Graceful Router Updates in Link-State Protocols

Francois Clad[1], Pascal Mérindol[1], Stefano Vissicchio[2], Jean-Jacques Pansiot[1] and Pierre Francois[3]

[1]Université de Strasbourg
{fclad,merindol,pansiot}@unistra.fr

[2]Université catholique de Louvain
stefano.vissicchio@uclouvain.be

[3]Institute IMDEA Networks
pierre.francois@imdea.org

***Abstract*—Manageability and evolvability are crucial needs for IP networks. Unfortunately, planned topological changes may lead to transient forwarding loops in link-state routing protocols commonly used in IP networks. These lead to service unavailability, reducing the frequency at which operators can adapt the network topology.**

Prior works proved that the state of a given link can be modified while avoiding forwarding inconsistencies without changing protocol specifications. In this paper, we study the more general problem of gracefully modifying the state of an entire router, while minimizing the induced operational impact. As opposed to a single-link modification, the router update problem is k-dimensional for a node of degree k. Moreover, we show that the interplay between operations applied at the router granularity can lead to loops that do not occur considering a single-link modification. In this paper, we present an efficient algorithm that computes minimal sequences of weights to be configured on the links of the updated node. Based on real IP network topologies, we show that the size of such sequence is limited in practice.

I. Introduction

IP networks are evolving entities that often undergo topology modification, e.g., to support hardware, software and configuration upgrades [1], [2]. Often, single links or single devices are involved in each of those operations [3]. Link-state routing protocols such as OSPF or IS-IS are used by the vast majority of intra-domain IP networks. Unfortunately, they are prone to convergence loops in case of topological changes [4]–[6]. Operators currently lack simple mechanisms allowing to gracefully adapt a network topology while avoiding transient forwarding loops. On the one hand, recently proposed techniques for lossless reconfigurations do not directly apply to router changes [7] or introduce large overhead [8], [9], e.g., requiring two IGPs to be simultaneously run network-wide. On the other hand, protocol specific features, like the overload-bit in IS-IS, do not guarantee absence of transient forwarding loops during IGP convergence. Unfortunately, the possibility that planned operations may lead to packet losses can affect the accommodation of strict Service Level Agreements (SLAs), hence reducing the ability of operators to perform frequent maintenance operations [10]. Moreover, the scarcity of maintenance windows makes it difficult to promptly react to sudden events, like the saturation of a network component due to changing traffic conditions. It also limits the ability to apply software and hardware upgrades at arbitrary times (considering that some of those updates are meant to introduce security patches, they would ideally need to be applied as soon as possible).

978-1-4799-1270-4/13/$31.00

In this paper, we investigate the possibility for link-state IGPs to support graceful router update operations, without changing protocol specifications. We show that the interplay between operations applied at the router granularity can lead to loops that do not occur in the single-link modification problem. Nevertheless, we prove that forwarding loops possibly occurring during protocol convergence can be provably avoided by progressively modifying the weights of the links incident on the router to be modified. Such a progressive modification of weights can be imposed by propagating a conveniently crafted sequence of Link State Advertisement (LSA) packets from the router to be updated. In order to ensure a minimal operational impact, we aim at computing minimal sequences of weight increments. This need prevents us from re-using known techniques [7] to de-tour traffic on a per-link basis. Thus, we present a new and efficient algorithm, which we called GBA, to compute minimal sequences of weight increments (hence, intermediate LSAs packets to be propagated in order to avoid any transient loops). Our algorithm also guarantees the absence of transient loops in IGP networks relying on LDP. The proposal described in this paper is hence valid even when the IGP is used in conjunction with BGP, as long as Ingress-Egress encapsulation is deployed in the network [11]. Such deployments are very common in Service Provider networks.

The remainder of the paper is organized as follows. In Section II, we motivate our approach by showing the inefficiency of solutions defined for single link modifications. In Section III, we propose a formal theoretical framework for the router update problem. In particular, we introduce the notion of loop-constraints, which define sufficient and necessary intervals of metrics that need to be applied to avoid transient forwarding loops. In Section IV, we describe the GBA algorithm that relies on a greedy backward search approach to optimally capture all loop constraints. Also, we formally prove correctness and efficiency of GBA. In Section V we report the performance of GBA on several real network topologies. We show that most sequences are short, typically including less than five intermediate vectors of increment in most of the cases. Finally, we conclude and discuss future work in Section VI.

II. The Node Shutdown Problem

In link-state routing protocols, forwarding information about the network topology are propagated to all the routers through Link-State Advertisement (LSAs). On the basis of this in-

