On the Consistent Migration of Splittable Flows: Latency-Awareness and Complexities

Klaus-Tycho Foerster
klaus-tycho.foerster@univie.ac.at
University of Vienna, Austria

***Abstract*—Network traffic demands change all the time, giving rise to the well-investigated topic of consistent network updates. We in this paper study the consistent migration of flows, in particular the avoidance of transient congestion. Most previous work implemented rather coarse-grained techniques, ignoring the effect of link latency in their computations. Recent work shows that, in order to achieve the goal of zero packet loss, earlier methods need to be adapted to cope with link latency.**

However, the current work only considers unsplittable flows, which are routed along a single path. We present the first study of splittable flows in this context, where a single flow may be spread over the network for better throughput and load balancing.

Interestingly, splittable flows seem to be harder to tackle in this setting from a complexity point of view. Nonetheless, we provide the first methods and insights to integrate splittable flows into consistent update frameworks under the aspect of link latencies.

***Index Terms*—network updates, consistency, congestion, SDN**

I. Introduction

Background. Fast and consistent updates of the data plane are a frequent challenge in centrally-controlled networks [1]. With the rise of Software-Defined Networking, there is a large amount of work that deals with the consistency of such updates in particular [2]. An important consistency property in this context is congestion-freedom, *i.e.,* avoiding packet loss [3].

Motivation. Recent work [4]–[6] has shown that ignoring the impact of link latencies can lead to packet loss when considering consistent flow migration. However, previous work in the area does not contain mathematical errors, but rather a conceptional oversight in their (coarse-grained) model: flows do not just exist in their "old" and "new" states, after respectively before the update, but can also be present on a link at the same time. For example, consider the practical scenario that the old path has high latency and the new path has low latency. Then, for a time equivalent to the latency delta, the flow will congest *itself*, see [6]. As such, previous work is not congestion-free.

The only exceptions we are aware of are [4]–[6], but these works only consider the migration from/to *unsplittable* flows along simple paths. We in this work show how to adapt prior consistent update frameworks to deal with *splittable* flows under link latency, focusing our efforts on the standard dependency graph [3] approach popularized by *Dionysus* [1].

Challenge. *Dionysus* [1] employs a dependency graph approach, where the old (current) and new (desired) network states are analyzed, to see in which orderings the consistent updates can proceed. Dependency graphs do not require the complete update schedule to be computed in advance, rather, they can be used to dynamically update on the fly, depending on how fast the update components progress in different parts of the network. In order to do so, the individual updates need to be verified for consistency, *i.e.,* are link capacities violated?

To the best of our knowledge, no current work can handle such a verification process for splittable flows under the impact of link latencies. After briefly describing our formal model in the next paragraph, we show how our different techniques can be used to adapt dependency graph approaches in general, by providing algorithms to check the updates for consistency.

Model. A network N is a directed graph $G = (V, E)$, $|V| = n$, with link capacities $c(e)$ and a latency function ℓ, assigning non-negative latencies to every link. The link latency $\ell(e)$ describes the time it takes to traverse a link. We focus on splittable flows F, which are routed from a source s to a destination t, where the flow links form a directed acyclic graph, respecting standard flow constraints [7]. For theoretical completeness, we assume that the flow splitting can be performed exactly, and refer to [8] for practical considerations. A *network update* is a tuple (N, F, F'), where F describes the *old* and F' the *new* flow, performed using 2-phase commits [9]:

In the 2-phase commit technique, packets are tagged s.t. they can be identified as belonging to F (old) or F' (new). Packets tagged with F' may only be used when the appropriate rules are installed in the switches. Once no more F-tagged packets exist, the corresponding switch rules can be deleted.

We assume that a node v can change its *own* forwarding rules atomically, but different nodes can only updated asynchronously [2]. We also set buffer sizes to zero, as filling switch buffers negatively impacts the network performance. A network update is called *consistent*, if no link capacities are violated during any time.

Contributions. We present the first study of consistent updates for splittable flows under the effect of link latencies. Previous work only considered 1) splittable flows *without* link latencies or 2) *unsplittable* flows with or without link latencies.

