Augmenting Anycast Network Flows

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

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 anycast applications.

CCS Concepts

•Networks → Network management; •Theory of computation → Network flows;

Keywords

Software Defined Networks, Congestion, Anycast, Flow Augmentation, Multi-Commodity Flow

1. INTRODUCTION

The rise of *Software Defined Networks (SDNs)* has sparked an increasing interest in applying network flow algorithms. In contrast to networks that use standardized distributed protocols, SDNs allow for utilizing the available bandwidth almost completely. 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 [3, 8, 11, 12, 14] 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.

In this paper we propose to abandon LP-based solutions in favor of path augmentation, a technique developed and studied primarily in the era of The Beatles. Even though path augmentation is still taught in introductory lectures, little research has been devoted to it since, probably because more versatile (and also polynomial-time) LP-based techniques were introduced.

We believe that SDNs should be managed in an incremental way. If a commodity (a source-destination node pair) wants to reduce its bandwidth, we can simply do that without harm. 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. Maybe we are lucky, and the increased demand fits without changing any of the other flows. Maybe we are less lucky, and re-routing (also called migration) of some other flows is necessary, conceivably even recursively.

Understanding flow migration is still in its infancy: It is not clear in general (1) 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 paper [9], "*it is unlikely that similar techniques can be developed for constructing multicommodity flows*".

This is why this paper focuses on an important special case of multi-commodity flow, the so-called *anycast* problem, cf. [24]. 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. 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 [25]) data, as long as "the data gets to the cloud".

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.

ICDCN '16, January 04-07, 2016, Singapore, Singapore

DOI: http://dx.doi.org/10.1145/2833312.2833450

1.1 Structure of our Paper

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 concluding in Section 6, we show in Section 5 how to implement our method efficiently in practice.

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.

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 [5, 6], 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. [4] and Ahuja et al. [1] for an in-depth overview.

Hu [9] 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.

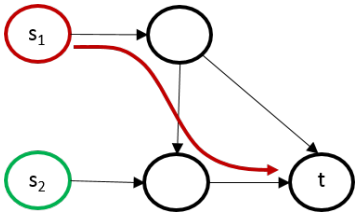
Furthermore, in 1978, shortly before the celebrated publication of the Ellipsoid method [13], Itai [10] published an improved version of Hu's two-commodity flow algorithm and showed that maximizing a two-commodity flow is as difficult as linear programming in the sense that they are polynomially equivalent.

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.

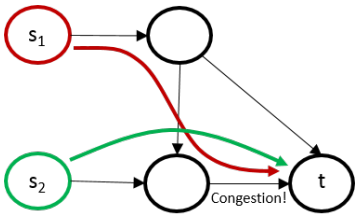
Rothfarb et al. applied augmenting paths in the following way [23]: To maximize multi-commodity flows with just one destination t, they added a logical super-source s, considered all commodities as the same commodity with new source s, and then solved the obtained single-commodity flow problem using 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 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

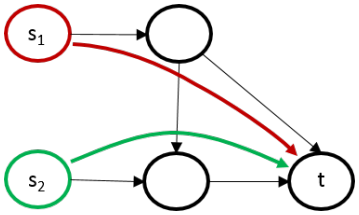
Unlike multi-commodity flow problems regarding demand satisfaction, the study of migration of flows is still in an emergent state. Perhaps it was not before the rise of Software Defined Networks (SDNs) that moving flows onto other paths became a relevant topic in practice. In SDNs, a central controller can change the behaviour of the switches in the network, allowing for, e.g., arbitrary flow allocations among



(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.

the links. There is a great amount of interesting checkable properties that the controller might want to verify (for example by model checking, cf. [16, 20]), such as waypointing via firewalls, loop freedom, or network connectivity, cf. [3] for an overview of the current state of the art.

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 [18], 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 indus-

try switches might straggle [11] – taking up to 100x longer than average to implement updates [12].

Hong et al. [8] 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 $1, 2, 4, 8, \ldots$ update steps, possibly temporarily rerouting flows to arbitrary edges in the network. Nonetheless, they cannot decide if a consistent migration is possible or not. A similar approach is also used in a datacenter setting by Liu et al. [14].

