced by the LPs to be bearable in practice.

6. A Stronger Consistency Model

In Definition 5 we defined the consistency model for capacity constraints as proposed in [21, 25, 30]. A main motivation behind this model is that changes in the flow should not violate the capacity of any edge, no matter what "mix" of old and new flow rules is currently in place due to asynchrony. However, the model only considers specific discrete points in time: Either a flow F_i has changed completely or it has not. Thus, the impact of latency on the different routes is neglected.

In order to avoid congestion due to latency, we suggest the following: A part of the old flow that coincides with a part of the new flow is left in the network unchanged. The remaining part B of the old flow is migrated to the remaining part B' of the new flow, but we only consider this migration to be *strongly consistent* if B' can be added to the complete old flow without violating any edge capacity constraints. This ensures that there cannot be any congestion due to latency. What is required for the migrated part B' of the new flow is that any packets in B' stay in B' until they reach their destination (otherwise, again, congestion can occur due to latency). In other words, we require B' to be a *half flow* as given by the following definition:

Definition 8. *A* half flow *is a function $H : E \to \mathbb{R}_{\geq 0}$ s.t. for all nodes $v \in V \setminus \{s\}$ it holds that $\sum_{e \in \boldsymbol{in}(v)} H(e) \leq \sum_{e \in \boldsymbol{out}(v)} H(e)$. A* multi-commodity half flow *is a tuple $\mathcal{H} = (H_1, \ldots, H_k)$ of half flows.*

By formalizing the above considerations, we obtain a stronger consistency model:

Definition 9. *Let N be a network and let $\mathcal{F} = (F_1, \ldots, F_k)$ and $\mathcal{F}' = (F_1', \ldots, F_k')$ be multi-commodity flows for the multi-commodity $\mathcal{K}$ s.t. $d_{F_i} \leq d_{F_i'}$, $1 \leq i \leq k$. A consistent migration update from $\mathcal{F}$ to $\mathcal{F}'$ is called a* strongly consistent migration update *if there exists a multi-commodity half flow $\mathcal{H} = (H_1, \ldots, H_k)$ s.t. the following three conditions hold:* 1) *$H_i(e) \leq F_i'(e)$ for all $e \in E$ and all $1 \leq i \leq k$,* 2) *$F_i'(e) - H_i'(e) \leq F_i(e)$ for all $e \in E$ and all $1 \leq i \leq k$, and* 3) *$\mathcal{F}(e) + \sum_{i=1}^{k} H_i(e) \leq c(e)$ for all $e \in E$.*

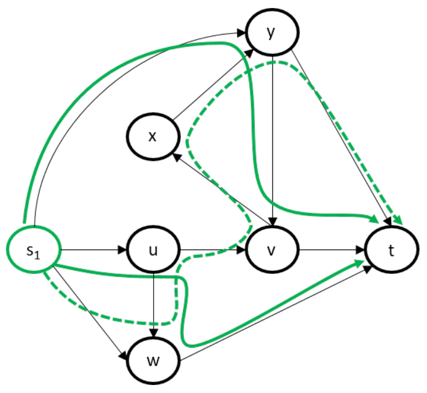
A strongly consistent migration *is a sequence of strongly consistent migration updates.*

Condition 1) ensures that each half flow is contained in the respective new flow, Condition 2) ensures that when subtracting the half flow from the respective new flow, the result is contained in the respective old flow (thus, it can remain in the network unchanged) and Condition 3) ensures that even if the old flow is still completely present in the network, the multi-commodity half flow can be added without violating any edge capacity constraints.

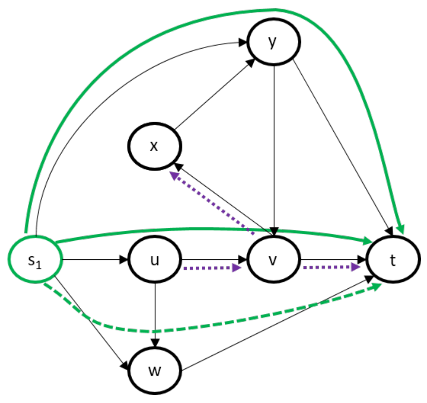
As Figure 4 shows, the migration updates performed by Algorithm 1 are not necessarily strongly consistent.

In the following, we show how to adapt Algorithm 1 in order to make the performed migration update strongly consistent. In a way, the point where the strong consistency breaks in Algorithm 1 is the replacement/removal of cycles in steps 5 and 6. More precisely, if there are no cycles in any flow in $\mathcal{F}$ after step 4, then the performed update is strongly consistent as the flows added in step 3 can be taken as the multi-commodity half flow whose existence is required. The design of Algorithm 1 asserts immediately that the flow added in step 3 indeed satisfies the half flow conditions. Note that cycles occurring in the augmenting flow F_A do not change this assessment and therefore do not have to be avoided specifically.

Our adapted algorithm proceeds as follows: Starting from the destination t, find a first back-edge along the augmenting flow. However, now, starting from this farthest back-edge, we will consider each commodity individually, and



(a) Initial network with a solid green flow F of size 2 and a dashed augmenting flow F_A for F of size 1 via s_1, w, u, v, x, y, t. All edges have a capacity of 1.



