On the Consistent Migration of Unsplittable Flows: Upper and Lower Complexity Bounds

Klaus-Tycho Foerster
ktfoerster@cs.aau.dk
Aalborg University, Denmark

***Abstract*—In consistent flow migration, the task is to change the paths the flows take in the network, but without inducing congestion during the update process. Even though the rise of Software Defined Networks allows for centralized control of path changes, the execution is still performed in an inherently asynchronous system, the switches distributed over the network.**

To this end, a multitude of scheduling systems have been proposed since the initial papers of Reitblatt et al. (*Abstractions for Network Update*, SIGCOMM '12) and Hong et al. (*SWAN*, SIGCOMM '13). While the the complexity of consistently migrating splittable flows is well understood, for the practically more relevant unsplittable flows, few non-heuristic results are known – and upper complexity bounds are missing.

We give a dynamic programming algorithm for unsplittable flows, showing the containment in EXPTIME, for both computation time and schedule length. In particular, there are cases where flows must switch between paths back and forth repeatedly: as thus, flow migration is not just an ordering problem.

We also study lower bounds and show NP-hardness already for two flows, via reduction from edge-disjoint path problems.

Lastly, we also discuss some practical application cases for our dynamic programming algorithm, furthermore showing how it can be extended to min-max load considerations.

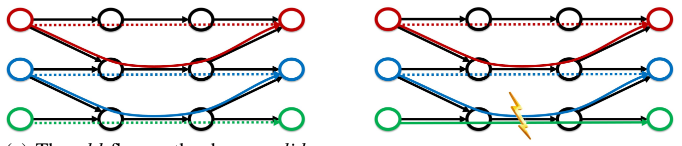
I. Introduction

Traffic demands change all the time in networks [1], [2], [3], requiring the use of good traffic engineering techniques unless performance is to be sacrificed. As such, it can be necessary to change the path allocation for live traffic flows. In fact, optimal traffic engineering requires the ability to swap flow paths, if traffic arrives in an online fashion [4].

The toolkit of Software Defined Networking (SDN) allows the network operator to take central control of such network reconfigurations, no longer relying on distributed protocols. To this end, major operators such as Microsoft [1] or Google [2] optimize their WAN traffic engineering every few minutes. While the network will be in an enhanced state afterwards, inconsistencies can occur during the network update, such as blackholes, forwarding loops, or congestion [3], [5].

In this short paper, we focus on avoiding congestion during the network update process. First studied by Hong et al. [1] in SDNs, the problem is to move flows from their old to their new paths, but without inducing bandwidth violations in the process due to asynchrony, see Fig. 1 for an example.

There has been a wide landscape of system proposals, which compute flow migration schedules in order to avoid transient congestion, e.g., [1], [6], [7], [8], [9].



(a) The *old* flow paths drawn *solid*, the *new* flow paths are *dotted*.

(b) If the new green bottom flow is inserted first, congestion occurs.

Fig. 1. In this introductory example, all flows and edges have a size resp. capacity of one. In 1a, two flows (in red and blue) are in the network, which need to be moved for the green flow to be inserted without congestion. If all three operations are issued simultaneously, the green flow could be inserted before the other flows have moved, as in 1b. However, three updates suffice to update consistently: first move the top red flow, then the middle blue flow, and lastly insert the green flow.

However, while the computational complexity for splittable flows is polynomial [10], no upper complexity bounds for unsplittable flows are known [11, Table 3]. On the other end, it is known that a linear number of unsplittable flows are NP-hard to migrate consistently, but further lower bounds are unknown as well, except in the related node-ordering model of Amiri et al. [12], discussed in Section VI, along with related work.

Contribution. We present a dynamic programming algorithm for the consistent migration of unsplittable flows, generating a schedule of optimal length in exponential time (§ III). To motivate our super-polynomial runtime, we show in § II that flow migration is not just an ordering problem, but can require to move flows back and forth repeatedly. In particular, already the migration of two flows is an NP-hard problem. We investigate applications in § IV where our dynamic programming algorithm has sub-exponential worst-case runtimes, extending our approach min-max load considerations in § V. Lastly, we conclude in § VII with an open complexity question.