- In §II, we present an algorithm to check if a given flow update is consistent under some given link latencies. However, as a single flow update can lead to exponentially many different link utilizations, such an algorithm can have exponential complexity. We thus provide a faster algorithm with almost linear runtime, which performs a tradeoff between computation efficiency and covered solution space.

978-1-5386-7659-2/18/$31.00 ©2018 IEEE

- In §III, we consider the case of being unaffected by exact latencies, computing solutions that are valid even if the link latencies change during the update. In this setting, we can provide a linear program formulation that makes use of the concept of *preflows* [7]. Our methodology, which temporarily separates a splittable flow into two parts with separate flow rules, is not only computationally efficient, but also provides consistent updates in situations where the standard 2-phase commit techniques are unable to avoid congestion.

II. Checking Consistent Updates Part I: Handling Explicit Latencies

The current state-of-the-art for checking consistent network updates for splittable flows under link latencies are simulations respectively so-called time-extended networks [4], [5]: the effects of a flow network update are considered at discrete points in time, which capture all different states (from the point of link utilization) in the continuous time setting. We provide an adaptation for splittable flows in Algorithm 1, where the network update is consistent if the calculated $\Delta_{\max}$ is zero.

Theorem 1. *The $\Delta_{\max}$ calculated by Algorithm 1 is the minimum time F' needs to be delayed s.t. the applied network update (N, F, F') will become consistent for the latencies ℓ.*

The correctness arguments for Theorem 1 are analogous to the considerations for unsplittable flows [4], [5]: by generating all simple flow paths, we can reduce the splittable to the unsplittable setting, we omit the details due to space constraints.

Exponential Runtime. However, the runtime of Algorithm 1 can be exponential, if the number of different paths is exponential, which is already possible in simple directed acyclic graphs. For example, consider the network at the top of Figure 1: by extending the "ladder"-construction in the middle, the number of simple s-t paths reaches $2^{\Theta(n)}$. Even if these paths were considered from a joint perspective, the number of different link utilizations can also be exponential, as seen in the flow size diagram in Figure 1 for the incoming link e of t.

Improved Runtime. As such, we depart from the simulation approach and aim for polynomial runtimes. To this end, we propose computing a similar output in almost linear time: How much does F' need to be delayed s.t. F will have completely departed from all links e with $F(e) + F'(e) > c(e)$?

Input: Network $N = (G = (V, E), c)$, latency ℓ, splittable flows F, F' inducing subnetworks $N_F = (G_F = (V_F, E_F))$, $N_{F'} = (G_{F'} = (V_{F'}, E_{F'}))$.

1: Recursively generate all simple paths from s to t in N_F and $N_{F'}$, denoted as the sets $\mathcal{P}_F$ and $\mathcal{P}_{F'}$.
2: $\forall$ paths in $\mathcal{P}_F$, for each link e calculate the initial utilization of e and at what time T the subflow of F will no longer utilize e, and save these values attached to e.
3: $\forall$ paths in $\mathcal{P}_{F'}$, for each link e calculate at what time T the subflow of F' will arrive and with which utilization, save these values attached to e.
4: $\forall$ links $e \in E$, calculate the minimum time Δ_e that F' needs to be delayed s.t. the capacity of e is not violated.
5: Set $\Delta_{\max} := \max_{e \in E} \Delta_e$

Output: $\Delta_{\max}$

Algorithm 1: The algorithm computes for splittable flows F, F' how much ($\Delta_{\max}$) the new flow F' has to be delayed additionally s.t. F' will not utilize an link that F still uses which has not enough capacity to support both F, F'.

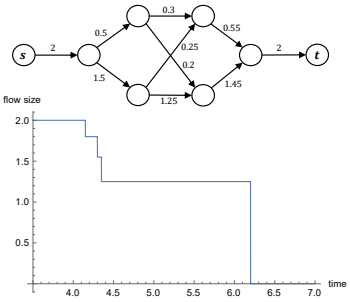

Fig. 1. In this network, all links have a latency and identical capacity as depicted next to them (the values for latency and capacity could also differ). We assume that the flow F uses all links of the network at full capacity, and is switched off at time $T = 0$. The flow is not drawn here to not clutter the picture. The diagram below depicts the utilization of the incoming link e of t. At time $T = 4.15$ the flow utilization on e drops to 1.8, then at $T = 4.3$ to 1.55, then at $T = 4.35$ to 1.25, and lastly at $T = 6.2$ to 0. By extending the network, the amount of different flow utilizations can grow exponentially.

