Dynamic Flow Migration for Delay Constrained Traffic in Software-Defined Networks

Peter Danielis*, György Dán†, James Gross† and André Berger‡
*Faculty of Computer Science and Electrical Engineering, University of Rostock, Germany
†School of Electrical Engineering and the ACCESS Linnaeus Center, KTH Royal Institute of Technology, Sweden
‡School of Business and Economics, Maastricht University, The Netherlands
E-mail: peter.danielis@uni-rostock.de,{gyuri,james.gross}@kth.se, a.berger@maastrichtuniversity.nl

***Abstract*—Various industrial control applications have stringent end-to-end latency requirements in the order of a few milliseconds. Software-defined networking (SDN) is a promising solution in order to meet these stringent requirements under varying traffic patterns, as it enables the flexible management of flows across the network. Thus, SDN allows to ensure that traffic flows use congestion-free paths, reducing the delay to forwarding and processing delays at the SDN nodes. However, accommodating new flows at runtime is under such a setting challenging as it may require the migration of existing flows, without interrupting ongoing traffic. In this paper, we consider the problem of dynamic flow migration and propose a polynomial time algorithm that can find a solution if direct flow migration is feasible. We furthermore propose an algorithm for computing both direct and indirect flow migration and prove its correctness. Numerical results obtained on a FatTree network topology show that flow migration is typically necessary for networks with a moderate number of flows, while direct flow migration is feasible in around 60% of the cases.**

***Keywords*-SDN, dynamic flow migration, routing.**

I. Introduction

Future networks are facing new challenges due to new application requirements, mainly driven by the ongoing digitization drive in factory automation and process control and by emerging tactile applications, which has triggered an increasing interest in low-latency networking [1]. As an example, in industrial automation environments, a wide range of applications require a reliable real-time (RT) communication system that ensures the timely delivery of sensing or actuation data. A violation of communication deadlines may lead to unacceptable consequences like damage imposed on parts of the production facility. Recently, Industrial Ethernet (IE) systems, which can guarantee RT of below 1 ms, have emerged as serious competitor to traditionally used fieldbusses. As pointed out in [2], each IE system has however at least one drawback such as limited scalability in terms of the number of devices, insufficient self-configuration features, or increased costs due to the use of proprietary hardware instead of standard Ethernet hardware.

Therefore, previous works have developed new RT communication systems that overcome the drawbacks of the existing IE systems while providing comparable latency characteristics (below 1 ms) and independence of the used network topology [3], [4]. These works leverage the approach of Software-Defined Networking (SDN) and exploit the SDN controller's central view on the network to discover the topology and to collect the requirements of the applications using the network. Then, the collected information is used to compute paths for all required network flows that ensure compliance with the application requirements (e.g., bandwidth, maximum latency). The features provided by existing SDN technologies like OpenFlow are used to install the customized, application-specific flows in the network. However, the systems are currently solely able to compute paths for a fixed network configuration.

Extending these previous works, in this work we consider the problem of flow migration assuming unsplittable flows like in [5] using edge-disjoint paths, and adding one flow at a time. We propose novel algorithms for dynamic flow migration that migrate paths at runtime to accommodate for the new flow without interrupting ongoing flows. We use the proposed algorithms to address three fundamental questions related to the flow migration problem:

1) In what scenarios is a flow migration necessary to be able to accommodate a new flow?
2) If a flow migration is necessary, how many migration steps are necessary to accommodate a new flow and what is the proportion of direct flow migrations compared to indirect flow migrations?
3) How computationally expensive is flow migration as a function of the number of existing flows?

The rest of this paper is organized as follows. Section II gives an overview of SDN and OpenFlow and discusses related work. Section III presents the system model and the problem formulation. Section IV describes the algorithms for direct and indirect flow migration. Numerical results are shown in Section V, and Section VI concludes the paper.

II. Background and Related Work

A. SDN for Hard Real-time Communication

The SDN approach simplifies traditional network infrastructure devices (NIDs) by moving the control logic to a central SDN controller [6]. It is the SDN controller that computes routing decisions, and installs appropriate rules on NIDs to implement routing decision. Unlike the traditional distributed control logic, the implementation of essential parts of the

control logic, such as topology discovery is thus facilitated due to the controller's overview of the network. The interface between the SDN controller and the NIDs used to transfer rules is referred to as the Southbound API. Moreover, the controller usually provides for a Northbound API, which enables the communication with applications in the network, which could be used by the controller to learn the delay constraints of all applications.

One widely used protocol for conveying information over the Southbound API is the OpenFlow protocol [7] [6]. We will subsequently refer to OpenFlow-capable NIDs as OpenFlow switches. In addition to the specification of the communication between OpenFlow switches and the SDN controller, OpenFlow further defines the composition of forwarding rules. A forwarding rule is installed on a switch and will be applied to each incoming packet that contains matching header fields. If a switch does not have a matching forwarding rule available it requests an appropriate rule from the controller. It can be assumed that future IE switches support OpenFlow since it has already become the de-facto Southbound API standard [7], [8].