Model A network N is a directed graph $G=(V,E)$, $|V|=n$, with edge capacities $c(e)$. An unsplittable flow F is routed along a simple path from s to t, with a multi-commodity flow denoted by $\mathcal{F}=\{F_1,F_2,\ldots,F_k\}$, respecting capacity constraints. We assume all flows to be unsplittable.

A *network update* is a tuple $(N,\mathcal{F},\mathcal{F}')$, where $\mathcal{F}$ describes the *old* and $\mathcal{F}'$ the *new* flow paths. W.l.o.g., we can assume that a flow F_i, F_i' has the same size before and after the update [10], [1], as else can could decrease its size before/increase after.

Due to asynchrony, the individual flows can update in any order [1], [7], cf. Fig. 1. Hence, Hong et al. [1] call a network update *consistent*, if no bandwidth violations occur independently whether the flows are on their old or new path:

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

Lastly, a schedule of consistent network updates, using 2-phase commits [3], is called a *consistent flow migration*, where the flows may be on intermediate paths not contained in $\mathcal{F}, \mathcal{F}'$.

As such, we study the following problem in this paper:[1]

- Given a network N with old and new flow paths $\mathcal{F}, \mathcal{F}'$, is there a consistent flow migration from $\mathcal{F}$ to $\mathcal{F}'$?

II. Order is not everything

Before presenting our **EXPTIME**-algorithm in Section III, we would like to motivate with two examples why we conjecture consistent flow migration not to be in **NP**.

- In Section II-A, we show that flow migration is already **NP**-hard for *two* unsplittable flows, due to exponentially many intermediate path options.
- In Section II-B, we show that flow migration is not an ordering problem – flows need to be moved repeatedly.

A. Two flows are NP-hard

Taking a step back, in mathematical flow theory, splittable flow problems usually turn out to be in **P**, while unsplittable ones are mostly **NP**-hard, already for two flows [16]. Single-source/destination problems are an (easier) exception though, i.e., for a single flow. We will now show that the situation is analogous for flow migration. By definition, a single flow can always be migrated in a consistently in a single update:

Observation 1. *For one flow, consistent migration is in **P**.*

There is a large hardness-gap though. Except for in the node-ordering model of Amiri et al. [12] (§VI-A), previous work only showed consistent migration to be **NP**-hard for a linear number of unsplittable flows, via, e.g, PARTITION [9]. We will now show the **NP**-hardness already for two flows, via reduction from two two edge-disjoint path problem:

Theorem 1 ([17])**.** *Let $G = (V, E)$ be a directed graph, with $s_1, t_1, s_2, t_2 \in V$. Finding two edge-disjoint paths, one from s_1 to t_1, and one from s_2 to t_2, is **NP**-complete.*

The fundamental underlying problem is that there can be exponentially many path options between two nodes, turning the joint optimization of already two paths intractable.

Theorem 2. *For two flows, consistent migration is **NP**-hard.*

Proof: Reduction from the two directed edge-disjoint path problem: all edges and flows will have capacity or size of 1. For an instance I with $G = (V, E)$, $s_1, t_1, s_2, t_2 \in V$, we can assume w.l.o.g. that individual paths P_i from s_i to t_i, $i \in \{1, 2\}$ exist. We add new nodes s, t, connecting both s_1, s_2 to s; s to t; and t to t_1, t_2. We now set F_1 as s_1, s, t, t_1 and F_2 as s_2, P_2, t_2, and set F_1' as s_1, P_1, t_1 and F_2' as s_2, s, t, t_2. In order to migrate the unsplittable flows consistently, we need to "store" two edge-disjoint flows from s_i to t_i, $i \in \{1, 2\}$ in I, else the capacity constraints on the edge from s to t would be violated, finishing the reduction. ■

[1]We defer the case that N changes, e.g., due to link failure [5], [13] resp. being taken offline for maintenance work [14] or variable link capacities [15].