This question is equivalent to calculating the path with the highest latency (that F uses) to each such link from s and then comparing it with the lowest latency path from F'.[1]

Still, finding a longest path in a general undirected graph is an *NP*-complete problem [11], but each single network flow is directed and acyclic. As such, we can use known algorithms to first calculate the longest paths with, *e.g.,* a modified topological sorting of the subgraph used by F in a runtime of $O(n+m)$ [7].

We can then use topological sorting again (with logarithmic overhead) to find the lengths of the shortest paths from s to all other nodes in the subgraph induced by F'. We can then check all links $e = (u, v)$ with $F(e) + F'(e) > c(e)$ and calculate the difference between the complete departure of F at u and the first arrival of F' at u. The maximum such value is the $\Delta_{\max}$ that F' has to be delayed. We cast our insight into Theorem 2, omitting the pseudo-code due to space constraints.

Theorem 2. *The minimal time Δ_{max} that F' has to be delayed s.t. for every link e with $F(e) + F'(e) > c(e)$ holds, that F, F' do not utilize e at the same time, can be computed in a runtime of $O(n \log n + m)$ using topological sorting.*

Next Steps. We provided two algorithms with a tradeoff between covered solution space and maximum runtime. In the next section, we show how to obtain a polynomial runtime, by considering all possible link latencies: as thus, we have the benefit that updates stay consistent even if latencies change.

III. Checking Consistent Updates Part II: Consistency under all possible Latencies

Before delving directly into consistent updates unaffected by latencies, let us take a step back and consider Figure 2:

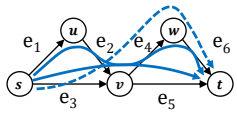

Fig. 2. In this network, all links have a capacity of two and a latency of one. The old flow F is depicted with solid blue lines: Starting from s, it diverges to u and v, but the flow via u is merged at v again. Along each link, the flow F has a size of one. The new flow F', of size two, is depicted as dashed.

[1] An extreme assumption would be to enforce capacity for both F, F' [10].

Congestion occurs. When a network update from the flow F to F' is being applied at time zero in Fig. 2, the following will occur: From $T = 0$ on, no new packets from F are pushed onto e_1, e_3, while the link e_3 is already utilized by the new flow F'. From $T = 1$, F will no longer send packets on e_2.

Furthermore, the links e_4, e_5 now are only utilized by the part of the flow F which was routed along the top path via e_1, e_2. This part of F is split equally along e_4 and e_5. The flow F' does now utilize e_4 as well: This leads to a violation of capacity on e_4, as depicted in Figure 4, since a flow of size 2.5 needs to be routed on e_4, but e_4 has only a capacity of 2.

Splitting a splittable flow. Is there a way to implement the network update (N, F, F') without violating link capacities? Consider the splitting of F into two subflows F_1, F_2 in Figure 3:

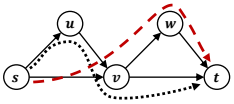

Fig. 3. Splitting F into two subflows of size one: the preflow is marked in red, the remaining flow of F in black. Even though the amount of flow from s to t is still two, we can now update consistently to the dashed flow F' from Figure 2 by "adding" the flow difference between the preflow and the new flow F', removing the remaining part of F, and later merging F' again.

Instead of splitting the incoming flow of v equally along the links e_4, e_5, the flow via e_3 is now routed along e_4 (called F_1) and the flow via e_1, e_2 is routed along e_5 (called F_2). We will denote the flow composed of F_1, F_2 by (F_1, F_2). The flow (F_1, F_2) has the same utilization on every link as F, they just differ in the forwarding rules at the nodes. Observe that F' can be split equally into two subflows F_1', F_2' of size 1, resulting in the flow (F_1', F_2'). Apart from the injection times, F_1' and F_1 are identical. Thus, a network update (N, F_1, F_1') would not change the utilization of any link at any time. Therefore, if F_2' can be added to (F_1, F_2) without violating capacity constraints, then the network update $(N, (F_1, F_2), (F_1', F_2'))$ will not cause any link capacity to be violated. As can be seen in Figure 3, adding F_2' does not violate any capacity constraints.