Owing to the advantages of the centralized control logic, SDN has recently been proposed as a means of enabling hard-real time communication for industrial control applications [3], [4]. The proposed solutions either use SDN for assigning time slots to flows across the network [4], or for enforcing flow-specific bandwidth allocations combined with static priority queuing in the switches [3]. While these works show the feasibility of using SDN for hard real-time communication, they have in common that they do not support changing networks, thus they do not allow adding new devices and flows into the network at runtime, without potentially interrupting existing flows. Our work addresses this gap, and could be used in combination with the flow-specific bandwidth allocation mechanism proposed in [3] for accommodating new flows.

B. Related Work

Flow migration (FM) in SDN has recently received some attention [9], [10], [5], [11]. In [9], each packet or flow is stamped with a version number to refer to old or new forwarding rules, and a packet/flow is solely routed with regard to one set of rules. Unfortunately, this approach does not always ensure a conflict-free FM in the case of migrating multiple flows at once. The SWAN approach is presented in [10], which finds conflict-free FM if a fraction of capacity (slack) is left on all paths. However, this assumption in not realistic in practice. The authors in [5] try to find a consistent migration ordering by executing a search in a dependency graph of possible migration steps. Like our work, they also assume unsplittable flows, which are only allowed to migrate as a whole to another path. They show the corresponding decision problem to be NP-hard under switch memory constraints. Unlike the algorithm we propose, the algorithm in [5] does not always find a conflict-free FM if a temporary flow migration to a path is required to achieve the final FM. To address this, certain flows are assumed rate-limited to enable a conflict-free FM. Brandt et al. show that given a network flow configuration, there is a polynomial-time algorithm to decide if a congestion-free migration is possible for the case of splittable flows [11]. However, they mention that the decision problem is NP-hard if all flows must be integer or are unsplittable.

Our work extends these recent works by proposing a polynomial time algorithm for deciding whether direct FM is feasible, and by evaluating the necessity and complexity of indirect FM. Although our work focuses on the algorithms for calculating forwarding rules, we also suggest a strategy how to deploy them on switches, namely changing the rules of one switch at a time. As opposed to the work in [12], which proposes to compute a network update schedule in order to minimize the overall network update time, our approach does not require switches to be synchronized.

III. System Model and Problem Formulation

We consider an SDN-enabled network and model it by a directed graph $\mathcal{G} = (V, E)$, where V is the set of vertices (the switches and the hosts), and E is the set of edges. There is a set $\mathcal{C} = \{(s_i, t_i) : 1 \leq i \leq N, s_i, t_i \in V\}$ of N source-destination host pairs that want to exchange data over the network. For a source-destination pair (s_i, t_i), $s_i, t_i \in V$, we call a sequence of edges in $\mathcal{G}$ from s_i to t_i a path, and define the length of path P, denoted by $l(P)$, to be the number of edges it contains.

We consider that the traffic from s_i to t_i has a specific delay constraint, and for simplicity we consider that the delay constraint can be met if the path from s_i to t_i has length at most $L_i \in \mathbb{N}^+$. For a pair (s_i, t_i) we define the set $\mathcal{P}_i = \{P_i | l(P) \leq L_i\}$ of length-constrained paths, and we use the notation $P_i \in \mathcal{P}_i$ to refer to a member of this set. Clearly, any of the paths $P_i \in \mathcal{P}_i$ allows s_i to send data to t_i. Given the set $\mathcal{C}$ of source-destination pairs, we say that a collection of paths $R = \{P_1, \ldots, P_N\}$ is valid if P_i and P_j are mutually edge-disjoint, and we refer to it as a route. A route allows the source-destination pairs in $\mathcal{C}$ to communicate simultaneously. Finally, we denote the set of all possible routes for the host pairs in $\mathcal{C}$ by $\mathcal{R}_\mathcal{C}$. Figure 1 illustrates the model, not showing the directions of the edges for simplicity.

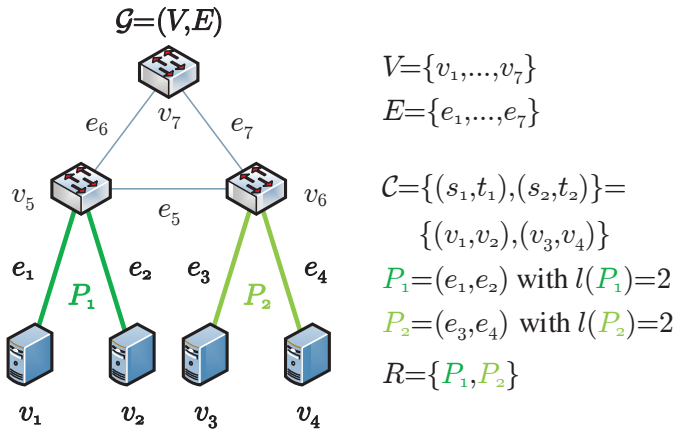

Fig. 1. Example network topology including source-destination pairs and paths.

We consider that the SDN controller has global information and can be used to update the forwarding rules of one switch at a time, e.g., using the SDN Southbound API. We refer to this one-step updating process in the following as *atomic forwarding table update*. The restriction of changing the rules of one switch at a time avoids unpredictable network behavior due to race conditions caused by imperfect synchronization between the switches. Atomic forwarding table updates could thus serve as atomic steps for lightweight path- and route migration, defined as follows.