(b) Network with F' after applying a consistent update via Algorithm 1. The purple arrows depict where strong consistency cannot be maintained.

Figure 4: In this small example, Algorithm 1 updates directly from F in Subfigure 4a to F' in Subfigure 4b. While the update is consistent, it is not strongly consistent: The purple arrows in Subfigure 4b from u to v, from v to x, and from v to t depict where strong consistency cannot be maintained: Assume that the update is strongly consistent and a half flow as required exists. Due to Condition 1), the half flow on (v, x) needs to be zero. Due to Condition 2), the half flow on (u, v) needs to be positive. Lastly, due to Condition 3), the half flow on (v, t) needs to be zero as well. Hence the half flow definition is violated at node v, since the incoming half flow is positive, but all outgoing half flow is zero.

augment along the farthest cycle beyond this back-edge (seen from s), made up of the augmenting flow and the respective commodity.

If one arc of the cycle is constituted by the augmenting path and the remaining arc by the chosen commodity (i.e., the cycle consists only of two continuous segments of commodity or augmenting flow), then we can augment along a cycle analogously to augmenting w.r.t. a back-edge. As we will show, there is always a farthest cycle of this kind.

Initially starting from t, in each of these augmenting updates, we will progress closer towards the farthest back-edge, guaranteeing that the number of updates is polynomial. As such, we need a more fine-grained Algorithm 3, that handles these updates.

For ease of readability, we give a high-level description of the algorithm:

Algorithm 3.

1. *Choose a farthest back-edge* $(u, v) = e^*$.
2. *Choose a commodity* K_i.
3. *Choose a partial flow* F_a *of* F_A *starting from* u *of size* $\min(F_i(e), F_A(e^*))$ *s.t.* $F_a(e^*) = \min(F_i(e), F_A(e^*))$.

4. *Decompose F_a into $r \leq m$ augmenting unsplitted flows $F_a^1, \ldots, F_a^r$, i.e., $\sum_{j=1}^{r} F_a^j = F_a$.*

5. *Mark edges on F_a^1 successively from t towards u until a cycle composed of marked edges and F_i appears. Let $e' = (u', v')$ be the edge which was marked last.*

6. *Choose such a cycle where one continuous arc consists of marked edges and the remaining arc of commodity K_i. Augment along this cycle. Iterate until there is no cycle composed of marked edges and F_i left.*

7. *Delete cycles of F_a^1 until no more cycles of F_a^1 exist. For this, always choose cycles of the following kind: There exists a node y on the not re-routed part of F_a^1 s.t. the cycle consists of a part of F_a^1 from y to u' that was not re-routed in step 6 and a part of F_a^1 from u' to y that was re-routed. Of these cycles, choose one with the largest number of edges in the second part. From now on, consider just the edges of F_a^1 between u' and t (according to the order given by F_a^1) as marked.*

8. *Iterate steps 5 to 7, always starting from the farthest unmarked edge and proceeding towards u in step 5, until all edges in F_a^1 are marked.*

9. *Iterate steps 5 to 8 for the flows $F_a^2, \ldots, F_a^r$ successively.*

10. *Iterate steps 3 to 9 for all other commodities than K_i successively.*

11. *Iterate steps 2 to 10, always choosing a farthest back-edge, until no back-edge remains.*

12. *Iterate steps 3 to 9 for the commodity K to which the augmenting flow F_A "belongs", where we set $u := s$. However, after each execution of step 8, replace the respective augmenting flow F_a^j with a flow of commodity K.*

We will now show that Algorithm 3 performs a strongly consistent migration for a given multi-commodity flow and a corresponding augmenting flow F_A belonging to commodity K. Recall that all flows are cycle-free and observe that the design of the algorithm prevents the occurrence of cycles for any commodity. The only cycles occurring are those containing augmenting flow, which are subsequently deleted.

Further observe that in step 4, F_a can be decomposed into at most m augmenting paths by, e.g., iteratively choosing a path F_a^j in F_a from u to t s.t. there exists some edge e'' with $F_a^j(e'') = F_a(e'')$ and removing F_a^j from F_a.

After every execution of step 7 (and also during every execution of 5), the respective augmenting flow F_a^j has the following property $\mathcal{Q}$: There is a node w (after step 7 we have $w = u'$) s.t. all edges beyond w are marked and all edges before w are not marked, with the flow F_a^j constituting the same simple path between u and w as in the last iteration of steps 5 to 7. Note that even though we start with an unsplitted flow F_a^j in step 5, the augmenting flow F_a^j beyond w may be splitted due to the performed augmentations.

Before showing property $\mathcal{Q}$, we assume for now that $\mathcal{Q}$ holds, in order to show two things: First, we show that in step 6, when choosing a cycle, we can actually choose a cycle as described in step 6.