Hence, it is possible to implement the network update (N, F, F') without violating link capacities by changing F into (F_1, F_2), then applying the network update $(N, (F_1, F_2), (F_1', F_2'))$, and lastly changing (F_1', F_2') into F'.

Note that the first and the last step only involve changing forwarding rules at the nodes without changing any link utilization. Furthermore, while in this simple case (F_1', F_2') and F' are essentially the same flow, this is not the case in more complex examples. Thus, distinguishing between these two flows is necessary in general. The above considerations regarding the implementation of the network update (N, F, F') are completely independent of the specific link latencies in the network. As such, the implementation above is consistent. We generalize and formalize this concept of a multi-stage network update with shared subflows in the following under the notion of *preflows*. We then show that preflows and consistent network updates consistent for all latencies are inherently related.

Preflows In the example described above, we were able to find a common subflow F_1 of both F and F' s.t. $(F(e) - F_1(e)) + (F'(e) - F_1(e)) + F_1(e) = F(e) + F'(e) - F_1(e) \leq c(e)$ for

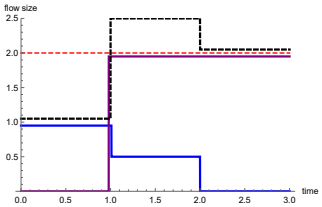

Fig. 4. In this plot, the red dashed line depicts the capacity of the link e_4. However, we are going to assume for now that the capacity of e_4 is unbounded. Let the network update from F to F' be performed at time $T = 0$. Then, the utilization of e_4 by F' is depicted in purple, while the the utilization of e_4 by F is depicted in blue. Together, their utilization is depicted by the black dashed line. Thus, from time $T = 1$ to time $T = 2$, we have congestion on e_4, as $c(e_4) = 2$ (red dashed), but flows of size 2.5 (black dashed) are routed.

all links e in the network. As it turns out, by relaxing the flow conservation condition of the subflow F_1, we can extend our new concept even further. The intuition is as follows: We would like to satisfy $F(e) + F'(e) - F_1(e) \leq c(e)$ for all links e, but F_1 does not necessarily have to be a flow. As long as we can find a common subpart of F and F' that can still provide correct forwarding rules, we can consider this part F_1 as stationary during the network update. As such, the incoming flow of F_1 at every node v has to be at least as large as the outgoing flow of F_1 at v, leading to the following definition:

Definition 1. *Let F, F' be flows from source s to destination t. We define an (F, F')-preflow as a map $\phi^{F,F'} : E \to \mathbb{R}_{\geq 0}$ s.t.*

$$\forall e \in E : \phi^{F,F'}(e) \leq F(e) \quad \text{and} \quad \phi^{F,F'}(e) \leq F'(e), \tag{1}$$

$$\forall v \in V \setminus \{s\} : \sum_{e \in out(v)} \phi^{F,F'}(e) \leq \sum_{e \in in(v)} \phi^{F,F'}(e). \tag{2}$$

Note that in general, a preflow $\phi^{F,F'}$ is not unique. As a preflow $\phi^{F,F'}$ intuitively represents a stationary part of a network update (N, F, F'), and as such, is not subject to the differences in link latencies, it is advantageous to pick a preflow as large as possible in order to mitigate congestion.

However, we have not yet defined how to actually perform a network update with preflows; especially the intricate part on how to define forwarding rules for the flows and their preflow.

Definition 2. *Let N be a network, let F, F' be flows in N from s to t, and let $\phi^{F,F'}$ be an (F, F')-preflow. Analogously to the decomposition of a flow into two subflows as performed above, we can decompose F into its subpreflow $\phi^{F,F'}$ and the remaining "postflow" $F - \phi^{F,F'}$. Imagine changing the forwarding rules of F s.t. they coincide with the forwarding rules of $\phi^{F,F'}$ when F is restricted to its subpreflow $\phi^{F,F'}$. We denote the resulting flow by $\left(\phi^{F,F'}, F - \phi^{F,F'}\right)$.*

Observe that changing the forwarding rules in a given flow, without changing the utilization of any link in the network, does not violate any link capacity, independent of the actual latencies. In particular, the following observation holds:

Observation 1. *Let N be a network and let F, F' be flows in N from s to t. Furthermore, let $\phi^{F,F'}$ be an (F, F')-preflow. Then the two network updates $\left(N, F, \left(\phi^{F,F'}, F - \phi^{F,F'}\right)\right)$ and $\left(N, \left(\phi^{F,F'}, F' - \phi^{F,F'}\right), F'\right)$ are consistent.*

To complete the transition from F to F', we need a consistent update from $\left(\phi^{F,F'}, F - \phi^{F,F'}\right)$ to $\left(\phi^{F,F'}, F' - \phi^{F,F'}\right)$:

Theorem 3. *Let F, F' be flows in N from s to t, and let $\phi^{F,F'}$ be an (F, F')-preflow. If the condition $F(e) + F'(e) - \phi^{F,F'}(e) \leq c(e)$ holds for all $e \in E$, then the network update $\left(N, \left(\phi^{F,F'}, F - \phi^{F,F'}\right), \left(\phi^{F,F'}, F - \phi^{F,F'}\right)\right)$ is consistent.*

Proof Sketch. Observe that, due to the forwarding rules of the considered flows, the part of F corresponding to $\phi^{F,F'}$ and the part of F' corresponding to $\phi^{F,F'}$ are never on the same link at the same point in time. Thus, at each point in time the utilization of every link e satisfies $F(e) - \phi^{F,F'}(e) + F'(e) - \phi^{F,F'}(e) + \phi^{F,F'}(e) = F(e) + F'(e) - \phi^{F,F'}(e) \leq c(e)$. □

Hence, if a preflow as required by Theorem 3 exists, we obtain a sequence of consistent updates unaffected by latencies from F to F' by combining Observation 1 and Theorem 3.

Efficiency. Similar to finding a maximum flow, the problem of finding such a feasible preflow can be solved efficiently in polynomial time [7] by a linear program as given in Figure 5.

Find $(x_e)_{e \in E}$, s.t.
1) $\forall v \in V \setminus \{s\} : \sum_{e \in out(v)} x_e \leq \sum_{e \in in(v)} x_e$
2) $\forall e \in E : x_e \leq F(e)$ 3) $\forall e \in E : x_e \leq F'(e)$
4) $\forall e \in E : x_e \geq 0$ 5) $\forall e \in E : x_e \geq F(e) + F'(e) - c(e)$

Fig. 5. Linear program to find a preflow $\phi^{F,F'}$ for the flows F, F' that satisfies the requirements of Theorem 3.

IV. Related Work

Reitblatt *et al.* [9] provided one of the standard methods for consistent flow updates with their 2-phase commit protocol. Shortly after a first chart of the complexity landscape of consistent updates was given by Mahajan and Wattenhofer [3], [12], who also proposed the concept of dependency graphs, applied to flows by Jin *et al.* [1]. Even without link latencies, the consistent migration of unsplittable flows is *NP*-hard, but for splittable flows it is in P [13], with Zheng *et al.* [14] providing approximation algorithms for NP-hard cases. We further note that we assumed that the new flow paths are fixed, but they can also be seen as part of the output [15], [16].

Inspired by the concept of timed updates [17], Zheng *et al.* present algorithms for consistent flow migration under link latencies [4], [5], albeit restricted to unsplittable flows—for which the concept of considering all link latencies was recently proposed [6]. Further studies of flow consistency in the so-called node-ordering model (without 2-phase commit) have also been performed by Amiri *et al.* [18], [19], considering unsplittable flows without link latencies. Even though unsplittable flows are well-studied, many open questions remain [20].

Lastly, we note that there is also a large set of works considering further consistency properties, such as waypoint or policy enforcement [21]. We refer to [2] for a recent survey.

V. Conclusion

In this paper we studied consistent updates for splittable flows, where the goal is to respect link capacities at all times. We provided the first such study for splittable flows under the impact of link latencies, where prior work only considered 1) splittable flows in a coarse-grained approach, ignoring link latencies, leading to packet loss, or 2) *unsplittable* flows.

We provided two algorithms for given link latencies, providing a tradeoff between computation speed and solution space.