The work of Jin et al. [12] 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. [11]: 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 proposed by Mizrahi, Rottenstreich, and Moses [17, 18, 19]: 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. [21, 22] 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.

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 $\texttt{out}(v)$ and the set of incoming edges by $\texttt{in}(v)$. 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.*

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

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

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

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 $\sum_{e \in \texttt{in}(s)} F(e) = 0 = \sum_{e \in \texttt{out}(t)} F(e)$ 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:*

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

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

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 [8, 12, 14], 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 [2, 15]. 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 commodity $K \in \mathcal{K}$ 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.

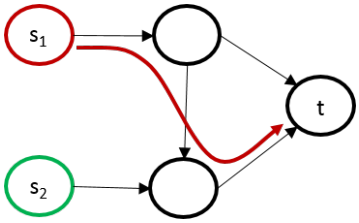
4. AUGMENTING FLOWS FOR MULTIPLE COMMODITIES

In the case of one source and one destination, it is well-known [6] 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.

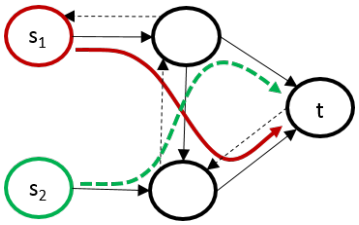
In the following Definition 6, we define an augmenting flow for the case of a multi-commodity flow where we have multiple sources (but only one destination). The augmenting flow may use edges from the residual network, which is created by adding a back-edge in the reverse direction for every edge with 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.

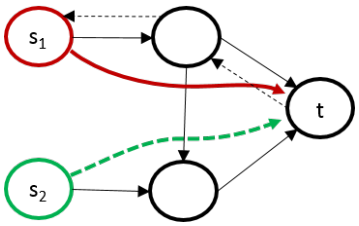
DEFINITION 6. *Let N be a network and let $\mathcal{F}$ be a multi-commodity flow for the multi-commodity $\mathcal{K}$. We denote by $\overline{G}$ the graph obtained from G by adding an edge $e^* = (v, u)$ to G for any edge $e = (u, v) \in E$. Let E^* be the set of all newly added edges. If an edge $e^* \in E^*$ starts and ends in the same vertices as some edge in E, we still consider them as distinct edges. Set $\overline{N} := (\overline{G}, \overline{c})$ 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 $E^*_{F_A} \neq \emptyset$.*



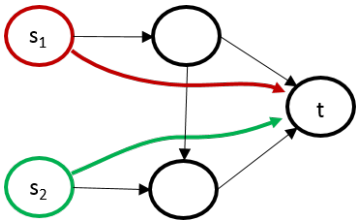
(a) Initial network with just one flow from s_1 to t. Currently, there is no space for a flow from s_2 to t, the red flow needs to be moved first.



(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 paper, such an augmenting flow always "belongs" to a specific commodity K contained in the respective multi-commodity.

We develop a technique in Algorithm 1 to transform the given multi-commodity flow step by step into a multi-commodity 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 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 $\mathcal{F} = (F_1, \ldots, F_k)$ 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 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 $F_{k_1}, ..., F_{k_q} \in \mathcal{F}, k_1 < \cdots < k_q$ 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 $\mathcal{F}' = (F'_1, \ldots, F'_k)$ 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 $F_a^{(1)}, \ldots, F_a^{(r)}$ of sizes $F_{k_1}(e_0), \ldots, F_{k_{r-1}}(e_0), U$ 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.)*