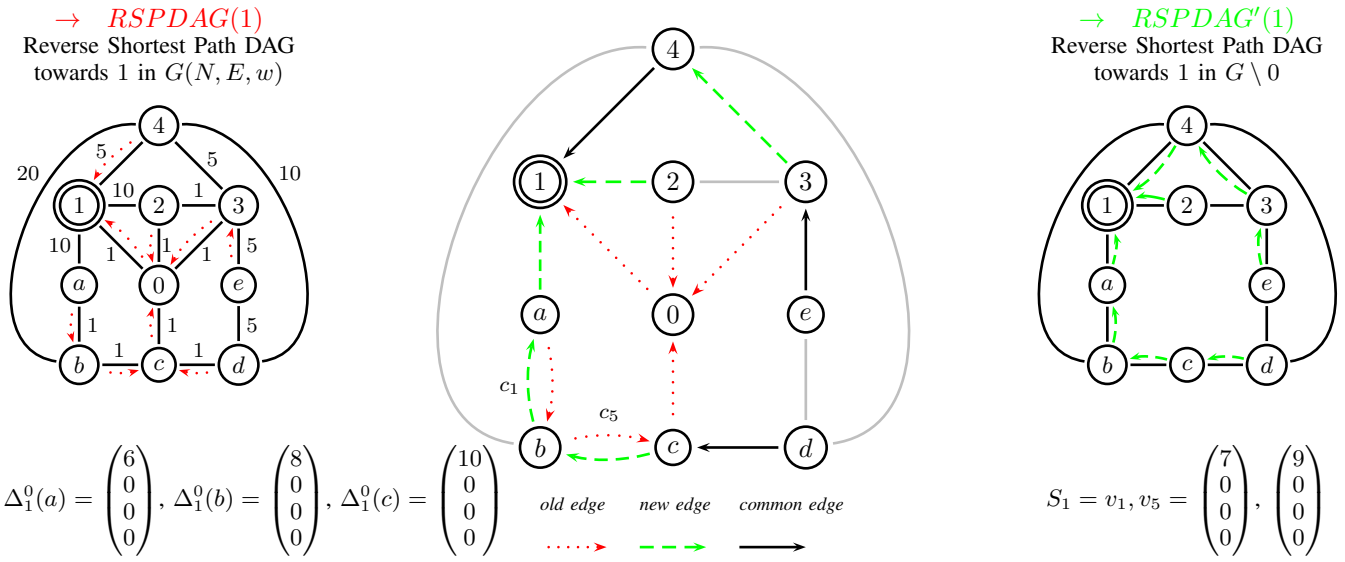


Fig. 1: Compute a loopfree sequence for a single destination (towards 1 on this gadget).

Destination 2	$c_2 = \{c, d, c\}$	$S_2 = v_2 = \begin{pmatrix} 0 & 9 & 8 & 0 \end{pmatrix}$
Destination 3	$c_3 = \{c, d, c\}$	$S_3 = v_3 = \begin{pmatrix} 0 & 7 & 8 & 0 \end{pmatrix}$
Destination 4	$c_4 = \{c, d, c\}$	$S_4 = v_4 = \begin{pmatrix} 3 & 2 & 3 & 0 \end{pmatrix}$
Uniform		
$c_4 \to c_1 \to c_3 \to c_2, c_5$		$S_{UFR} = \begin{pmatrix} 3 \\ 3 \\ 3 \\ 3 \end{pmatrix}, \begin{pmatrix} 7 \\ 7 \\ 7 \\ 7 \end{pmatrix}, \begin{pmatrix} 8 \\ 8 \\ 8 \\ 8 \end{pmatrix}, \begin{pmatrix} 9 \\ 9 \\ 9 \\ 9 \end{pmatrix}$
Forward Greedy		
$c_2, c_3, c_4 \to c_1 \to c_5$		$S_{GFA} = \begin{pmatrix} 3 \\ 9 \\ 8 \\ 0 \end{pmatrix}, \begin{pmatrix} 7 \\ 9 \\ 8 \\ 0 \end{pmatrix}, \begin{pmatrix} 9 \\ 9 \\ 8 \\ 0 \end{pmatrix}$
Backward Greedy		
$c_1, c_4 \to c_2, c_3, c_5$		$S_{GBA} = \begin{pmatrix} 7 \\ 2 \\ 3 \\ 0 \end{pmatrix}, \begin{pmatrix} 9 \\ 9 \\ 8 \\ 0 \end{pmatrix}$

TABLE I: Multiple destinations: towards a minimal safe convergence

formation, each router locally computes its shortest paths [1] to each destination. A known consequence of such a local computation of the shortest path is that link-state routing IGPs are prone to convergence loops. In particular, since IGPs do not impose any constraint on routing update ordering, the routing information is flooded asynchronously. This causes temporarily inconsistent state at different routers, possibly causing forwarding loops during IGP convergence.

A. Routing Convergence Leads to Transient Forwarding Loops

As an illustration, consider Fig. 1 that shows a router removal example. The topology of a network is depicted as a graph, in which nodes represent routers, and links represent the direct connections (or IGP adjacencies) used by routers to exchange routing information. Graphs are weighted according to the cost set on each IGP adjacency. We thus model the IGP network as a logical graph $G = (N, E, w)$, where N is the set of routers, E is the set of IGP links (or adjacencies) between the routers, and w is a function associating a weight to each link. Let 0 be the router to be removed from G. Colored arrows represent intended packet forwarding paths from each router to destination 1. In this paper, we refer to the set of forwarding paths towards a given destination d as *Reverse Shortest Path DAG* and denote it as $RSPDAG(d)$

The graphs in the leftmost and rightmost sub-figures represent the configuration of the network and the RSPDAGs before and after the removal of 0, respectively. The central sub-figure illustrates how inconsistent routing information owned by different routers may cause transient forwarding loops during the convergence. In particular, if router c updates its FIB before b does, then c starts forwarding traffic to destination 1 according to the new topology, while b keeps forwarding traffic as in the initial topology. This creates a forwarding loop between c and b. A similar loop is present between a and b, and such cycles may occur for all destinations in the network. Note that the possibility for transient loops to occur depends on the message propagation delays and the RIB-FIB updating time of each router. Of course, those loops are eventually solved as soon as the routers involved in the loops are aligned with respect to the topological information. Nevertheless, convergence loops can cause transient packet losses towards multiple destinations during the convergence process. Those problems can be exacerbated if some routing updates are lost or delayed.

In this paper, we study how to prevent the transient forwarding loops caused by the planned addition or removal of a single router. Note that while we focus on the removal of a node, symmetric considerations apply for the case of node addition. Intuitively, in order to prevent convergence loops, we aim at forcing an ordering in which routers change their forwarding paths, so that no conflicting choices occur. We tar-

[1] ECMP enables the use of several Equal Cost Multi-Path and thus induces routing DAG - for Directed Acyclic Graph - instead of routing tree. In this paper, we take equal cost multiple paths into account.

get to a solution that can be applied without changing current protocol design, mechanisms, and message format. We only admit to slightly change the implementation of the protocols, namely, on how IGP topology modification commands are implemented. The underlying goal is to make our solution incrementally deployable, and implementable with a minor router operating system upgrade.