Property $\mathcal{Q}$ guarantees the existence of at least one such cycle: Just choose some arbitrary cycle and then follow the continuous part of the cycle consisting of commodity K_i from u' to the other end, given by some node x. Then there must be a path from u' to x consisting of marked edges, which together with the aforementioned path forms a cycle as described. After augmenting along this cycle, we may consider the re-routed augmenting flow as marked. Thus, property $\mathcal{Q}$ still holds and hence guarantees the existence of another cycle as described, under the condition that a cycle of marked edges and commodity K_i still exists. Note that in this context we still speak of marked edges, though technically some specific flow on the edges is marked (which is necessary because of potential cycles in the augmenting flow after augmenting along the first cycle).

Second, we show that in step 7, when choosing a cycle, we can actually choose a cycle as described in step 7. Again assuming property $\mathcal{Q}$, for the first cycle deletion we can choose a cycle as described due to the choice of u' (if a cycle exists).

According to the properties of the deleted cycle, the (deleted) flow going from y to u' must continue towards t only along edges which cannot be part of a cycle. Thus, if we ignore this deleted flow and its continuation towards t, all the (augmenting) flow re-routed in step 6 still arises from u'. Hence, after deleting the first (and any subsequent) cycle, we can again choose a cycle as described, should a cycle still exist. Note that the amount of flow in F_a^j going from u to u' is sufficient for all cycle deletions determined by the re-routed flow. Moreover, note that the considered cycles may use any edge only once, but are allowed to contain a node more than once in the corresponding cyclic order of nodes.

We now show that property $\mathcal{Q}$ holds: Observe that property $\mathcal{Q}$ holds in the beginning of each iteration of steps 5 to 8. If property $\mathcal{Q}$ holds in the beginning of step 5, then it holds during the whole step 5. Also, it holds after step 7, due to the last sentence in step 7 in conjunction with the above considerations regarding step 7. In particular, a part of the (unmarked) flow from u to u' must remain after the deletion of cycles in step 7, since all the flow in F_a^j starts along this path and some part of F_a^j must actually arrive at t.

Note that the edges marked at the end of any execution of step 7 together with the edges of commodity K_i cannot form a cycle, since these newly marked edges were used by commodity K_i before the augmentation in step 6 (while the reverse applies to the re-routed flow of commodity K_i).

All of the above applies analogously for the execution of step 12.

After these preliminaries regarding the correctness of Algorithm 3, we will now show the strong consistency. Observe that the only updates performed in the physical network occur in step 6 and step 12. We first deal with step 6:

When flow of commodity K_i is re-routed in step 6, it replaces (parts of) the augmenting flow F_a^j, yielding a half flow H_a^j. As this replaced flow did not form a cycle with itself or together with flow of the commodity K_i, adding flow F_i along these edges will not form a cycle with F_a^j or the commodity K_i either.

It remains to show for step 6 that H_a^j satisfies the strong consistency conditions given in Definition 9. As H_a^j is part of the new flow, it satisfies Condition 1). Furthermore, the new flow was just increased by H_a^j, thus guaranteeing Condition 2). Lastly, as the augmenting flow F_a^j together with the old flow did not violate any edge capacity constraints (and the augmenting flow is not actually inserted in the physical network), Condition 3) follows.

Again, analogous arguments apply for step 12, as the newly added flow just replaces an augmenting flow. Hence, the migration performed by Algorithm 3 is a strongly consistent migration.

Furthermore, observe that the only (physical) network updates performed occur in steps 6 and 12. Careful analysis shows that step 6 will be executed at most km^2n times, where each execution requires at most n updates. As step 12 invokes iterations of steps 3 to 9, and performs a strongly consistent migration update after each invocation of step 8, the asymptotic number of updates does not change. Hence, we can bound the total number of updates by $O(km^2n^2)$.

However, in order to strongly consistently migrate to new desired demands, we still need a framework akin to Algorithm 2, which we will now provide:

Algorithm 4. *Let N be a network and let $\mathcal{F}$ be a multi-commodity flow for the multi-commodity $\mathcal{K}$. Let $(d_1, \ldots, d_k)$ be a vector of demands s.t. i) there exists a multi-commodity flow for $\mathcal{K}$ satisfying these demands, and ii) $d_1 \geq d_{F_1}, \ldots, d_k \geq d_{F_k}$.*

1. *Compute a multi-commodity flow $\mathcal{F}_1'$ with a demand vector of $(d_1, d_{F_2}, \ldots, d_{F_k})$ using an LP.*
2. *Compute the difference flow $Z^{\mathcal{F},\mathcal{F}_1'}$.*
3. *Augment $\mathcal{F}$ with $Z^{\mathcal{F},\mathcal{F}_1'}$ by using Algorithm 3.*
4. *Iterate the above three steps for the remaining commodities $K_2, \ldots, K_k$, thereby obtaining the flows $\mathcal{F}_2, \ldots, \mathcal{F}_k$ with the demand vectors of $(d_1, d_2, d_{F_3}, \ldots, d_{F_k})$, $\ldots$, $(d_1, d_2, d_3, \ldots, d_{k-1}, d_{F_k})$, $(d_1, \ldots, d_k)$.*

Note that all calculations performed by Algorithm 4 (and by its invoked Algorithm 3) can be performed in polynomial time.