Definition 1. *A flow migration (FM) from route R to route R' is a sequence $\mathcal{S} = (R^{(0)}, \ldots, R^{(K)})$ of $K \geq 1$ routes $R^{(k)}$, such that $R^{(0)} = R$ and $R^{(K)} = R'$, and route $R^{(k)}$ can be obtained from route $R^{(k-1)}$ by a sequence of atomic forwarding table updates.*

In what follows, our focus is on the feasibility of FM when the set $\mathcal{C}$ of source-destination pairs changes. Since the problem is trivial when a source-destination pair is removed, we focus on the case when a flow for a new source-destination pair is added to the existing source-destination pairs. Given a set $\mathcal{C}$ of source-destination pairs, a route $R = \{P_1, \ldots, P_N\}$ for $\mathcal{C}$, and a source destination pair (s_{N+1}, t_{N+1}), we want to find a FM $\mathcal{S}$ from route R to route $R' = \{P'_1, \ldots, P'_N\}$ such that there is a path P'_{N+1} for which $\{P'_1, \ldots, P'_N, P'_{N+1}\} \in \mathcal{R}_{\mathcal{C}'}$ is a route for the set $\mathcal{C}' = \mathcal{C} \cup \{(s_{N+1}, t_{N+1})\}$ of source-destination pairs. We refer to this as the Flow Migration Problem (FMP).

IV. Flow Migration Algorithms

It is important to note that a FMP may be infeasible, either because $\mathcal{R}_{\mathcal{C}'} = \emptyset$ or because there is no valid FM to any route R'. At the same time, for a feasible FMP a FM may or may not be needed. In the simplest case of the FMP there is a path $P'_{N+1} \in \mathcal{P}_{N+1}$ such that $\{P_1, \ldots, P_N, P'_{N+1}\}$ is a route, and thus there is no FM needed. If such a path does not exist, i.e., none of the paths $P'_{N+1} \in \mathcal{P}_{N+1}$ is edge disjoint with the paths in R, then route R needs to be reconfigured into a route $R' \in \mathcal{R}^* = \{(P'_1, \ldots, P'_N) | \exists P'_{N+1} s.t. (P'_1, \ldots, P'_N, P'_{N+1}) \in \mathcal{R}_{\mathcal{C}'}\}$.

In what follows we provide two algorithms for the FMP that rely on the assumption that the set $\mathcal{R}^*$ of routes can be computed. While in general the problem of computing edge-joint paths is NP-hard, for $N = 2$ [13] provides an FPT algorithm. Furthermore, the computation of $\mathcal{R}^*$ can be feasible on certain network topologies [14], such as the FatTree topology used in Section V.

The two algorithms we propose compute a sequence of FMs. Each FM is based on changing a single path at a time, which we call an elementary FM.

Definition 2. *Given a route $R = \{P_1, \ldots, P_N\}$, a route $R' = \{P_1, \ldots, \ldots, P_{i-1}, P'_i, P_{i+1}, \ldots, P_N\}$ is an elementary FM.*

The following lemma states that any elementary FM can be composed of a sequence of atomic forwarding table updates, and is thus a FM in the sense of Definition 1.

Lemma IV.1. *An elementary FM can be performed using a sequence of atomic forwarding table updates.*

Proof: Consider an elementary FM that involves changing a path P_i into a path P'_i. If P_i and P'_i are edge disjoint then we can install forwarding table entries for P'_i starting from t_i. Otherwise, let $(v, \ldots, v')$ be a segment of P'_i that is edge disjoint with P_i. We can install forwarding table entries starting at v'. ■

A. Direct Flow Migration (DFM)

The first algorithm we propose has polynomial complexity, but can only be used for computing a DFM, which is a permutation of the N source-destination host pairs such that switching individual paths from P_i to P'_i in the sequence using elementary FMs results in a sequence of routes.

Definition 3. *Given a route $R = \{P_1, \ldots, P_N\}$ and a route $R' = \{P'_1, \ldots, P'_N\}$, a DFM is a sequence $(i_1, \ldots, i_N)$ such that for every $1 \leq r \leq N$, $\{P'_{i_1}, \ldots, P'_{i_r}, P_{i_{r+1}}, ..., P_{i_N}\}$ is a route.*

Observe that any path in a DFM sequence may only be migrated once for the DFM algorithm to be able to find a solution and thus *at most* N elementary FMs can occur. A DFM may further not exist for arbitrary routes R and R', e.g., given $N = 2$ source-destination pairs and $P_1 = P'_2$ and $P_2 = P'_1$. In what follows, we provide an efficient algorithm for computing a direct FM if it exists. To formulate the result, let us define the directed graph $\mathcal{G}^* = (V^*, A^*)$, where $V^* = [N] = \{1, ..., N\}$, and there exists an $arc(i, j) \in A^*$ if and only if $P'_i \cap P_j \neq \emptyset$. Observe that $\mathcal{G}^*$ is in essence a conflict graph.