We realize this objective by mandating the router to be removed to progressively change the weight of its outgoing links, i.e., by propagating a sequence of conveniently crafted LSAs. Indeed, an LSA can contain local multiple topological information. For example, a router LSA of type 1 in OSPF carries information on the set of outgoing interfaces of a given node and their respective IGP weights. Hence, the information in a single LSA influences forwarding decisions of all the routers in the network, since each router computes its forwarding paths on the basis of the LSAs it receives. In the following, we refer to this progressive change of link weights as weight change or metric increment sequence.

Observe that each LSA causes IGP routers to exchange the new topological information and converge to a new routing state, leading to control plane overhead. Thus, a higher number of LSAs translates to additional IGP convergence time and more stress on the control planes of all routers (IGP churn). As the convergence time and control-plane overhead are major issues in link-state IGP networks, one of our primary objectives is to *minimize the number of LSAs being generated to prevent convergence loops*. Assuming that convergence for each LSA does not exceed one second, we think that a realistic upper bound for the number of LSAs to be generated can be fixed to a value of 5. An opportunity to minimize the number of LSAs comes from the possibility to include weight changes for different links in the same LSA. However, finding the minimal number of LSAs that provably prevent all the possible transient loops leads to an algorithmic problem that is not straightforward to solve.

Consider again the example in Fig. 1, and focus on the convergence loop between a and b. Depending on the IGP message timing, the loop can occur because a sends to b its traffic towards destination 1 before the removal of router 0, while the opposite holds after the removal. The loop is provably avoided if and only if a starts forwarding packets directly to 1 before b changes its forwarding decision. This occurs, for example, if a weight of 8 is configured on the link $(0, 1)$. Indeed, changing the weight of link $(0, 1)$ to 8 before removing router 0 provably prevents the forwarding loop to occur. The same does not hold if the weight of link $(0, 1)$ is set to 7 or to 9. In the former case, the weight increment is not enough to make a dismiss its initial path (a temporarily uses ECMP), while in the latter case both a and b switch forwarding path and the convergence loop can occur. In other words, there is a strict constraint on the weight to be configured on link $(0, 1)$ in order to avoid convergence loops. On the other hand, the paths traversing links $(0, 2)$ and $(0, 3)$ are already longer than path $(a\ 1)$, which is the final one. Hence, the weight of links $(0, 2)$ and $(0, 3)$ can be kept the same or arbitrarily increased with no effect on the convergence loop between a and b. Intuitively, constraints for different possible transient loops (possibly, to different destinations) have to be combined together. However, strict constraints may be required on several link increments, thus limiting the number of transient loops being prevented with a single metric increment.

B. Single Link-based Graceful Updates Are not Efficient

In [7], it has been shown that convergence loops can be provably avoided for single link weight changes. In principle, the same technique could be directly applied to avoid convergence loops for the router shutdown problem. Indeed, a simple way to de-tour traffic from the router to be shutdown is to make it unattractive through all its links, one at a time. Hence, the technique proposed in [7] can be sequentially applied, in order to prevent all IGP convergence loops. In the following, we refer to such a technique as *link-by-link*. Unfortunately, this technique would not minimize the number of LSAs to be generated, since it overlooks the possibility to change the weight of several links with the same LSA. In many cases, two or three link weight changes are needed to gracefully re-route the traffic from a single link [7]. Assuming that a router has three outgoing links, this would lead to generate 6 to 9 LSAs, which significantly deteriorates the convergence time.

A variant of the link-by-link technique is what we call the *uniform* increase algorithm. This algorithm consists in applying the technique proposed in [7] to a modified model of the IGP topology. In this modified model, the node to be removed is replaced by two virtual nodes with a unique virtual link connecting them. In this new model, the node shutdown problem translates to de-touring the traffic off the virtual link. As a result, the original router shutdown is solved by generating LSAs that contain the same weight increase for all the links adjacent to the router to be shut down. While the uniform algorithm allows for concurrent weight modification on multiple links, it limits the weight increases that can be applied. Hence, it does not minimize the number of generated LSAs. An example in which the sequence computed by the uniform algorithm is provided in Fig. 1 and Table I where all the destinations are considered at the same time. In particular, the table shows two sequences, namely S_{GFA} and S_{GBA} , still prevents all the transient loops, while being shorter than the one computed by the uniform algorithm, namely S_{UFR}.

While those techniques do not guarantee the minimality of the computed metric increment sequence, they prove that a progressive weight sequence that avoids transient loops always exist. In the rest of the paper, we present a new node-specific algorithm minimizing the number of LSAs to flood, and we evaluate the gain in term of link weight size with respect to link-by-link and uniform techniques.

C. Flapping Side Effect Potentially Leads to More Loops

While enabling the minimization of the sequence, the opportunity of simultaneously applying non-uniform weight