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 $e \in E$.*

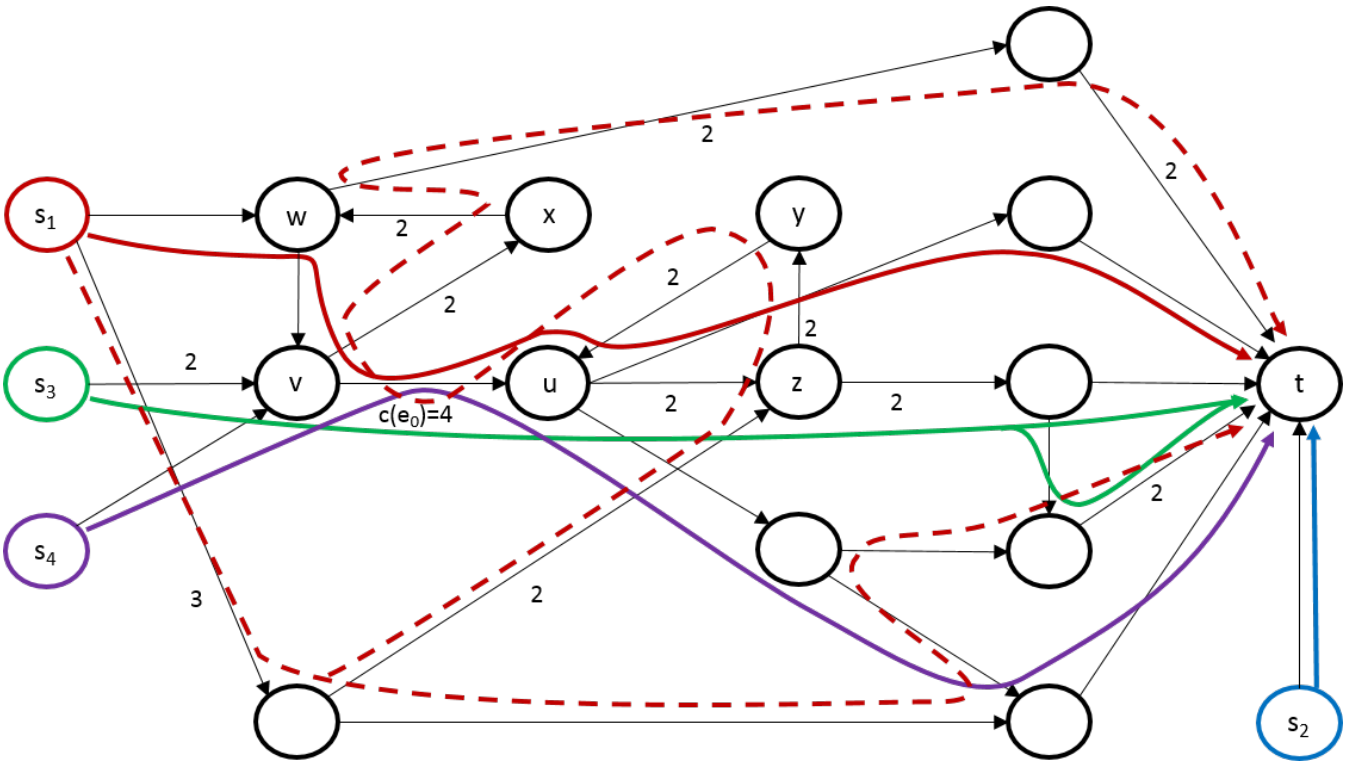
PROOF. Since F_A is an augmenting flow w.r.t. $\mathcal{F}$, it holds that $F_A(e) + \sum_{F \in \mathcal{F}} F(e) \leq c(e)$ for all $e \in E$. Thus, after step 1 we have $F'_A(e) + \sum_{F' \in \mathcal{F}'} F'(e) \leq c(e)$ for all $e \in E$. In step 2, $F'_A(e) + \sum_{F' \in \mathcal{F}'} F'(e)$ 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 $e \in E$.*