Lemma IV.2. *The direct FM problem has a feasible solution if and only if the graph $\mathcal{G}^*$ is acyclic. If $\mathcal{G}^*$ is acyclic, a feasible solution can be found in polynomial time.*

Proof: If $\mathcal{G}^*$ is acyclic, there must be a vertex $i_1 \in V^*$ with outdegree equal to zero, i.e., $P'_{i_1} \cap P_j = \emptyset$ for all $j \neq i_1$. Remove i_1 from the graph $\mathcal{G}^*$ and continue in the same manner to obtain the feasible sequence. On the other hand, if there exists a cycle $i_1, i_2, ..., i_{r_1}, i_r, i_1$ in $\mathcal{G}^*$, then in any feasible sequence i_1 has to appear after i_2, i_2 has to appear after i_3, etc., which is impossible. ■

Thus, given routes R and R', the algorithm provided in the proof of Lemma IV.2 can be used to decide if a DFM from R to R' is possible, and to compute the sequence of elementary FMs needed. We refer to the algorithm as the DFM algorithm. Nonetheless, even though a DFM is not existent the FMP may still be feasible.

B. Generic Flow Migration (GFM)

Unlike the above algorithm for DFM, the GFM algorithm can migrate a path multiple times. This allows *GFM* to find Indirect Flow Migrations (IFMs) in addition to DFMs. An IFM consists of at least 3 elementary FMs, and is characterized by that at least one path is migrated more than once in the FM sequence. The pseudo-code of the GFM algorithm is shown

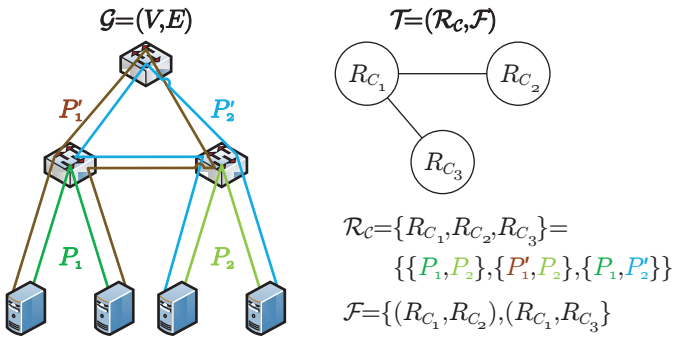

Fig. 2. Graph $\mathcal{T}$ constructed for the network in Figure 1.

Data: Graph $\mathcal{G}$,Routes $\mathcal{R}_C$,Set $\mathcal{C}'$
Result: Flow migration sequence $\mathcal{S}$
$\forall i$ compute $\mathcal{P}_i$;
Construct graph $\mathcal{T} = (\mathcal{R}_C, \mathcal{F})$, where $(R, R') \in \mathcal{F}$ if R' is an elementary FM of R ;
Let $\mathcal{R}^* = \{(P_1', \ldots, P_N') | \exists P_{N+1}' s.t. (P_1', \ldots, P_N', P_{N+1}') \in \mathcal{R}_{\mathcal{C}'}\}$;
Find shortest path from R to any $R^* \in \mathcal{R}^*$ in graph $\mathcal{T}$;
if $dist(R, R^*) < \infty$ **then**
  $\mathcal{S} = path(R, R^*)$;
else
  $\mathcal{S} = \emptyset$;
end

Algorithm 1: GFM algorithm

in Algorithm 1. The algorithms first computes the undirected graph $\mathcal{T} = (\mathcal{R}_C, \mathcal{F})$ in which the vertices are the routes, and there is an edge between R and R' if there is an elementary FM from R and R'. Next, the algorithm computes if R is connected to any of the routes in $\mathcal{R}^*$. If it is, then the algorithm computes the shortest path between route R and any route $R' \in \mathcal{R}^*$, and returns it as the FM sequence $\mathcal{S}$. Otherwise, it returns the empty sequence. As an illustration, Figure 2 shows the graph $\mathcal{T}$ constructed for the network in Figure 1.

Since the graph $\mathcal{T}$ is unweighted and undirected, the complexity of computing the shortest path, and thus the complexity of GFM, is $O(|\mathcal{R}_C| + |\mathcal{F}|)$ [15].

Proposition IV.3. *The GFM algorithm is correct, that is, if the FM problem has a solution, then the algorithm finds an FM. If the problem is infeasible, the GFM algorithm returns an empty FM sequence.*

Proof: Observe that the routes $\mathcal{R}^* \subseteq \mathcal{R}_C$, thus both R and $R' \in \mathcal{R}^*$ are vertices of $\mathcal{T}$. Due to Lemma IV.1, there is a path from R to some $R' \in \mathcal{R}^*$, corresponding to a valid sequence of atomic forwarding table updates, if and only if the FM problem has a solution. ■

C. Putting it all together

Upon arrival of a new flow between source-destination pair (s_{N+1}, t_{N+1}) an SDN controller would use the proposed