Corollary 4. *Algorithm 4 performs a strongly consistent migration from $\mathcal{F}$ to some multi-commodity flow with a demand vector of $(d_1, \ldots, d_k)$, using only $O(k^2m^2n^2)$ strongly consistent migration updates.*

7. NP-Hardness of Consistently Migrating Unsplittable Flows

So far we considered splittable multi-commodity flows as defined in Section 3, however this is just one side of the coin. Not only recently, "*Motivated by routing problems arising in real-life applications* [...] *unsplittable flows have moved into the focus of research.*" [3] An unsplittable flow is defined as a flow only taking one simple path from its source to its destination:

Definition 10. *A single-commodity flow F is called an* unsplittable single-commodity flow *if the set of edges $e \in E : F(e) > 0$ forms a simple (i.e., cycle-free) path from s to t. A multi-commodity flow $\mathcal{F}$ is called an* unsplittable multi-commodity flow *if all of its single-commodity flows are unsplittable single-commodity flows.*

Similarly, we define an unsplittable consistent migration to be a consistent migration using only unsplittable flows:

Definition 11. *A consistent migration update $(N, \mathcal{F}, \mathcal{F}')$ is called an* unsplittable consistent migration update *if both $\mathcal{F}, \mathcal{F}'$ are unsplittable multi-commodity flows. A consistent migration is called an* unsplittable consistent migration *if it consists of unsplittable consistent migration updates.*

7.1. Hardness of Deciding if Demands can be Satisfied

Even et al. [12] showed for general[4] multi-commodity flow problems that it is NP-hard to decide if even the demand of two commodities can be met by integral flows in graphs with unit capacities; this case is equivalent to unsplittable multi-commodity flows. Kleinberg showed the NP-hardness also for the single-source unsplittable multi-commodity flow case [28], which is equivalent to the model used in this article (by reversing all edge directions). From the results of Baier et al. [3], it can be inferred that the NP-hardness also holds for just two unsplittable flows with common source s and destination t. The case of just one unsplittable single-commodity flow can be solved in polynomial time by, e.g., finding the path with the highest capacity from s to t.

7.2. Consistently Migrating Unsplittable Flows

As it is already NP-hard to decide if the demands of some commodities can be met (see the above Subsection 7.1), it is also an NP-hard problem to consistently migrate to a multi-commodity flow meeting these demands.

However, what happens if we know that the desired demands of the commodities can be met? I.e., if we are given the current multi-commodity flow and a multi-commodity flow meeting the desired demands (both unsplittable), is unsplittably consistently migrating NP-hard as well? As it turns out, the answer is yes:

Theorem 5. *Let N be a network and let $\mathcal{F} = (F_1, \dots, F_k)$, $\mathcal{F}' = (F'_1, \dots, F'_k)$ be unsplittable multi-commodity flows for the multi-commodity $\mathcal{K}$ s.t. $d_{F_i} \le d_{F'_i}$, $1 \le i \le k$. It is NP-hard to decide if there is an unsplittable consistent migration from $\mathcal{F}$ to some unsplittable multi-commodity flow satisfying the demands of $\mathcal{F}'$.*

Note that for multi-commodity flows $\mathcal{F}, \mathcal{F}'$ with identical demands, the problem is trivial as zero updates are required to meet the demands of $\mathcal{F}'$.

We will prove Theorem 5 by reduction from the classic NP-hard problem *PARTITION* (also known as number partitioning):

[4]I.e., with each flow having a possibly distinct source and destination respectively.

Definition 12 (*PARTITION* [17]). *Let $\mathcal{A}$ be a finite set of k positive real-valued elements $a_1, \dots, a_k$, and set $A := \sum_{i=1}^{k} a_i$. Is it possible to partition $\mathcal{A}$ into two sets $\mathcal{A}_1, \mathcal{A}_2$ s.t. the sums $A_1 := \sum_{a_i \in \mathcal{A}_1} a_i$, $A_2 := \sum_{a_i \in \mathcal{A}_2} a_i$ of their respective elements are identical, i.e., $A_1 = A_2 = \frac{A}{2}$?*

Theorem 6 ([17]). *The* PARTITION *problem from Definition 12 is NP-hard.*

We can now prove Theorem 5:

Proof of Theorem 5. For every instance I of the *PARTITION* problem we will construct in polynomial time an instance I' of the problem described in Theorem 5 s.t. I is a yes-instance if and only if I' is a yes-instance.

Given an instance I of the *PARTITION* problem, we create a network $N = (G = (V, E), c)$ as follows: V consists of k sources $s_1, \dots, s_k$, two sources s_a, s_b, two nodes v_a, v_b, and a destination t, i.e., $k + 5$ nodes in total. E is composed of an edge from each $s_1, \dots, s_k$ to both v_a and v_b with capacity of a_i, $1 \leq i \leq k$ respectively. Furthermore, there are edges from v_a, v_b to t with a capacity of A each, and edges from s_a to v_a and s_b to v_a, v_b with capacities of $A/2$. In total, there are $2k + 2 + 1 + 2 = 2k + 5$ edges.