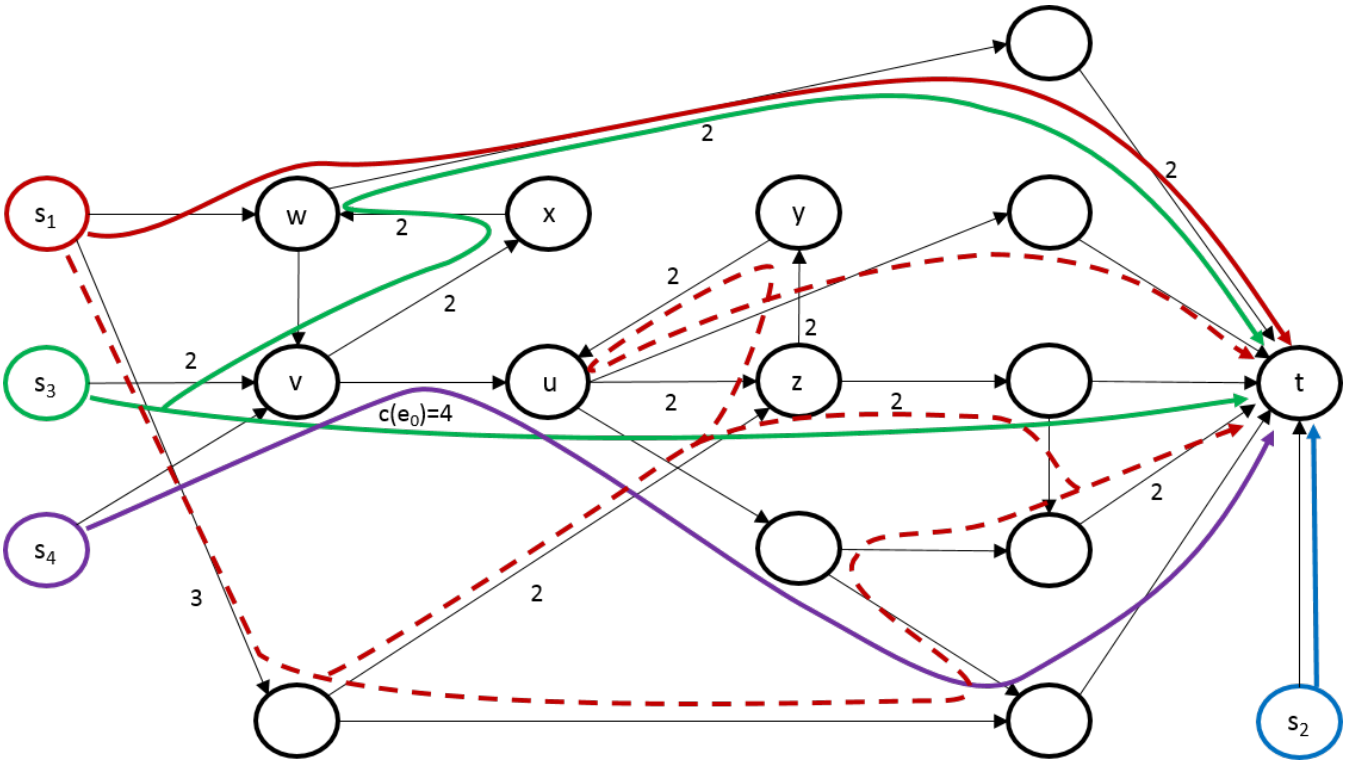
PROOF. Since F_A is an augmenting flow w.r.t. $\mathcal{F}$, it holds that $F'_A(e^*) \leq \sum_{F' \in \mathcal{F}'} F'(e)$ for all $e \in E$ after step 1. In step 2, both $\sum_{F' \in \mathcal{F}'} F'(e_0)$ and $F'_A(e^*_0)$ are diminished by $F'_A(e^*_0)$, while nothing changes for the edges $e \neq e_0$. 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 $\sum_{F' \in \mathcal{F}'} F'(e)$. □

[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.

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



(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.

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.

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

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

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,

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

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, $d_{F^h_y} = d_{F_y}$ 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 iterate this process until the desired demand is reached, if possible. However, this

[3]We note that other mechanisms such as, e.g., *SWAN* [8] or *zUpdate* [14], do not give *any* bound on the number of updates needed for consistent migration.