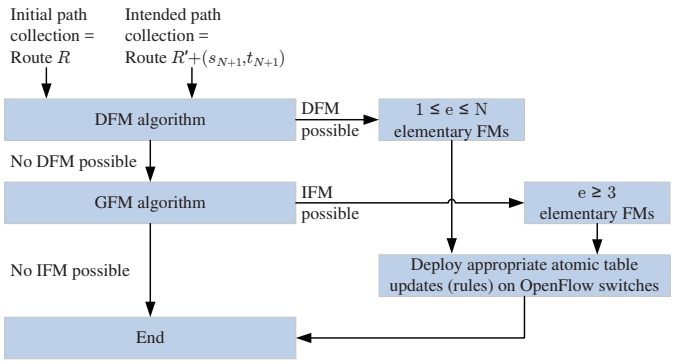

Fig. 3. Flowchart of the execution of the proposed algorithms by an SDN controller.

algorithms as shown in Figure 3. If a flow for the new source-destination pair (s_{N+1}, t_{N+1}) cannot be inserted without changing the paths of the existing flows, it tries to find a Direct Flow Migration (DFM) from the initial route R to the intended route R'. If a DFM exists, the controller deploys appropriate atomic forwarding table updates (rules) on the OpenFlow switches through the SDN Southbound API. If DFM is not possible, the controller uses the GFM algorithm to find an indirect flow migration (IFM), and deploys appropriate atomic forwarding table updates (rules) on the OpenFlow switches through the SDN Southbound API.

V. Performance Evaluation

In what follows we show numerical results to give insight into the feasibility and necessity of FM. The results were obtained in Matlab R2015b on a PC with 64 bit Windows 8, Intel i7-2600 CPU with 3.4 GHz, and 16 GB RAM. The focus of our evaluation is on the importance of flow migration, i.e., under what conditions is FM typically necessary and how many migration steps are typically required. Our evaluation is not concerned with practical aspects of the SDN Southbound API; an implementation based evaluation is subject of our future work.

A. Methodology

For the evaluation we consider the FatTree topology as shown in Figure 4. Unlike the line and star topologies, used in small-scale industrial networks today, which do not provide alternative paths between hosts, the FatTree topology is a hierarchical network topology designed for efficient and resilient networking in data centers. The FatTree topology provides redundant paths between hosts, and thus it could find adoption in industrial automation environments, as it may enable the emerging standards like Time Sensitive Networking (TSN) seamless redundancy (IEEE 802.1CB [16]).

The considered FatTree topology consists of 16 hosts connected by a network consisting of 20 OpenFlow switches. The labels for hosts and switches are based on [17]. For the evaluation we considered $N = \{1, ..., 7\}$ source-destination pairs on the FatTree topology with a length constraint of 6 edges. On the FatTree topology, if each node has upward and

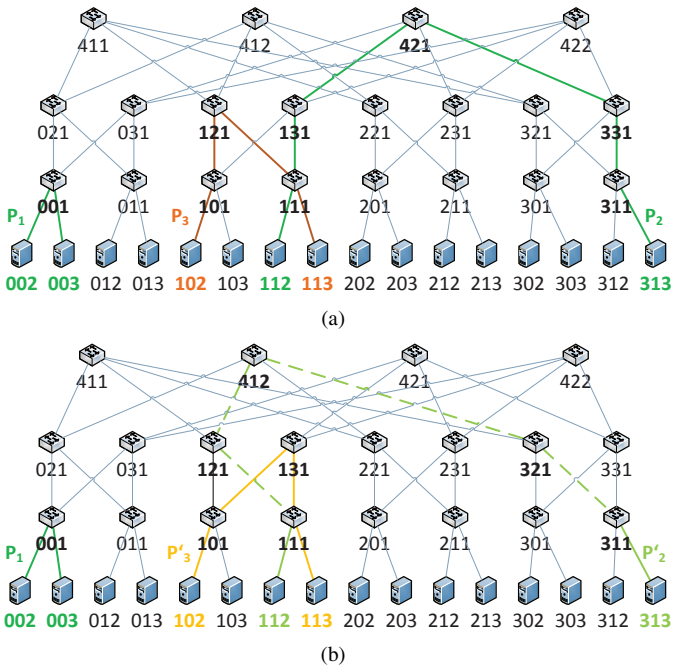

Fig. 4. FatTree topology with 16 hosts. (a) Path P_3 can be inserted without migrating the paths P_1 and P_2. (b) Path P_3' can only be inserted when migrating the path P_2 to P_2' through DFM. The SDN controller computing and deploying appropriate rules for FM is omitted for clarity.

downward degree $\leq d$, and the network has l levels, then for N source-destination pairs the set of routes has cardinality $|\mathcal{R}_\mathcal{C}| = (d^{2l})^N$ in the worst case.

For each N, we generated up to 5000 sets $\mathcal{C}$ of source-destination host pairs on the FatTree topology by choosing source and destination hosts at random without replacement. For each set $\mathcal{C}$ we used the APAC (All Paths And Cycle) algorithm proposed in [14] for computing all possible routes $\mathcal{R}^*$. For each set $\mathcal{C}$ and route R, we choose the $N+1$-st source-destination host pair at random, to create an instance of an FMP. Thus, for each N we consider thousands of FMPs.