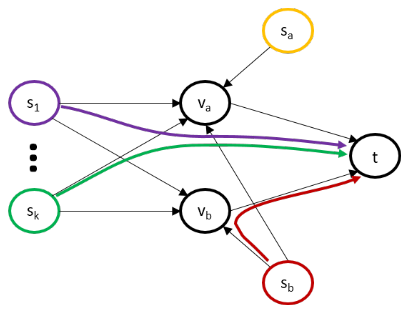
The unsplittable multi-commodity flow $\mathcal{F}$ composed of the unsplittable flows $F_1, \dots, F_k, F_b, F_a$ is defined as follows: $d_{F_1} = a_1, \dots, d_{F_k} = a_k$, with each of these k flows F_i being routed from its source s_i via v_a to t, $1 \leq i \leq k$. Furthermore, $d_{F_b} = A/2$, with it being routed from its source s_b via v_b to t. Lastly, $d_{F_a} = 0$.

The unsplittable multi-commodity flow $\mathcal{F}'$, composed of the unsplittable flows $F'_1, \dots, F'_k, F'_b, F'_a$, is defined as follows: $d_{F'_1} = a_1, \dots, d_{F'_k} = a_k$, but with each of these k flows F_i being routed from their source s_i via v_b to t, $1 \leq i \leq k$. Lastly, $d_{F_b} = d_{F_a} = A/2$, with both being routed via v_a.

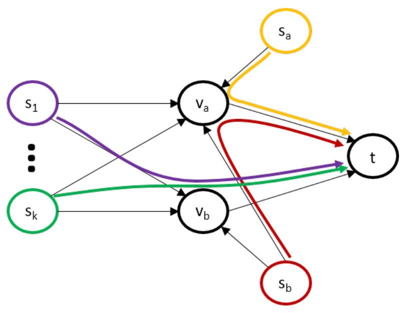
The construction can be performed in polynomial time and is depicted in Figure 5.

Let us start by assuming that the *PARTITION* instance I is solvable, i.e., it is a yes-instance. Then, we can select the flows corresponding to $\mathcal{A}_1$ and unsplittably consistently migrate them to the path via v_b, as their combined size is exactly $A/2$. In the next step, we can add an unsplittable flow F'_a of size $A/2$ from s_a via v_a to t, showing that I' is a yes-instance as well.

To conclude the proof, let us assume that the *PARTITION* instance I is not solvable, i.e., it is a no-instance. Observe that currently, the edge from v_a to t has no free capacity, and the edge from v_b to t has a free capacity of exactly $A/2$. Unless we can move a subset of the set of flows $F_1, \dots, F_k$ of combined size exactly $A/2$ to the path via v_b, neither the edge from v_a to t nor the edge from v_b to t will have a free capacity of at least $A/2$. As the *PARTITION* instance is not solvable, this is not possible, meaning neither F_b can be moved consistently nor F_a can be added to the network, as both their sizes are $A/2$. Hence, I' is a no-instance as well. □



(a) Unsplittable multi-commodity flow $\mathcal{F}$



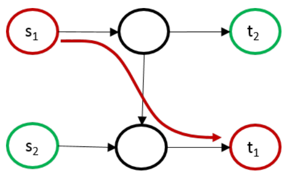
(b) Unsplittable multi-commodity flow $\mathcal{F}'$

Figure 5: The old unsplittable multi-commodity flow $\mathcal{F}$ is depicted in Subfigure 5a on the left, while the new unsplittable multi-commodity flow $\mathcal{F}'$ is depicted in Subfigure 5b on the right. In Subfigure 5a, the edge from v_a to t is used at full capacity. Similarly, in Subfigure 5b both edges from v_a, v_b to t are used at full capacity. In order to consistently migrate to a multi-commodity flow with the same demands as $\mathcal{F}'$, flows from $F_1, \ldots, F_k$ of combined size exactly $A/2$ need to migrate to the edge from v_b to t. However, this is equivalent to solving the corresponding *PARTITION* instance.

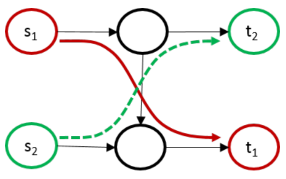
8. Augmenting Flows beyond a Single Destination

Besides the case of unsplittable flows as covered in Section 7, there is another natural extension of our model: Namely, the case of multi-commodity flows with multiple sources and multiple destinations.

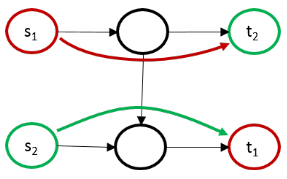
As a simple example shows (cf. Figure 6), applying an augmenting path in a straightforward way to a network with multiple sources and destinations will not even necessarily result in a correct multi-commodity flow. The outgoing flow can end up being re-routed to a wrong destination.



(a) Initial network with just one flow from s_1 to t_1.



(b) An augmenting flow is found from s_2 to t_2.



(c) The resulting flows are not feasible in the network!