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 $\mathcal{K} = (K_1, \ldots, K_k)$ be a multi-commodity, and fix some $i \in \mathbb{N}$ with $1 \leq i \leq k$. Let $\mathcal{F} = (F_1, \ldots, F_k), \mathcal{F}' = (F'_1, \ldots, F'_k)$ 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 $Z^{\mathcal{F},\mathcal{F}'_1}$ using Algorithm 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, thereby obtaining flows $\mathcal{F}_2, \ldots, \mathcal{F}_k$ with 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][4]. If the computation time is an issue as well, one can also resort to approximation algorithms with a better runtime [7].

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. CONCLUDING REMARKS

In this work, we extended the notion of augmenting paths to the anycast setting, providing algorithms to efficiently tackle the problem of consistent migration in Software Defined Networks. A natural question arises: Can we generalize the concept of an augmenting path to the general multi-commodity setting?

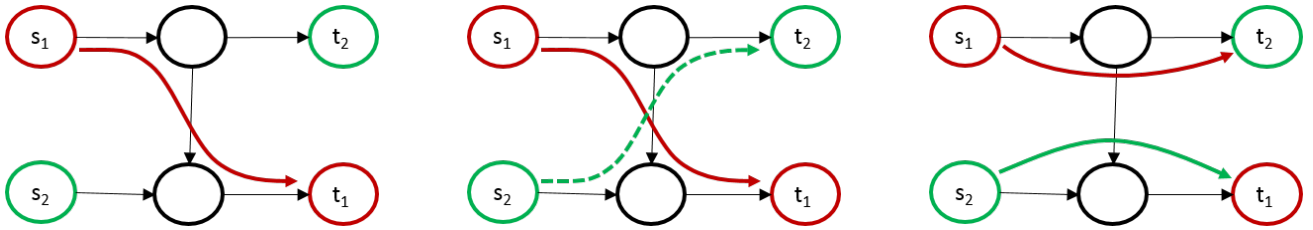
As a simple example shows (cf. Figure 4), applying an augmenting path in a straightforward way to a network with multiple sources and destinations will not even necessarily result in a correct multi-commodity flow. The outgoing flow can end up being re-routed to a wrong destination.

A 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 [9], 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? A more intricate example (cf. Figure 5) shows that, even in this case, it is not always possible to migrate consistently from the initial flow to the augmented flow.

We thus believe that fundamentally different techniques are required to apply the method of augmenting flows for consistent migration updates beyond the anycast setting.

7. ACKNOWLEDGEMENTS

We would like to thank the anonymous reviewers for their helpful comments. Klaus-Tycho Foerster was supported in part by Microsoft Research.