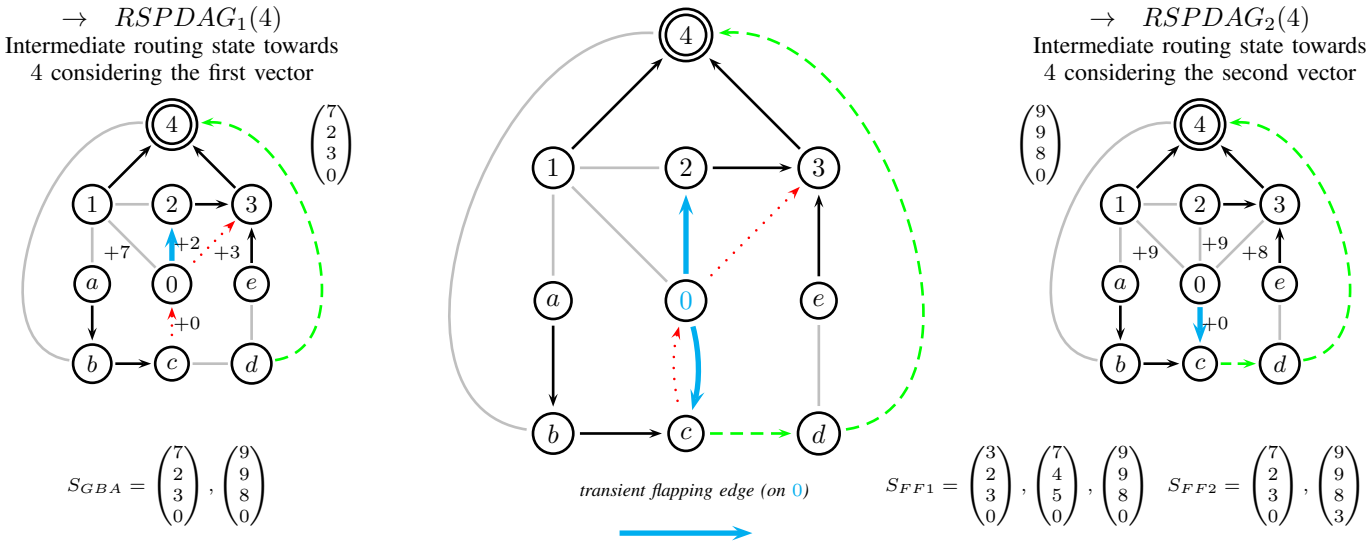


Fig. 2: Illustration of a transient loop induced by route flapping: a floop between nodes 0 and c

increments to different links may come at the cost of introducing traffic flapping, i.e., multiple routing changes on the node being updated. Such a phenomenon potentially leads to additional transient loops that we call *floop*. Floops are due to transiently used shortest paths that are not captured neither in the initial RSPDAG $RSPDAG(d)$ nor in the final RSPDAG $RSPDAG'(d)$ and form cycles implying node 0. Note that floops cannot appear when a single link has to be re-weighted, i.e., they are specific to the graceful router update problem.

An illustration of floop for destination 4 is shown in Fig. 2. In this example, the optimal sequence computed by considering only the initial and final RSPDAGs, i.e., S_{GBA} in the figure, includes two metric increments. Each of those two increments can be achieved with a single LSA originated by 0. Such a transition is safe for all the routers in the network, but for 0 itself. Indeed, a floop could occur when applying the second set of link weight increments, as shown in the central sub-figure in Fig. 2. At that step, the new link weights $(9, 9, 8)$ (assigned to $(0, 1)$, $(0, 2)$ and $(0, 3)$, respectively) induce 0 to use c as next-hop for destination 4, while the opposite holds before applying this metric increment. This can cause a floop between 0 and c.

We decided to address the constraints of floop avoidance separately from usual loop constraints, as floops have different properties than usual ones. Both an algorithmic and a technological solutions to avoid floops are given in Section IV. In Section V, we show that additional increment vectors to handle these loops are seldom required.

III. Problem Formalization

In this section, we define the theoretical framework of our solution. We first introduce the notion of vectorial increments, representing a set of simultaneous weight increments on several outgoing links of the node being modified. Then, we explain how to model potential forwarding loops as a set of vectorial constraints, which represents necessary intervals of metric configurations allowing to compute a *safe* weight sequence, preventing loops.

A. Towards Multi-Dimensional Increments

We now present a formalization of our problem, which generalizes the main concepts previously illustrated. Our formalization is based on the usage of positive vectors. We denote the i-th component of any vector v as $v[i]$. The *size* of a vector v, $|v|$ is the number of components of v. Vectors of the same size can be compared, and a partial order relationship exists between them. In particular, we say that two vectors v_1 and v_2 of size $k \geq 0$ are equal, i.e., $v_1 = v_2$, if $\forall i \in 1 \,..\, k, v_1[i] = v_2[i]$. Greater ($>$), greater or equal ($\geq$), smaller ($<$), and smaller or equal ($\leq$) relationships between vectors are similarly defined, and apply on vectors if they hold on all the corresponding components. In addition, given two vectors v_1 and v_2 (such that $|v_1| = |v_2| = k$), we say that v_1 is *positively greater* than v_2 if $\forall i \in 1 \,..\, k$ either $v_1[i] > v_2[i]$ or $v_1[i] = v_2[i] = 0$. If v_1 is positively greater than v_2, then v_2 is *positively smaller* than v_1, and we use notation $v_1 >^+ v_2$. This relation is introduced to model the absence of constraints on some alternative links while it imposes to verify strict constraints on others.

Given any router $x \neq 0$, we define $\Delta_d^0(x)$ as the vector of the minimum increments such that router x uses at least a new path not including 0 to reach d. For the sake of simplicity, we reduce the notation whenever the destination or the node to be removed are clear from the context. $\Delta(x)$ are vectors in which each component represents the increment necessary to start to detour x's traffic from each of the links outgoing from the router 0 to be shut down. The values of $\Delta(x)$ can be derived by shortest paths topological properties on graphs G and $G' = G \setminus 0$. Let l_i be the i-th link outgoing from router 0, let $C'(x, d)$ be the cost of the shortest path from x to a destination d after the router removal (G'), and let $C(x, l_i, d)$ be the cost of the shortest path from x to a destination d

traversing l_i before the router 0 removal (G). Then, we have

$$\Delta_d^0(x)[i] = C'(x,d) - C(x,l_i,d)$$

As $\Delta(x)$ represents distance amounts to be added to a set of link weights, we also impose that each of its components to be a positive value, i.e., if $C'(x,d) - C(x,l_i,d) < 0$, then $\Delta(x)[i] = 0$. Thus, if $\Delta(x)[i] = 0$, it implies the absence of constraints on the weight of the i-th link.