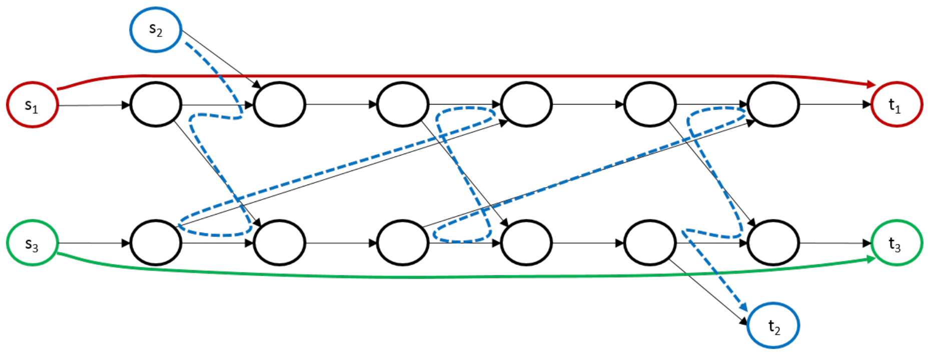
Figure 6: The existence of an augmenting flow does not guarantee feasible flows for multiple sources and destinations. E.g., the flow from s_1 might end up in t_2.

A logical consequence is to admit only augmenting flows which re-route correctly, i.e., each outgoing flow of a source is still routed to its assigned destination. However, as Hu noted [22], it is unlikely that the technique of augmenting paths can be extended to a general multi-commodity setting (cf. Section 1).

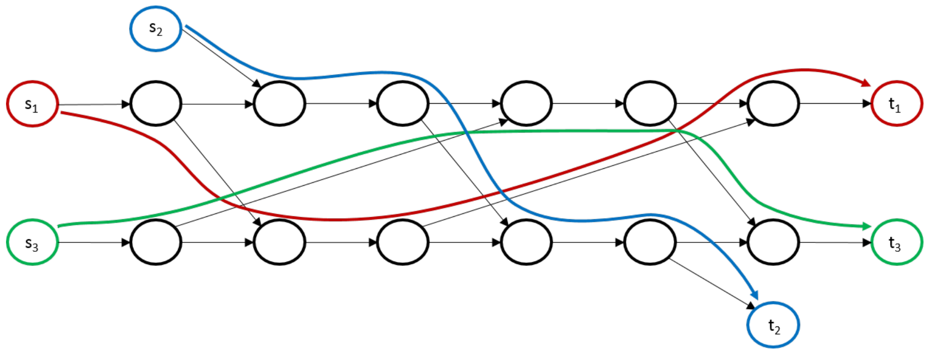
Nonetheless, what would happen if we could develop an augmenting path approach that results in correct multi-commodity flows?

Sacrificing polynomial runtime, one could check all possible flows between source-destination pairs in the residual networks to see if there is an augmenting flow that respects each flow ending at its correct destination.

However, a more intricate example (cf. Figure 7) shows that, even in this case, it is not always possible to migrate consistently from the initial flow to the augmented flow.



(a) There is an augmenting flow from s_2 to t_2 that results in a proper multi-commodity flow.



(b) The resulting new flow after the augmenting flow from above is applied to the network.

Figure 7: Neither (part of) the red nor the green flow can consistently migrate to any imaginable flow in the network: Moving any part of the red flow to the bottom path (or any part of the green flow to the top path) in Subfigure 7a will violate the consistency condition. Still, there is an augmenting flow moving both flows to other edges – which also respects the assignment of the sink-destination pairs, see Subfigure 7b for the resulting network. Hence, even an augmenting flow resulting in a proper multi-commodity flow does not guarantee a consistent migration for multiple sources and destinations.

9. Concluding Remarks

In this work, we extended the notion of augmenting flows to the setting of multi-commodity flows for a single logical destination, providing algorithms to efficiently tackle the problem of consistent migration in Software Defined Networks. We also showed that augmenting flows can guarantee stronger consistency properties in this setting, and that consistent migration is NP-hard for unsplittable flows, even if both initial and desired demands are satisfiable. A natural question arises: Can we generalize the concept of an augmenting path to the general multi-commodity setting? As it turns out, even if we could efficiently find augmenting paths respecting the source-destination pairs, they will break consistency during migration. We thus believe that fundamentally different techniques are required to apply the method of augmenting flows for consistent migration updates beyond the anycast setting.

10. Acknowledgements

We would like to thank the anonymous reviewers of the 17th International Conference on Distributed Computing and Networking (ICDCN '16) for their helpful comments on our preliminary extended abstract [4].

Furthermore, we would like to thank the anonymous reviewers of this journal article for their helpful comments as well.

Klaus-Tycho Foerster was supported in part by Microsoft Research.

References

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

[2] Z. Al-Qudah, S. Lee, M. Rabinovich, O. Spatscheck, and J. E. van der Merwe. Anycast-aware transport for content delivery networks. In J. Quemada, G. León, Y. S. Maarek, and W. Nejdl, editors, *Proceedings of the 18th International Conference on World Wide Web, WWW 2009, Madrid, Spain, April 20-24, 2009*, pages 301–310. ACM, 2009.

[3] G. Baier, E. Köhler, and M. Skutella. The k-splittable flow problem. *Algorithmica*, 42(3-4):231–248, 2005.