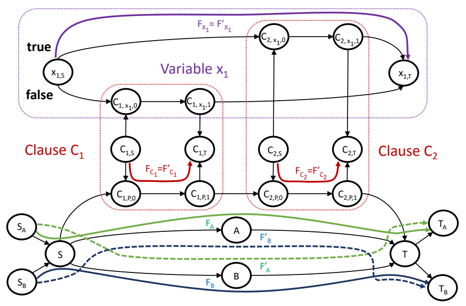


Fig. 2. Taken from [10, Figure 3]. In order to swap the green and blue flow, s.t. they reach their new dashed paths, the purple variable flow must be assigned in such a way that both clause flows can free up a swapping path. In this case, the instance is not solvable, analogous to the formula $(x_1) \wedge (\neg x_1)$.

B. Flows need to move back and forth repeatedly

As observed by Jin et al. [7, Appendix A], consistent flow migration is in **NP** if every flow is only allowed to be moved once: at most, every flow is moved in a solitary fashion, meaning the total schedule length is linear.

Subsequently, Brandt et al. [10] devised an **NP**-hardness reduction from 3-SAT, where some flows must be moved twice: once to an intermediate path[2], and secondly, to their final path [10, Theorem 4.2]. The reduction relies on providing free capacity for flow swaps, which requires solving 3-SAT.

It is easy to extend their construction, illustrated in Fig. 2, to require some flows to be moved back and forth arbitrarily often. Namely, we create instances from multiple 3-SAT formulas, but let them all share the variable flows, depicted in purple in Fig. 2. Then, the purple variable flows must cycle through the corresponding formula truth assignments, switching back and forth between both the true and the false paths repeatedly.

We cast our insights into the following Theorem 3.

Theorem 3. *A consistent flow migration algorithm, which is restricted to move each flow only a constant number of times, cannot solve all feasible instances.*

III. An EXPTIME-Algorithm

As seen in Section II, polynomial algorithms, already for 2 flows, would imply $\mathbf{P} = \mathbf{NP}$. We thus go beyond polynomial algorithms in this section, presenting an **EXPTIME** scheme.

As no containment in a complexity class is known [11, Table 3], it can make sense to consider restricted models of flow migration, especially in light of Theorem 2. E.g., if every flow may only move once, flow migration is in **NP** [7]. To this end, the popular *path* models for flow migration are:

- ***Per-packet*** consistency [3], where flow packets may only take the old or new path from $\mathcal{F}, \mathcal{F}'$, respectively.
- Each packet may only traverse the union of old and new path [19], which we call ***mixed***.
- No restrictions, which we call the ***general*** model.

Now, instead of analyzing the three models seperately, we can cover them all with combined arguments. Interestingly, our dynamic programming approach will also find an *optimal* schedule with a *minimum* number of updates.

[2]Of independent interest might be an analogous flow migration scheme inspired by Valiant load balancing [18], where flows are "parked" randomly.

Theorem 4. *Via dynamic programming, unsplittable {per-packet, mixed, general} consistent flow migration can be solved with minimum schedule length in the following runtime, where #paths denotes the maximum number of paths available:*

$$\mathcal{O}\left(\left((\#paths)^{|\mathcal{F}|}\right)^2\right) \cdot poly(n) \qquad (1)$$

Proof: As each flow $F \in \mathcal{F}$ can only take #paths in the network, there are only $\mathcal{O}(\#\text{paths}^{|\mathcal{F}|})$ many possibilities in which state the network can be after an update. We store each such state in a table, marking each state as not yet reachable, except for $\mathcal{F}$. We then traverse the table, beginning from the initial state $\mathcal{F}$, checking what other states are reachable. Similar to the argumentation used when proving Dijkstra's algorithm (for unit edge lengths), we then continue from any state that is reachable in one update. In further steps, we always take a state $\mathcal{F}^*$ of minimum schedule length, and see which other states are reachable, marking $\mathcal{F}^*$ as checked afterward. As thus, we need to check only $\mathcal{O}\left(\left((\#\text{paths})^{|\mathcal{F}|}\right)^2\right)$ many updates for consistency (similar to running Dijkstra on a complete graph where every edge has either cost 1 or ∞). For unsplittable flows, each check can be performed in polynomial time. ■

Observe that the optimal schedule length is therefore bounded by $\mathcal{O}\left((\#\text{paths})^{2|\mathcal{F}|}\right)$, if a feasible one exists. For our next corollary, we quickly also introduce some notation: following Immerman [20, p.26], **EXPTIME** is defined as $\bigcup_{k=1}^{\infty} \textbf{DTIME}\left[2^{n^k}\right]$, which can be simplified as $2^{\text{poly}(n)}$.