By definition of $\Delta(x)$, a router x switches to the final forwarding path if and only if each of the links outgoing from 0 is configured with a weight which is strictly greater than the corresponding component of $\Delta(x)$ (except for null values by definition of the positively greater relation $>^+$). Otherwise, the initial path is still used by x. Indeed, if the weight increment of a link l_i is equal to $\Delta(x)[i]$, then the path traversing l_i may be still used as well in an ECMP context. Moreover, note that by definition of $C(x,l_i,d)$ a total ordering applies among Δ_d (for a given destination d), while it is not the case among Δ vectors for several destinations (i.e., only a partial ordering applies at the global view).

Let us illustrate the use of Δ on Fig. 1: different values of such vectors are reported on the left side, in the bottom part of the figure. As an example, consider $\Delta(a)$: $\Delta(a)[1] = 10-4 = 6$ meaning that adding $6+1$ to the weight of link $(0,1)$ makes any path from a to 1 traversing link $(0,1)$ strictly longer than the final one. Hence, setting 7 on link $(0,1)$ ensures that a does not use link $(0,1)$ to reach 1. Moreover, $\Delta(a)[2] = \Delta(a)[3] = \Delta(a)[4] = 0$ because a switches to its final forwarding path even if the links $(0,2)$, $(0,3)$ and $(0,c)$ remain unchanged.

B. Modeling Loops as Vectorial Constraints

Δ vectors allow us to model constraints on the weights to be used in order to avoid convergence loops. In the following, we refer to any change in the weight of the links or in the topology of the IGP graph as *topological change*. Let c be the constraint modeling a convergence loop L that can occur during a topological change, and let 0 the router to be shut down. Then, we define the *loop-constraint*, or simply *constraint*, associated to L as follows.

$$c := (\underline{c} := \min_{\forall x \in L}(\Delta(x)), \bar{c} := \max_{\forall x \in L}(\Delta(x)))$$

For the sake of simplicity, we shorten the notation to $c := (\underline{c}, \bar{c})$. By definition of Δ, the following property holds.

Property III.1. *Given a constraint c, $\forall i \in 1 \,..\, |c|$ either $\bar{c}[i] = \underline{c}[i] = 0$ or $\bar{c}[i] \geq \underline{c}[i] + 2$.*

In particular, $\bar{c}[i] = \underline{c}[i] = 0$ represents the absence of constraints for the i-th link outgoing from router 0. Intuitively, a constraint defines the interval of weights that allows for breaking a loop L. Both $\underline{c}$ and $\bar{c}$ are vectors of size $k = |c|$ where k is the out degree of router 0. In particular, $\underline{c}$ represents the set of minimum relative weight increments to be set on the corresponding link in order to force one router in L to use at least one of its final forwarding path (recall that we consider ECMP in our model), while weight increments equal or higher than $\bar{c}$ cause all the routers in L to use at least one of their respective final paths (not including 0). Moreover, for any increment of link weights positively greater than $\underline{c}$ and smaller than $\bar{c}$, a subset of routers in L switch *definitively* to their respective final paths (i.e. they do not use anymore any paths including node 0). Note that we consider here the simple case of applying only a single weight increment.

More formally, we define a weight increment as a vector with k components (where k is the number of links incidents to 0). We say that a weight increment v *meets* a constraint $(\underline{c}, \bar{c})$ if $v >^+ \underline{c}$ and $\exists i \in 1 \,..\, k \mid v[i] < \bar{c}[i]$. We also say that a weight increment v *precedes* a constraint c iff $\exists i \in 1 \,..\, k \mid v[i] \leq \underline{c}[i] \neq 0$. On the contrary, a weight increment v *follows* constraint c iff $\forall i \in 1 \,..\, k \mid v_2[i] \geq \bar{c}[i]$.

By definition of Δ, the following properties hold for loops involving an arbitrary number of routers, and in case of asymmetric link weights.

Property III.2. *Given a loop L and its corresponding constraint c, if a weight increment v meets c then the router $x \in L$ verifying $\Delta(x) = \underline{c}$ is ensured to definitively switch to its final next-hop. It no longer uses its initial paths through 0.*

Property III.3. *Given a loop L and its corresponding constraint c, if a vector w precedes c then any router $x \in L$ still uses its initial forwarding paths.*

Property III.4. *Given a loop L and its corresponding constraint c, if a vector w follows c then any router $x \in L$ only uses its final forwarding paths.*

As an example, consider again Fig. 1. The possible convergence loop $L_1 = \{a, b, a\}$ is encoded as the following constraint $c_1 := (\underline{c}_1 = (6,0,0,0); \bar{c}_1 = (8,0,0,0))$. Let v_1 and v_5 be two weight assignments defined as in the right part at the bottom of the figure. Then, we have that v_1 meets constraint c_1. On the contrary, vector v_5 follows c_1, i.e., does not meet c_1, because $v_5 > \bar{c}_1$ (and the same for vector $\Delta_1^0(b)$ which is equal to $\bar{c}_1$, b enters in a ECMP state using such a vector). Observe that, despite the strict requirement that c_1 enforces, v_1 is not the only vector that meet c_1. Indeed, a vector such as $(7,10,10,10)$ also meets c_1, because if the value of a component i of a vector v is between the i-th component of $\underline{c}$ and of $\bar{c}$, then any other components $j \neq i$ of v can be arbitrarily higher than $\underline{c}[j]$. This is a crucial observation to compute minimal sequence of link weight assignments that satisfy several constraints at the same time.