Additionally, we explored the concept of considering all latencies for splittable flows, *i.e.,* the updates have to be consistent for any change in the network link latencies. In this setting, we provided a new toolkit to check for consistent updates by applying concepts from so-called preflows, where appropriate preflows can be computed in polynomial time.

As next steps beyond this short paper, we plan to extend our work with simulations and technical implementations.

Acknowledgments. The author gratefully acknowledges Sebastian Brandt for his assistance and comments on an earlier draft, in particular regarding preflows. The author would also like to thank the anonymous reviewers for their suggestions which helped clarify parts of this paper.

References

[1] X. Jin, H. H. Liu, R. Gandhi, S. Kandula, R. Mahajan, M. Zhang, J. Rexford, and R. Wattenhofer, "Dynamic scheduling of network updates," in *Proc. ACM SIGCOMM*, 2014.
[2] K.-T. Foerster, S. Schmid, and S. Vissicchio, "Survey of consistent network updates," *CoRR*, vol. 1609.02305v2, 2018.
[3] R. Mahajan and R. Wattenhofer, "On consistent updates in software defined networks," in *Proc. ACM HotNets*, 2013.
[4] J. Zheng, G. Chen, S. Schmid, H. Dai, J. Wu, and Q. Ni, "Scheduling congestion- and loop-free network update in timed sdns," *IEEE JSAC*, vol. 35, no. 11, pp. 2542–2552, 2017.
[5] J. Zheng, B. Li, C. Tian, K.-T. Foerster, S. Schmid, G. Chen, and J. Wu, "Scheduling congestion-free updates of multiple flows with chronicle in timed sdns," in *Proc. IEEE ICDCS*, 2018.
[6] K.-T. Foerster, "Don't disturb my flows: Algorithms for consistent network updates in software defined networks," PhD. thesis, ETH Zurich, Switzerland, September 2016.
[7] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms (3. ed.)*. MIT Press, 2009.
[8] N. Kang, M. Ghobadi, J. Reumann, A. Shraer, and J. Rexford, "Efficient traffic splitting on commodity switches," in *Proc. ACM CoNEXT*, 2015.
[9] M. Reitblatt, N. Foster, J. Rexford, C. Schlesinger, and D. Walker, "Abstractions for network update," in *Proc. ACM SIGCOMM*, 2012.
[10] K.-T. Foerster and R. Wattenhofer, "The power of two in consistent network updates: Hard loop freedom, easy flow migration," in *Proc. IEEE ICCCN*, 2016.
[11] M. R. Garey and D. S. Johnson, *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., 1990.
[12] K.-T. Foerster, R. Mahajan, and R. Wattenhofer, "Consistent updates in software defined networks: On dependencies, loop freedom, and blackholes," in *Proc. IFIP NETWORKING*, 2016.
[13] S. Brandt, K.-T. Foerster, and R. Wattenhofer, "On consistent migration of flows in sdns," in *Proc. IEEE INFOCOM*, 2016.
[14] J. Zheng, H. Xu, G. Chen, and H. Dai, "Minimizing transient congestion during network update in data centers," in *Proc. IEEE ICNP*, 2015.
[15] 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.
[16] R. Gandhi, O. Rottenstreich, and X. Jin, "Catalyst: Unlocking the power of choice to speed up network updates," in *Proc. ACM CoNEXT*, 2017.
[17] T. Mizrahi, E. Saat, and Y. Moses, "Timed consistent network updates," in *Proc. ACM SOSR*, 2015.
[18] S. A. Amiri, S. Dudycz, S. Schmid, and S. Wiederrecht, "Congestion-free rerouting of flows on dags," in *Proc. ICALP*, 2018.
[19] S. A. Amiri, S. Dudycz, M. Parham, S. Schmid, and S. Wiederrecht, "Short schedules for fast flow rerouting," *CoRR*, vol. 1805.06315, 2018.
[20] K.-T. Foerster, "On the consistent migration of unsplittable flows: Upper and lower complexity bounds," in *Proc. IEEE NCA*, 2017.
[21] A. Ludwig, S. Dudycz, M. Rost, and S. Schmid, "Transiently policy-compliant network updates," *IEEE/ACM Trans. Netw.*, 2018.