As an example, for $N = 2$ a possible combination of host pairs is $\{(002, 003), (112, 313)\}$, for which a possible route , i.e., an initial collection of paths would be $R = \{P_1, P_2\}$ with $P_1 = (002, 001, 003)$ and $P_2 = (112, 111, 131, 421, 331, 311, 313)$ (cf. Figure 3), with path lengths $l(P_1) = 2$ and $l(P_2) = 6$, respectively. Together with a new host pair $(s_{N+1}, t_{N+1}) = (102, 113)$, these constitute a FMP. For this FMP, a path that could be inserted without flow migration would be $P_3 = (102, 101, 121, 111, 113)$ (see Figure 4(a)), i.e., $R' = R$. However, the path $P_3' = (102, 101, 131, 111, 113)$ can only be inserted if P_2 is migrated to $P_2' = (112, 111, 121, 412, 321, 311, 313)$, which needs a direct flow migration with one elementary FM (see Figure 4(b)), i.e., $R' = \{P_1, P_2'\} \neq R = \{P_1, P_2\}$.

Based on this evaluation set-up, we were initially interested in the number of routes (i.e. the cardinality of $\mathcal{R}^*$ for a given number of source-destination pairs) as it determines the complexity of formulating the FMP. Furthermore, we considered the number of elementary FMs needed for FM as it determines the time it takes to realize a FM, and finally the computational runtimes of solving the FMP.

TABLE I
TOTAL, AVERAGE, MAXIMUM, AND MINIMUM NUMBER OF ROUTES FOR 5000 RANDOMLY CHOSEN SETS OF HOST PAIRS.

N	Number of routes			
	Total	Average	Minimum	Maximum
1	18310	4	1	4
2	47617	10	1	16
3	90548	18	0	48
4	126445	25	0	144
5	116812	23	0	192
6	79572	16	0	128
7	40076	8	0	256

B. Numerical Results

We start with discussing our results on the cardinality of $\mathcal{R}^*$. Figure 5(a) shows the total number of routes over all considered sets $\mathcal{C}$ for each value of N. The figure shows that the number of routes is highest for $N = 4$, which means that the FMP is, on average, most difficult to formulate for a moderate number of flows. Table I shows the average, minimum, and maximum number of routes for each value N over all considered sets $\mathcal{C}$. As an example, for $N = 4$ there are on average 25 routes for each set $\mathcal{C}$ of host pairs, but there is a set $\mathcal{C}$ of host pairs with as many as 144 routes. It is interesting to note that while the average number of routes is highest for $N = 4$, the maximum number of routes increases with N. Figure 5(b) shows the number of sets $\mathcal{C}$ for which no route exists as a function of N. The figure confirms that as N increases, for the given FatTree topology it becomes more probable that there is no route, which also limits the possibility of FM.

Necessity of flow migration: To assess the necessity of FM, Figure 6(a) shows the share of (in)feasible FMPs for the considered instances as a function of N. We observe that the number of infeasible FMPs increases with increasing N, which is due to the fact that the probability increases that most paths in the FatTree topology are already occupied and no more path can be inserted. Furthermore, in Figure 6(b) we show the share of FMPs in which FM is (not) necessary among the feasible FMPs (while again varying N). For low values of N, FM is typically not necessary as enough paths are left to insert an additional path. However, as we increase N the share of instances for which FM is necessary increases and peaks for $N = 5$ with a share of 30%, and decreases afterwards again. This is due to the fact that for medium-sized N an edge-disjoint path may become available if the existing paths are migrated while there is a fairly high probability that migration is possible. Nonetheless, for high N FM rarely helps, as the additional path can either be inserted without any FM or (more often) the FMP is infeasible.

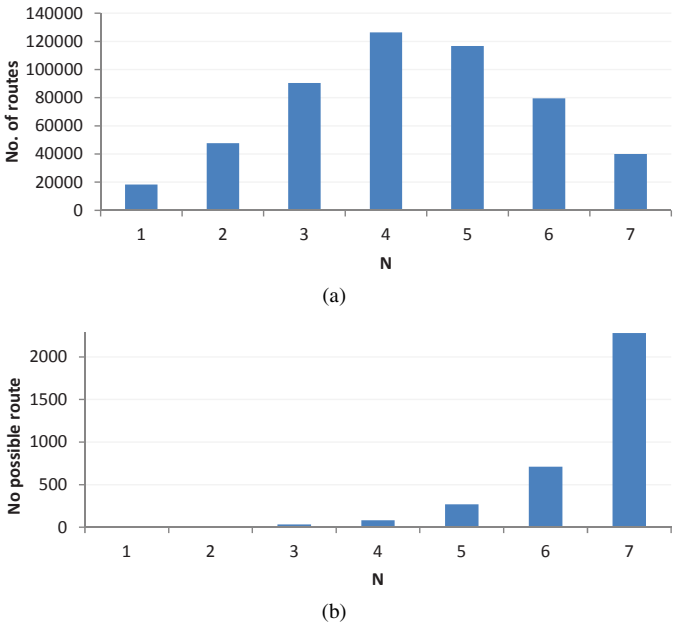

Fig. 5. (a) Total number of routes over all source-destination pair sets $\mathcal{C}$ and (b) Number of host-destination pairs for which no route exists, as a function of the number N of source-destination pairs.

TABLE II
SHARE OF CASES, IN WHICH A DIRECT FM IS POSSIBLE IF FM IS REQUIRED. $N = 1$ IS EXCLUDED AS A FM IS NOT NECESSARY IN ANY CASE.