C. Defining Safe Weight Increment Sequences

In this paper, we aim at computing sequences of weight increments, i.e., *weight sequences*, that provably prevent any convergence loops. A weight sequence $v_0, \ldots, v_m$ must also ensure that v_0 and v_m correspond to the initial and final routing state (i.e. v_m should be equivalent to a $(\infty, \ldots, \infty)$ vector metric for the node shut).

We now define a sufficient and necessary condition for a weight sequence to prevent a loop L. Let $c = (\underline{c}, \bar{c})$ be the constraint corresponding to L. We say that a weight sequence s includes an *unsafe transition* for c if it contains two

consecutive vectors v_i and v_{i+1} such that either i) v_i precedes $\underline{c}$ and v_{i+1} follows $\bar{c}$; or ii) vice versa, v_i follows $\bar{c}$ and v_{i+1} precedes $\underline{c}$ (decreasing sequence case).

Theorem 1. *A weight sequence s avoids a loop L if and only if all pairs of successive vectors of s form a safe transition with respect to the constraint corresponding to L.*

Proof: Let $c = (\underline{c}, \bar{c})$ and d respectively be the constraint and the destination associated to loop L. We prove the statement in two steps.

- if s includes an unsafe transition for a constraint, then s does not avoid L. Indeed, by definition of unsafe transition, we have two cases. Let v_i and v_{i+1} be the two consecutive vectors involved in the unsafe transition.
  - v_i precedes c and v_{i+1} follows c. By Property III.3 and III.4, all the routers in L will switch from their initial next-hop to their final next-hop to d.
  - v_i follows c and v_{i+1} precedes c. By Property III.3 and III.4, all the routers in L will switch from their final next-hop to their initial next-hop to d.

  In both cases, by definition of L, the transition from v_i to v_{i+1} can cause L to occur.
- if s does not include an unsafe transition for c, then s provably avoids L. Indeed, if there are no unsafe transitions, then, by definition, there exists a safe transition meeting c. From a vector $v_i \in s$ preceding c to a vector $v_j \in s$ $(j > i+1)$ following c, and vice versa, we know that there exists at least one vector v_k, $_{i<k<j}$, meeting c such that v_i, v_k is a safe transition. Hence, each time all routers in L change next-hop from their initial to their final routing state there is an intermediate step k, in which some router update breaks the possibility of the loop to occur by Property III.2.

The statement follows by applying the same argument to all loop constraints. ■

Intuitively, this theorem implies that, for each constraint $(\underline{c}, \bar{c})$, at least one vector must meet the constraint for each transition from weight increments smaller than $\underline{c}$ to increments greater than $\bar{c}$, and vice versa for weight increments that may actually imply some weight decreases. Indeed, always increasing sequences (i.e. that only include actual increments) are a subset of weight sequences space. A sequence $s = (v_0, \ldots, v_m)$ is *always increasing* if $\forall i \in 1 \,..\, m, v_i > v_{i-1}$. We now state a simplified version of Theorem 1 which holds for such sequences.

Theorem 2. *An always increasing weight sequence s avoids a loop L if and only if s contains at least one vector meeting the constraint corresponding to L.*

Proof: Let $c = (\underline{c}, \bar{c})$ be the constraint corresponding to any loop L. By definition of always increasing sequence, s is a concatenation of three subsequences, $s = l\ m\ h$, where l is composed by vectors preceding $\underline{c}$ (including the initial weights), m contains vectors meeting c, and h includes vectors following $\bar{c}$ (including the target weights). By hypothesis, m cannot be empty. As a consequence, s cannot contain any unsafe transition. The statement then follows by Theorem 1. ■

```
function GBA(G, n)
    Minimal sequence: S
    Set of constraints: CS
    Cardinal of the set of neighbors of n: k
    /* 1) Extract constraints */
    for d in N do
        for x in N do
            for i ∈ 1 .. k do
                Δ^n_d(x)[i] = C'(x,d) − C(x,i,d)
        cs = enum_cycles(RSPDAG(d) ⋃ RSPDAG'(d))
        for c in cs do
            CS.add ({c.min(Δ[d]), c.max(Δ[d])})
    /* 2) GBA main Loop */
    while CS ≠ ∅ do
        reinit (gv)
        /* 2a) Compute the current greedy vector */
        for c in CS do
            for i ∈ 1 .. k do
                gv[i] = max (gv[i], c̲[i] + 1)
        /* 2b) Build S in a backward fashion */
        S.append (gv)
        /* 2c) Remove satisfied constraints */
        for c in CS do
            for i ∈ 1 .. k do
                if c̄[i] > gv[i] then
                    CS.remove (c)
    return S
```

Fig. 3: The GBA algorithm

In the following, we show an algorithm that solves the node shut problem by computing always increasing weight sequences. Intuitively, sequences that do not verify such a monotonic behavior cannot help to reduce the number of intermediate vectors.

IV. Computing minimal sequences with GBA

In this section we start by presenting at a high level our main contribution, GBA, that stands for ***Greedy Backward Algorithm***. Its purpose is to enable a loop free convergence for all destinations with a minimal operational impact. After describing and illustrating GBA at a high level, we first prove that GBA weight sequences provably prevents convergence loops and have minimal size. Moreover, we discuss how to implement GBA as an efficient algorithm, whose running time is polynomial in the number of routers in the network. Time efficiency of GBA makes it practical: we argue that it can be directly supported in current routers operative systems. Finally, we describe simple solutions to deal with *floops*.

The GBA algorithm is presented in Figure 3. The algorithm can be divided into two macro-steps:

1) Extract the constraints corresponding to all potential forwarding loops;
2) Iteratively compute the weight sequence:

2a) compute a greedy weight increment gv that meets constraints c with highest $\underline{c}$ values;
2b) append gv in the weight sequence;
2c) and update the set of constraints still to be met.

The algorithm stops when all the constraints are met.

Loop-constraints may be computed by a standard cycle enumeration algorithm considering the merging of the initial and final $RSPDAG$. We discuss how to efficiently compute loop-constraints and satisfy them in Section IV-B.