Corollary 1. *Both the computation and number of updates needed, for an* optimal *scheduling of unsplittable {per-packet, mixed, general} consistent flow migration, is in* ***EXPTIME***.

We note that containment in **NP** is not even clear for two flows, as the feasible state table has exponentially many entries.[3]

IV. Polynomial Implications

While showing unsplittable flow migration to be in **EXPTIME** settled an open complexity question, exponential runtimes are usually beyond reach for time-critical approaches. We thus present two polynomial examples in this section.

Per-packet consistency for $O(\log n)$ unsplittable flows. The popular per-packet consistency model by Reitblatt et al. [3] restricts each flow to routing rules on either its old or its new path. Hence, applying Term 1 with $\#\text{paths} = 2$, we obtain a runtime of $\mathcal{O}(2^{\mathcal{O}(\log n)}) \cdot \text{poly}(n)$ for unsplittable flows. I.e., this restricted flow migration problem is polynomial.

Mixed consistency of unsplittable flows in the data center. The mixed model proposed by Nguyen et al. [19] can be seen as an extension of per-packet consistency, as flow-packets may now take any mix of the old and new routing rules. Still, in the worst case, the old and new paths can intersect a linear number of times, meaning *#paths* can be as high as $2^{\Omega(n)}$.

If we restrict ourselves to topologies of small path length however, we can obtain tractable migrations of the mixed model. E.g., in constant-level fat trees, using valley-free routing, each path length is constrained to $\mathcal{O}(1)$. Hence, mixed consistent flow migration of $|\mathcal{F}|$ unsplittable flows can be computed in a runtime of $\mathcal{O}(2^{\mathcal{O}(|\mathcal{F}|)}) \cdot \text{poly}(n)$, allowing for $\mathcal{O}(\log n)$ flows.

[3]Another approach could be a combinatorial classification, conceptually similar to solving the so-called 15-puzzle or multi-agent pathfinding [21].

V. Minimizing transient congestion

In case our dynamic programming approach outputs that no feasible schedule exists, there is no course of action recommended on how to proceed. However, as observed in, e.g., [9], we can then ask what the minimum transient congestion is that the network has to endure for migration.

Min-max load with polynomial overhead. We reformulate this idea by optimizing for the relative min-max load during the migration process, implemented by scaling every edge capacity by a global factor $\alpha \geq 1$. Observe that α only needs to be chosen in a discrete range to cover all $2^{|\mathcal{F}|}$ possible flow combinations, for every edge, then sorting the αs globally by size. Hence, for unsplittable flows, we just need to test $\mathcal{O}(|E| \cdot 2^{|\mathcal{F}|})$ different values of α, i.e., only an extra factor of $\mathcal{O}(|\mathcal{F}| + \log(|E|))$ iterations, via binary search over the range. As thus, polynomial algorithms remain polynomial.

Even if consistent migration is possible with $\alpha = 1$, by allowing for $\alpha < 1$, we can minimize the relative maximum load during the migration process with the same idea.

Traversing the pareto frontier. Using our dynamic programming approach, we can now also evaluate the trade-off between optimum schedule length and acceptable congestion or load, completely exploring the problem space in both dimensions.

VI. Further Related Work

Dynamically changing the flow of traffic has also been considered in traditional networks [22], [23], but it was not before SDNs that consistency during the network update was studied extensively in the context of flow migration.

The main selling point of SDNs is the separation of data and control planes, allowing a (logically) centralized controller to optimize previously distributed algorithms from an omniscient point of view [24]. E.g., Google's *B4* [2] optimizes the network based on global information every 2-3 minutes, using customized hardware & protocols to perform the updates quickly, thus minimizing turmoil effects during the update. There is a multitude of consistency properties to check during updates, with avoiding blackholes [5], loop freedom [5], [25], [26], [27], [28], [29], [30] with waypointing [31], [32], [33], packet coherence (rules from before and after the network update do not mix) [3], [34], and bandwidth violations [1] being the main focus of research [5]. Recent work also considered consistent updates in optical networks [35], [15]. Model checking is used to for consistent updates as well, but cannot guarantee congestion freedom yet [36]. Time synchronization has been studied in [4], [37], [38], [39]. For a summary of the state of the art of consistent updates, we refer to [11]. In specialized environments as single-tenant Data-Centers, one can also take an orthogonal approach to network updates, by scheduling all the traffic, eliminating the need for the migration of flows [40], [41], [42].