N	SHARE OF DIRECT FM CASES [%]
2	92
3	76
4	65
5	62
6	63
7	73

Number of elementary FMs: Figure 6(c) shows the average number of elementary FMs for those FMPs for which FM is necessary (again as a function of N) and confirms the impact of N on the number of elementary FMs. The figure shows that the number of elementary FMs is increasing up to $N = 5$ (where as many as 5 elementary FMs may be needed) but for higher N the number of elementary FMs remains constant.

Direct vs. Indirect FM: Table II shows the share of FMPs for which a direct FM is sufficient among the feasible FMPs that require FM, as a function of N (note that the case $N = 1$ is not shown, as an FM is never required). The results show that for $62 - 92\%$ of all feasible FMPs that need FM, a direct FM is sufficient. The importance of this result is that a direct FM can be computed in polynomial time using the DFM algorithm proposed in Section IV-A. We can thus conclude that for lightly loaded and heavily loaded topologies FM is either not necessary or the FMP is infeasible, hence FM cannot help. Nonetheless, for moderately loaded FatTree topologies (4 to 6 flows) a significant share of the FMPs requires FM (24 % to 30 %). In those cases, roughly in 60% of the cases, the direct FM algorithm solves the FMP.

Computation time: Finally, we evaluate the computation time needed to execute the GFM algorithm to assess the practical feasibility of FM. Table III shows the average time for executing the GFM algorithm (as implemented by us under Matlab) over an increasing N. The results show that the computation time increases with N, mainly due to the increasing complexity of computing $\mathcal{T}$. To further evaluate the dependence of the computation time on N, Figure 7 shows the CDF of the execution time of the GFM algorithm, as implemented by us under Matlab. The figure reveals that as N increases the ratio of longer computation times increases (for high N we observe for instance that 10 % of the execution times are 15 s or longer). Furthermore, while for low N almost all values are concentrated around the mean, for higher values of N the variance increases significantly. This is further evidence of the increasing complexity of determining $\mathcal{T}$ as N increases, which motivates the need for the scalable DFM algorithm as outlined in Section IV-A.

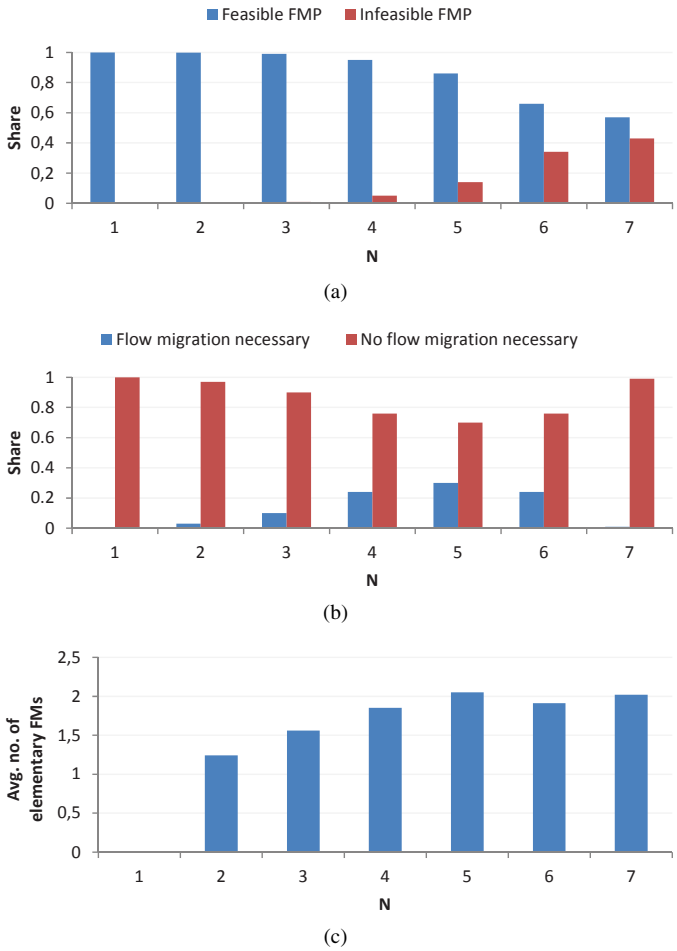

Fig. 6. (a) Share of (in-)feasible FMPs, (b) share of feasible FMPs where FM is (un)necessary, and (c) average number of elementary FMs if FM is necessary, as a function of the number N of source-destination pairs.

TABLE III
AVERAGE COMPUTATION TIME AND ITS STANDARD DEVIATION FOR THE EXECUTION OF THE GFM ALGORITHM.