[4] S. Brandt, K.-T. Foerster, and R. Wattenhofer. Augmenting anycast network flows. In *Proceedings of the 17th International Conference on Distributed Computing and Networking*, ICDCN '16, pages 24:1–24:10, New York, NY, USA, 2016. ACM.

[5] S. Brandt, K.-T. Foerster, and R. Wattenhofer. On Consistent Migration of Flows in SDNs. In *2016 IEEE Conference on Computer Communications, INFOCOM 2016, San Francisco, California, USA, April 2016.* IEEE, 2016.

[6] M. Calder, A. Flavel, E. Katz-Bassett, R. Mahajan, and J. Padhye. Analyzing the performance of an anycast CDN. In K. Cho, K. Fukuda, V. S. Pai, and N. Spring, editors, *Proceedings of the 2015 ACM Internet Measurement Conference, IMC 2015, Tokyo, Japan, October 28-30, 2015*, pages 531–537. ACM, 2015.

[7] M. Canini, P. Kuznetsov, D. Levin, and S. Schmid. Software transactional networking: concurrent and consistent policy composition. In N. Foster and R. Sherwood, editors, *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, HotSDN 2013, The Chinese University of Hong Kong, Hong Kong, China, August 16, 2013*, pages 1–6. ACM, 2013.

[8] M. Casado, N. Foster, and A. Guha. Abstractions for software-defined networks. *Commun. ACM*, 57(10):86–95, 2014.

[9] W. Cerroni, F. Callegati, B. Martini, and P. Castoldi. Analytical model for anycast service provisioning in data center interconnections. In D. Simeonidou, editor, *16th International Conference on Optical Network Design and Modelling, ONDM 2012, Colchester, United Kingdom, April 17-20, 2012*, pages 1–6. IEEE, 2012.

[10] D. Cicalese, D. Giordano, A. Finamore, M. Mellia, M. M. Munafò, D. Rossi, and D. Joumblatt. A first look at anycast CDN traffic. *CoRR*, abs/1505.00946, 2015.

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

[12] S. Even, A. Itai, and A. Shamir. On the complexity of timetable and multicommodity flow problems. *SIAM J. Comput.*, 5(4):691–703, 1976.

[13] K.-T. Foerster, R. Mahajan, and R. Wattenhofer. Consistent updates in software defined networks: On dependencies, loop freedom, and blackholes. In *2016 IFIP Networking Conference, Networking 2016 and Workshops, Vienna, Austria, May 17-19, 2016*, pages 1–9. IEEE, 2016.

[14] K.-T. Foerster, S. Schmid, and S. Vissicchio. Survey of consistent network updates. *CoRR*, arXiv:1609.02305 [cs.NI], 2016.

[15] L. R. Ford and D. R. Fulkerson. Maximal flow through a network. *Canad. J. Math*, 8:399–404, 1956.

[16] L. R. Ford and D. R. Fulkerson. *Flows in Networks.* Princeton University Press, Princeton, NJ, USA, 1962.

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

[18] M. Gharbaoui, B. Martini, and P. Castoldi. Anycast-based optimizations for inter-data-center interconnections. *J. Opt. Commun. Netw.*, 4(11):B168–B178, Nov 2012.

[19] 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 R. E. Bixby, E. A. Boyd, and R. Z. Ríos-Mercado, editors, *Integer Programming and Combinatorial Optimization, 6th International IPCO Conference, Houston, Texas, USA, June 22-24, 1998, Proceedings*, volume 1412 of *Lecture Notes in Computer Science*, pages 338–352. Springer, 1998.

[20] 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 J. Rexford and A. Vahdat, editors, *Proceedings of the 1st ACM Symposium on SDN Research, SOSR '15, Santa Clara, California, USA, June 17-18, 2015*, pages 25:1–25:6. ACM, 2015.

[21] C. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer. Achieving high utilization with software-driven WAN. In D. M. Chiu, J. Wang, P. Barford, and S. Seshan, editors, *ACM SIGCOMM 2013 Conference, SIGCOMM'13, Hong Kong, China, August 12-16, 2013*, pages 15–26. ACM, 2013.

[22] T. C. Hu. Multi-commodity network flows. *Operations Research*, 11(3):344–360, 1963.

[23] A. Itai. Two-commodity flow. *J. ACM*, 25(4):596–611, 1978.

[24] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, and A. Vahdat. B4: experience with a globally-deployed software defined wan. In D. M. Chiu, J. Wang, P. Barford, and S. Seshan, editors, *ACM SIGCOMM 2013 Conference, SIGCOMM'13, Hong Kong, China, August 12-16, 2013*, pages 3–14. ACM, 2013.

[25] X. Jin, H. H. Liu, R. Gandhi, S. Kandula, R. Mahajan, M. Zhang, J. Rexford, and R. Wattenhofer. Dionysus: Dynamic scheduling of network updates. In F. E. Bustamante, Y. C. Hu, A. Krishnamurthy, and S. Ratnasamy, editors, *ACM SIGCOMM 2014 Conference, SIGCOMM'14, Chicago, USA, August 17-22, 2014*, pages 539–550. ACM, 2014.