A. Node-ordering: Flow migration without header changes

Conceptually different yet strongly related to our work is the node-ordering model of Amiri et al. [12]. Instead of using the 2-phase commit [3] technique as in our and prior work, they propose a powerful new model: forwarding rules are modified in the nodes themselves, switching from old to new. As thus, they can omit modifying packet headers ("tagging"), making their approach more versatile with less overhead. Amiri et al. [12] prove that in the node-ordering model, NP-hardness already holds for the migration of two flows on general graphs. On directed acyclic graphs, they show NP-hardness in general, but also provide an intricate polynomial algorithm for a constant number of flows. A translation of complexity results between both models would be of great interest for future work.

VII. Conclusion

In this paper, we studied upper and lower complexity bounds for the consistent migration of unsplittable flows.

Regarding lower bounds, we proved that already two flows pose an **NP**-hard problem. Furthermore, we showed that consistent flow migration is not just an ordering problem, it can be necessary to move flows back/forth arbitrarily often.

With respect to upper bounds, no results were previously known to the best of our knowledge. Via dynamic programming, we settled that optimal unsplittable consistent flow migration is in **EXPTIME**, also extending our algorithm to min-max load optimizations. Future work will have to cover if **EXPTIME** is the correct answer, or if it is complete in **NP** or **PSPACE**.

Acknowledgments The author gratefully acknowledges Sebastian Brandt for his comments and assistance on an earlier draft of this manuscript. Klaus-Tycho Foerster is supported by the Danish VILLUM FONDEN project ReNet.

References

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

[2] 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 *SIGCOMM*, 2013.

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

[4] T. Mizrahi and Y. Moses, "On the necessity of time-based updates in SDN," in *ONS*, 2014.

[5] K.-T. Foerster, R. Mahajan, and R. Wattenhofer, "Consistent updates in software defined networks: On dependencies, loop freedom, and blackholes," in *NETWORKING*, 2016.

[6] S. Brandt, K.-T. Foerster, and R. Wattenhofer, "Augmenting flows for the consistent migration of multi-commodity single-destination flows in sdns," *Pervasive Mob. Comput.*, vol. 36, pp. 134–150, 2017.

[7] X. Jin, H. H. Liu, R. Gandhi, S. Kandula, R. Mahajan, M. Zhang, J. Rexford, and R. Wattenhofer, "Dynamic scheduling of network updates (extended version)," in *SIGCOMM*, 2014.

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

[9] J. Zheng, H. Xu, G. Chen, and H. Dai, "Minimizing transient congestion during network update in data centers," in *ICNP*, 2015.

[10] S. Brandt, K.-T. Foerster, and R. Wattenhofer, "On consistent migration of flows in sdns," in *INFOCOM*, 2016.

[11] K.-T. Foerster, S. Schmid, and S. Vissicchio, "Survey of consistent network updates," *CoRR*, vol. abs/1609.02305, 2016.

[12] S. A. Amiri, S. Dudycz, S. Schmid, and S. Wiederrecht, "Congestion-free rerouting of flows on dags," *CoRR*, vol. abs/1611.09296, 2016.

[13] M. Borokhovich and S. Schmid, "How (Not) to Shoot in Your Foot with SDN Local Fast Failover – A Load-Connectivity Tradeoff," in *OPODIS*, 2013.

[14] D. Zhuo, M. Ghobadi, R. Mahajan, K.-T. Foerster, A. Krishnamurthy, and T. E. Anderson, "Understanding and mitigating packet corruption in data center networks," in *SIGCOMM*, 2017.

[15] R. Singh, M. Ghobadi, K.-T. Foerster, M. Filer, and P. Gill, "Run, walk, crawl: Towards dynamic link capacities," in *HotNets*, 2017.

[16] S. Even, A. Itai, and A. Shamir, "On the complexity of timetable and multicommodity flow problems," *SIAM J. Comput.*, vol. 5, no. 4, pp. 691–703, 1976.