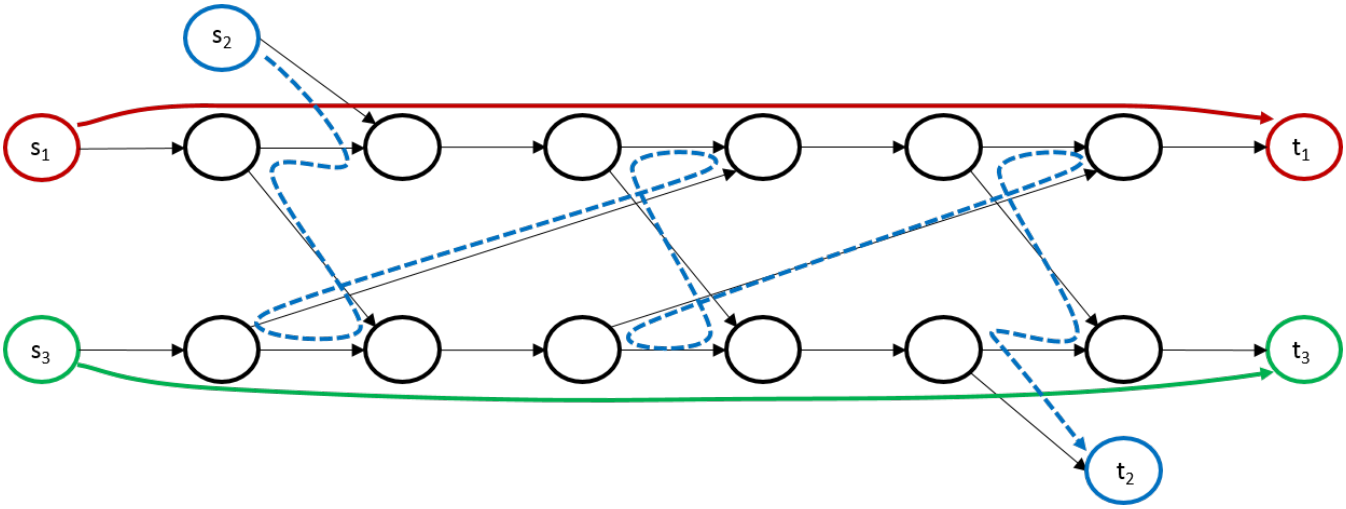


(a) Initial network with just one flow from s_1 to t_1.

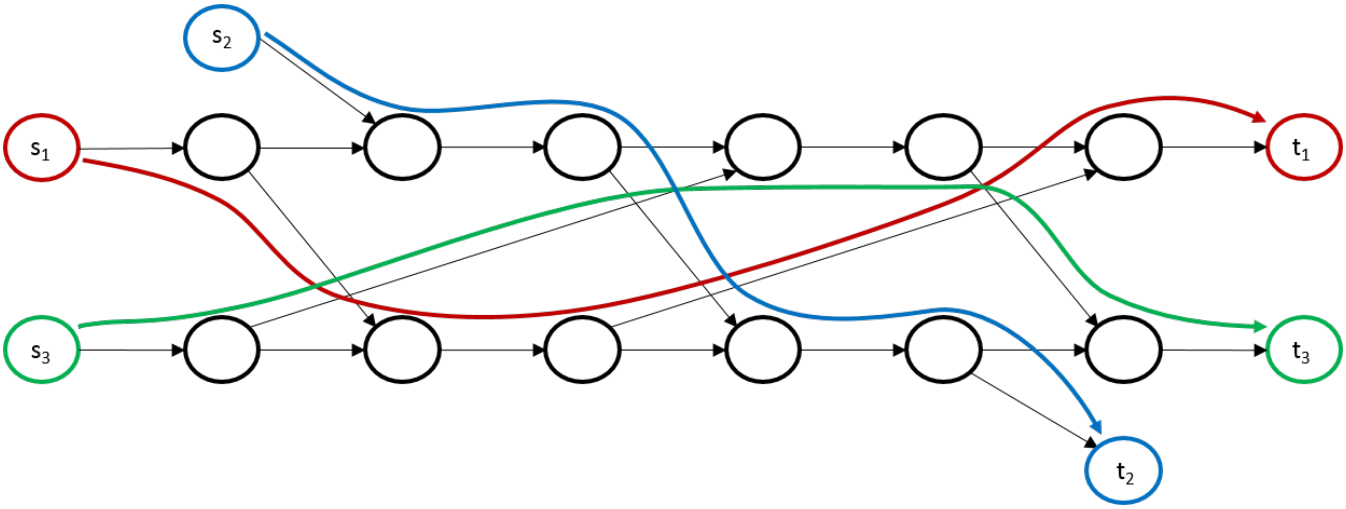
(b) An augmenting flow is found from s_2 to t_2.

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

Figure 4: 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) 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 5: Neither (part of) the red nor the green flow can consistently migrate to any imaginable flow in the network. Still, there is an augmenting flow moving both flows to other edges – which also respects the assignment of the sink-destination pairs. Hence, even an augmenting flow resulting in a proper multi-commodity flow does not guarantee a consistent migration for multiple sources and destinations.

8. REFERENCES

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

[2] 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.

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

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

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

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

[7] 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.

[8] 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.

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

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

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] R. Mahajan and R. Wattenhofer. On consistent updates in software defined networks. In D. Levine, S. Katti, and D. Oran, editors, *Twelfth ACM Workshop on Hot Topics in Networks, HotNets-XII, College Park, MD, USA, November 21-22, 2013*, pages 20:1–20:7. ACM, 2013.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

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

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

[25] 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.