[26] L. G. Khachian. A polynomial algorithm in linear programming. *Dokl. Akad. Nauk SSSR*, 244:1093–1096, 1979. English translation in Soviet Math. Dokl. 20, 191-194, 1979.

[27] H. Kim, J. Reich, A. Gupta, M. Shahbaz, N. Feamster, and R. J. Clark. Kinetic: Verifiable dynamic network control. In *12th USENIX Symposium on Networked Systems Design and Implementation, NSDI 15, Oakland, CA, USA, May 4-6, 2015*, pages 59–72. USENIX Association, 2015.

[28] J. M. Kleinberg. Single-source unsplittable flow. In *37th Annual Symposium on Foundations of Computer Science, FOCS '96, Burlington, Vermont, USA, 14-16 October, 1996*, pages 68–77. IEEE Computer Society, 1996.

[29] M. Kuzniar, P. Perešíni, and D. Kostic. What you need to know about SDN flow tables. In J. Mirkovic and Y. Liu, editors, *Passive and Active Measurement - 16th International Conference, PAM 2015, New York, NY, USA, March 19-20, 2015, Proceedings*, volume 8995 of *Lecture Notes in Computer Science*, pages 347–359. Springer, 2015.

[30] H. H. Liu, X. Wu, M. Zhang, L. Yuan, R. Wattenhofer, and D. A. Maltz. zUpdate: updating data center networks with zero loss. In D. M. Chiu, J. Wang, P. Barford, and S. Seshan, editors, *ACM SIGCOMM 2013 Conference, SIGCOMM'13, Hong Kong, China, August 12-16, 2013*, pages 411–422. ACM, 2013.

[31] J. McClurg, H. Hojjat, P. Cerný, and N. Foster. Efficient synthesis of network updates. In D. Grove and S. Blackburn, editors, *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation, Portland, OR, USA, June 15-17, 2015*, pages 196–207. ACM, 2015.

[32] T. Mizrahi and Y. Moses. Time-based updates in software defined networks. In N. Foster and R. Sherwood, editors, *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, HotSDN 2013, The Chinese University of Hong Kong, Hong Kong, China, August 16, 2013*, pages 163–164. ACM, 2013.

[33] T. Mizrahi and Y. Moses. On the necessity of time-based updates in SDN. In R. Sherwood, editor, *Open Networking Summit 2014, ONS 2014, Santa Clara, CA, USA, March 2-4, 2014*. USENIX, 2014.

[34] T. Mizrahi, O. Rottenstreich, and Y. Moses. Timeflip: Scheduling network updates with timestamp-based TCAM ranges. In *2015 IEEE Conference on Computer Communications, INFOCOM 2015, Kowloon, Hong Kong, April 26 - May 1, 2015*, pages 2551–2559. IEEE, 2015.

[35] A. Noyes, T. Warszawski, P. Cerný, and N. Foster. Toward synthesis of network updates. In B. Finkbeiner and A. Solar-Lezama, editors, *Proceedings Second Workshop on Synthesis, SYNT 2013, Saint Petersburg, Russia, July 13th and July 14th, 2013.*, volume 142 of *EPTCS*, pages 8–23, 2014.

[36] C. Prakash, J. Lee, Y. Turner, J. Kang, A. Akella, S. Banerjee, C. Clark, Y. Ma, P. Sharma, and Y. Zhang. PGA: using graphs to express and automatically reconcile network policies. *Computer Communication Review*, 45(5):29–42, 2015.

[37] M. Prince. A brief primer on anycast. https://blog.cloudflare.com/a-brief-anycast-primer/, October 2011.

[38] M. Prince. Load balancing without load balancers. https://blog.cloudflare.com/cloudflares-architecture-eliminating-single-p/, March 2013.

[39] M. Reitblatt, N. Foster, J. Rexford, C. Schlesinger, and D. Walker. Abstractions for network update. In L. Eggert, J. Ott, V. N. Padmanabhan, and G. Varghese, editors, *ACM SIGCOMM 2012 Conference, SIGCOMM '12, Helsinki, Finland - August 13 - 17, 2012*, pages 323–334. ACM, 2012.

[40] M. Reitblatt, N. Foster, J. Rexford, and D. Walker. Consistent updates for software-defined networks: change you can believe in! In H. Balakrishnan, D. Katabi, A. Akella, and I. Stoica, editors, *Tenth ACM Workshop on Hot Topics in Networks (HotNets-X), HOTNETS '11, Cambridge, MA, USA - November 14 - 15, 2011*, page 7. ACM, 2011.

[41] W. Rothfarb, N. P. Shein, and I. T. Frisch. Common terminal multicommodity flow. *Operations Research*, 16(1):202–205, 1968.

[42] A. Tanenbaum and D. Wetherall. *Computer Networks (5th Edition)*. Pearson Prentice Hall, 2010.

[43] L. Zhang, C. Wu, Z. Li, C. Guo, M. Chen, and F. C. M. Lau. Moving big data to the cloud: An online cost-minimizing approach. *IEEE Journal on Selected Areas in Communications*, 31(12):2710–2721, 2013.