N	AVG. EXECUTION TIME [S]
1	8.32±0.02
2	9.89±0.02
3	10.90±0.01
4	12.67±0.02
5	13.76±0.02
6	13.87±0.02
7	14.76±0.01

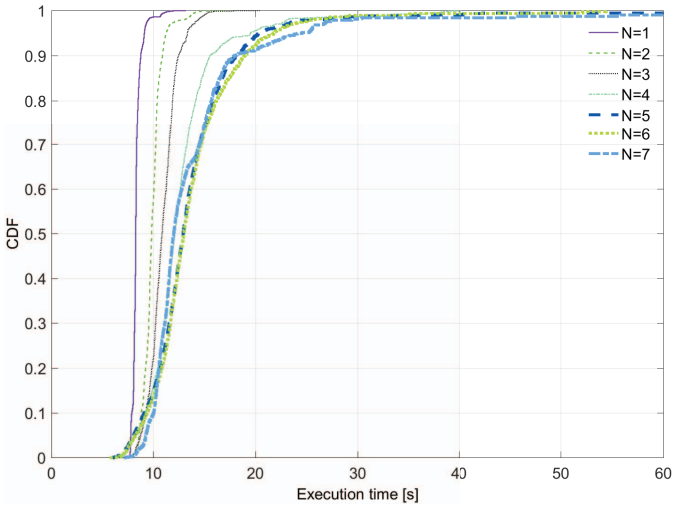

Fig. 7. CDF of the execution times of the GFM algorithm.

VI. CONCLUSION

In this paper, we addressed the problem of dynamic flow migration in software-defined networks. We developed two algorithms that allow conflict-free direct and indirect flow migration and thus enable inserting an additional flow at runtime. The proposed algorithm for direct flow migration runs in polynomial time but does not always find a solution to the flow migration problem. The generic algorithm finds all solutions to the flow migration problem in case it is feasible at all, but at the price of an increased computational complexity. Our results show that FM is required in 24 % up to 30 % of all cases for a FatTree topology with moderate traffic, and the necessary flow migrations typically require several migration steps. We also found that for the investigated topologies direct flow migration is possible in 62 % up to 92 % of the cases.

As part of our future work, we plan to evaluate the algorithms in a simulation environment that allows deploying atomic forwarding table updates and we will consider other topologies and compare our algorithms to existing solutions. Furthermore, we plan to relax the requirement of edge-disjointness and consider weighted graphs to allow for expressing finer-grained delay constraints.

ACKNOWLEDGMENT

The first author would like to thank the German Research Foundation (DFG) (research fellowship, GZ: DA 1687/2-1) for their financial support.

REFERENCES

[1] G. P. Fettweis, "The tactile internet: Applications and challenges," *IEEE Vehicular Technology Magazine*, vol. 9, no. 1, pp. 64–70, March 2014.

[2] P. Danielis, J. Skodzik, V. Altmann, E. Schweissguth, F. Golatowski, D. Timmermann, and J. Schacht, "Survey on real-time communication via ethernet in industrial automation environments," in *IEEE Intl. Conf. on Emerging Technology and Factory Automation (ETFA)*, Sep. 2014, pp. 1–8.

[3] J. W. Guck and W. Kellerer, "Achieving end-to-end real-time quality of service with software defined networking," in *IEEE Intl. Conf. on Cloud Networking (CloudNet)*, Oct. 2014, pp. 70–76.

[4] E. Schweissguth, P. Danielis, C. Niemann, and D. Timmermann, "Application-aware industrial ethernet based on an sdn-supported tdma approach," in *IEEE World Conference on Factory Communication Systems (WFCS)*, May 2016, pp. 1–8.

[5] X. Jin, H. H. Liu, R. Gandhi, S. Kandula, R. Mahajan, M. Zhang, J. Rexford, and R. Wattenhofer, "Dynamic scheduling of network updates," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 4, pp. 539–550, Aug. 2014.

[6] "Software-defined networking: The new norm for networks," White Paper, Open Networking Foundation, Apr. 2012. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf

[7] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: Enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, Apr. 2008.

[8] N. Feamster, J. Rexford, and E. Zegura, "The road to sdn: An intellectual history of programmable networks," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 2, pp. 87–98, Apr. 2014.

[9] M. Reitblatt, N. Foster, J. Rexford, C. Schlesinger, and D. Walker, "Abstractions for network update," in *Proc. of ACM SIGCOMM*, 2012, pp. 323–334.

[10] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer, "Achieving high utilization with software-driven wan," *SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 15–26, Aug. 2013.

[11] S. Brandt, K. T. Förster, and R. Wattenhofer, "On consistent migration of flows in sdns," in *Proc. of IEEE Infocom*, April 2016, pp. 1–9.

[12] J. Zheng, G. Chen, S. Schmid, H. Dai, and J. Wu, "Chronus: Consistent data plane updates in timed sdns," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, June 2017, pp. 319–327.

[13] L. Cai and J. Ye, "Finding two edge-disjoint paths with length constraints," 2015. [Online]. Available: http://arxiv.org/abs/1509.05559

[14] R. Simões, "Apac: An exact algorithm for retrieving cycles and paths in all kinds of graphs," *Tékhne-Revista de Estudos Politécnicos*, no. 12, pp. 39–55, 2009.

[15] M. Thorup, "Undirected single-source shortest paths with positive integer weights in linear time," *J. ACM*, vol. 46, no. 3, pp. 362–394, May 1999. [Online]. Available: http://doi.acm.org/10.1145/316542.316548

[16] TSN Task Group, "802.1cb - frame replication and elimination for reliability," Draft, 2017. [Online]. Available: http://www.ieee802.org/1/pages/802.1cb.html

[17] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 4, pp. 63–74, Aug. 2008.