Figure 1 and Table I illustrate how GBA works. Considering all destinations and their related cycles, the first greedy vector computed with GBA is equal to $(9\ 9\ 8\ 0)$ and satisfies constraints/cycles c_2, c_3 and c_5. These three constraints are met using a single weight increment. Since it does not meet the two remaining constraints due to an unsafe transition, at least one more iteration is needed. GBA performs a second step and computes vector $(7\ 2\ 3\ 0)$ that is sufficient and necessary to deal with the two last constraints. The length of the resulting sequence depends on the possibility of satisfying multiple constraints at the same time. In practice, GBA is able to produce small sequences because chains of successive dependent cycles (verifying in particular $\underline{c}_y \geq \bar{c}_x$) are limited in size (in $O(|N|)$ at worst for a given destination), hence constraints can often be met with only one weight increment, as shown in Section V.

Observe that the weight sequence is computed in a backward fashion, i.e., in the opposite order with respect to how they will be applied. Using a reverse order enables us to build the weight sequence greedily. On the contrary, a greedy forward search as the one provided in [7] does not ensure minimality of the computed sequence, as shown in the example in Table I. This significant difference with previous works is due to the asymmetry induced by constraint satisfaction: a vector v meets a constraint $(\underline{c}, \bar{c})$ if and only if $v >^+ \underline{c}$ and $v \not\geq \bar{c}$. The $\min$ and $\max$ rules do not form similar metric intervals such as with scalars. The $\min$ rule is much more restrictive than the $\max$ one in the node shut problem.

In the following, we discuss safety, minimality and efficiency properties of GBA.

A. GBA *Sequences Are Safe and Minimal*

We now prove that GBA computes weight sequences that prevent convergence loops. And, most of all, weight sequences computed by GBA are of minimal size. In our proofs, we rely on the properties of GBA that hold by definition of the algorithm. We say that a constraint is *unsatisfied* at an iteration j if it is not met by any vector of the weight sequence currently computed by GBA. Note that the term *before* denotes all previous iterations considering a backward sequence building.

Property IV.1. *At each iteration j,* GBA *computes a vector v such that $v >^+ \underline{cs}$ for all the constraints $c \in cs = (\underline{cs}, \bar{cs})$ still unsatisfied before j.*

Property IV.2. *At each iteration j,* GBA *computes a vector v such that for each component i, a constraint $c = (\underline{c}, \bar{c})$ exists such that $v[i] = \underline{c}[i] + 1$ and c is not met by any vector computed by* GBA *before j.*

Property IV.3. GBA *computes always increasing sequences.*

Property IV.4. GBA *stops as soon as all the constraints are met. Each of them is met by at least one vector in the returned sequence.*

Properties IV.1 and IV.2 are ensured by vector computation - 2a). Property IV.3 is the result of both vector computation and constraint removal - 2b). Properties IV.4 derives from the constraint removal mechanism - 2c). Properties IV.1 and IV.2 directly lead to the following important lemma.

Lemma 1. *At each iteration,* GBA *computes a vector v that meets at least one constraint not met before.*

Proof: Consider any iteration j of GBA. Let cs be the set of unsatisfied constraints at the beginning of iteration j, and let v the vector computed by the algorithm during the iteration j. By Property IV.1 and IV.2, there must exist at least one constraint $c \in cs$ such that $c = (\underline{c}, \bar{c})$, $v >^+ \underline{c}$ and $\exists i : v[i] = \underline{c}[i] + 1$. By Property III.1, then $\exists i : v[i] < \bar{c}[i]$. This means that c is met by v, hence the statement. ■

Lemma 1 ensures that GBA always terminates in a finite number of iterations. We now prove the safety and minimality of the algorithm. To this end, we refer to the following problem.

Problem 1. *Constraint Minimal Meeting Problem (*CMP*): Given a set $cs = \{(\underline{c}_1, \bar{c}_1), \ldots, (\underline{c}_n, \bar{c}_n)\}$ of loop-constraints, compute a minimal weight increment sequence which contains no unsafe transition for any constraint in cs.*

Provided that all the loop-constraints are correctly enumerated, solving CMP implies preventing all possible convergence loops by Theorem 1. First of all, we show that GBA is correct, that is, it computes safe weight sequences such that no convergence loop can occur.

Theorem 3. *Given a* CMP *instance I,* GBA *computes sequences that prevent convergence loops.*

Proof: By Property IV.4, GBA stops when all constraints are met. Moreover, Property IV.3 states that GBA computes always increasing sequences. Hence, the statement directly follows by Theorem 2. ■

We now prove the minimality of the sequences computed by GBA.

Lemma 2. *Consider a* CMP *instance I. Let $s = (v_1 \ldots v_n)$ be any sequence solving I, and let $g = (g_1 \ldots g_m)$ be the sequence computed by* GBA *on I, with possibly $n \neq m$. Then, the last respective vectors verify $v_n \geq g_m$.*

Proof: Assume by contradiction that $v_n[i] < g_m[i]$ for a given component i. By Property IV.2, for the same component i there must exist at least one constraint c in I, such that $g_m[i] = \underline{c}[i] + 1$. This implies that $v_n[i] \leq \underline{c}[i]$, i.e., the constraint vector c is not met by v_n. Since v_n is the last metric

increment in s and the final weight assignment is greater or equal than $\bar{c}$ to ensure all routers enter their final state, then s contains an unsafe transition for c. Thus, Theorem 1 implies that s is not a solution for I, contradicting the hypothesis. ■

Lemma 3. *Consider a* CMP *instance I. Let $s = (v_1 \dots v_n)$ be any sequence solving I, and let $g = (g_1 \dots g_m)$ be the sequence computed by* GBA *on I, with possibly $n \neq m$. Then, all the constraints met by v_n (and possibly more) are also met by g_m.*

Proof: Assume by contradiction that a constraint c is met by v_n, but not by w_m. This would imply that:

- in order for c to be met by v_n, $\exists i \mid v_n[i] < \bar{c}[i]$.
- by Property IV.1, $g_m >^+ \underline{c}$. Hence, if g_m does not meet c, it must be $g_m >^+ \bar{c}$. Also, Lemma 2 implies that $v_n \geq g_m$, hence $v_n[i] > \bar{c}[i],\ \forall i=1,\dots,k$.