[17] S. Fortune, J. E. Hopcroft, and J. Wyllie, "The directed subgraph homeomorphism problem," *Th. Comp. Sci*, vol. 10, pp. 111–121, 1980.

[18] L. G. Valiant, "A scheme for fast parallel communication," *SIAM J. Comput.*, vol. 11, no. 2, pp. 350–361, 1982.

[19] T. D. Nguyen, M. Chiesa, and M. Canini, "Decentralized fast consistent updates," in *SOSR*, 2017.

[20] N. Immerman, *Descriptive complexity*. Springer, 1999.

[21] K.-T. Foerster, L. Groner, T. Hoefler, M. König, S. Schmid, and R. Wattenhofer, "Multi-agent pathfinding with n agents on graphs with n vertices: Combinatorial classification and tight algorithmic bounds," in *CIAC*, 2017.

[22] E. Anderson and T. E. Anderson, "On the Stability of Adaptive Routing in the Presence of Congestion Control," in *INFOCOM*, 2003.

[23] R. Gao, D. Blair, C. Dovrolis, M. Morrow, and E. W. Zegura, "Interactions of intelligent route control with TCP congestion control," in *NETWORKING*, 2007.

[24] K. Kirkpatrick, "Software-defined networking," *CACM*, vol. 56, no. 9, pp. 16–19, 2013.

[25] S. A. Amiri, A. Ludwig, J. Marcinkowski, and S. Schmid, "Transiently consistent SDN updates: Being greedy is hard," in *SIROCCO*, 2016.

[26] S. Dudycz, A. Ludwig, and S. Schmid, "Can't Touch This: Consistent Network Updates for Multiple Policies," in *DSN*, 2016.

[27] K.-T. Foerster, T. Luedi, J. Seidel, and R. Wattenhofer, "Local checkability, no strings attached: (a)cyclicity, reachability, loop free updates in sdns," *Theoretical Computer Science*, 2016.

[28] K.-T. Foerster and R. Wattenhofer, "The power of 2 in consistent network updates: Hard loop freedom, easy flow migration," in *ICCCN*, 2016.

[29] A. Ludwig, J. Marcinkowski, and S. Schmid, "Scheduling Loop-free Network Updates: It's Good to Relax!" in *PODC*, 2015.

[30] L. Vanbever, S. Vissicchio, C. Pelsser, P. Francois, and O. Bonaventure, "Lossless migrations of link-state igps," *Trans. Netw.*, vol. 20, no. 6, pp. 1842–1855, 2012.

[31] A. Ludwig, S. Dudycz, M. Rost, and S. Schmid, "Transiently Secure Network Updates," in *SIGMETRICS*, 2016.

[32] A. Ludwig, M. Rost, D. Foucard, and S. Schmid, "Good Network Updates for Bad Packets: Waypoint Enforcement Beyond Destination-Based Routing Policies," in *HotNets*, 2014.

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

[34] P. Cerny, N. Foster, N. Jagnik, and J. McClurg, "Optimal consistent network updates in polynomial time," in *DISC*, 2016.

[35] X. Jin, Y. Li, D. Wei, S. Li, J. Gao, L. Xu, G. Li, W. Xu, and J. Rexford, "Optimizing bulk transfers with software-defined optical WAN," in *SIGCOMM*, 2016.

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

[37] T. Mizrahi, O. Rottenstreich, and Y. Moses, "Timeflip: Using timestamp-based TCAM ranges to accurately schedule network updates," *Trans. Netw.*, vol. 25, no. 2, pp. 849–863, 2017.

[38] T. Mizrahi, E. Saat, and Y. Moses, "Timed consistent network updates in software-defined networks," *Trans. Netw.*, vol. 24, no. 6, pp. 3412–3425, 2016.

[39] J. Zheng, G. Chen, S. Schmid, H. Dai, and J. Wu, "Consistent network updates in timed sdns," *IEEE J. Sel. Areas Commun*, 2017.

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

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

[42] J. Perry, A. Ousterhout, H. Balakrishnan, D. Shah, and H. Fugal, "Fastpass: a centralized "zero-queue" datacenter network," in *SIGCOMM*, 2014.