Those two conditions cannot be satisfied at the same time, contradicting the assumption and yielding the statement. ■

Theorem 4. *The* GBA *algorithm finds a minimal sequence for any* CMP *instance I.*

Proof: Let $s = (v_1 \dots v_n)$ and $g = (g_1 \dots g_m)$ respectively be any minimal solution (possibly not always increasing) of I and the sequence computed by GBA on I, with $n \leq m$. If $m = 1$, n must be equal to 1 as well, and the statement trivially follows. Otherwise, by Lemma 3, we know that if g_m meets a set cs of constraints, then v_n meets a subset of constraints $\in cs$. Consider now the sequences $(v_1 \dots v_{n-1})$ and $(g_1 \dots g_{m-1})$. Again by Lemma 3, g_{m-1} meets at least the same set of constraints that v_{n-1} meets. This implies that the sequence $(g_{m-1}\ g_m)$ meets the same constraints (and possibly more) than $(v_{n-1}\ v_n)$. By iterating the same argument on all the elements in s, we have that $(g_{m-n+1} \dots g_m)$ meets at least the same set of constraints as $(v_1 \dots v_n)$. Observe that, by definition of s, all the constraints in I must be met by $(v_1 \dots v_n)$, hence by $(g_{m-n+1} \dots g_m)$. Moreover, by Property IV.4, GBA stops when all the constraints are met, i.e., at g_{m-n+1}. Hence, it must be $m - n = 0$ and $|g| = |s|$, yielding the statement. ■

Theorem 3 and 4 respectively prove the safety and minimality of the weight sequences computed by GBA. Since GBA computes always increasing sequences, this implies that among the set of minimal sequences, there exists at least one which is always increasing. In other words, restricting to always increasing sequences does not limit our ability to optimally solve the node shutdown problem.

B. GBA *Can Be Implemented as An Efficient Algorithm*

The algorithm presented in Figure 3 is not time efficient. In particular, the first enumerative step can require an amount of time which is exponential with respect to the size of the input (i.e., the number of routers in the network). Indeed, the identification of constraints is based on the enumeration of cycles on the graph obtained by merging the initial and the final $RSPDAG$s. Since the number of cycles in a graph can be exponential with respect to the number of nodes, then enumerating all the cycles in a graph cannot be performed efficiently. In the following, we discuss a new version of GBA in which the loop-constraints are identified with a less time consuming procedure. We show that this new version is time efficient. Intuitively, this efficient version is based on the observation that we can produce a number of constraints that is significantly lower than the number of possible convergence loops. In this case, a convergence loop is always associated to a constraint, but meeting a constraint translates to prevent multiple loops. Due to space limitations, we will not enter into all the algorithmic details of the efficient version of GBA.

1) Extracting and updating constraints in polynomial time: To build an efficient version of GBA, we replace the explicit cycle enumeration with a more lightweight partial constraint detection. The key observation is that GBA only requires the $\max$ constraints for all destinations (see Fig. 3, Step 2a)) at each iteration. Hence, in our efficient version of GBA, we identify a subset of constraints during each iteration instead of enumerating all of them at the beginning of the algorithm.

More precisely, for any destination $d \in N$ and its related set of constraints cs, we thus look for routers x such that $\Delta_d^0(x) = \max_{\forall c \in cs}(\underline{c})$. An efficient constraint computation is achieved by considering each destination independently to divide the problem space, and computing a superset of Δ vectors that includes $\underline{cs}$. This computation relies on a standard cycle detection algorithm. In particular, we use the following algorithm, which is linear in the number of links in the network. Consider the graph resulting from merging two $RSPDAG$s, namely, $RSPDAG_1$ and $RSPDAG_2$. First, all the nodes with no incoming or no outgoing edges are removed from the merged graph, as they cannot be part of any convergence loop. Then, let δ be the minimum Δ_d^0 value as computed in the remaining graph. We update the graph by simulating an increase of δ on the merged graph. We iterated this update until all the nodes are visited. The last δ is the maximum value in $\underline{cs}$. Indeed, $\delta \leq \max(\underline{cs})$, otherwise the merged graph would have had no cycle and the algorithm would have stopped before this iteration. Moreover, if $\delta < \max(\underline{cs})$, then a still unsatisfied cycle must exist, preventing the algorithm to terminate.

Observe that in this optimized version of GBA, we do not need to compute $\bar{c}$-vectors anymore. Indeed, in the original version of GBA, $\bar{c}$-vectors are only used to check which constraints are met by the vector computed during the current iteration, see Fig. 3. This is not needed anymore as constraints are updated at each macro-step iteration of the algorithm.

2) Overall complexity analysis: Let k be the degree of the node to be removed. The size of a minimal weight sequence for an arbitrary weight change on a single link is in $O(|N|^2)$ [7]. Hence, removing one link at a time leads to sequences of at most $k|N|^2$ elements. This property implies Theorem 5.

Theorem 5. GBA *terminates in a number of main loop iterations that is polynomial with respect to the number of routers in the network.*

Proof: Lemma 1 ensures that GBA always terminates.

Moreover, at each iteration, GBA appends a weight assignment to the weight sequence, and never backtracks. As a consequence, GBA performs a number of iterations that is equal to the size of the weight sequence it computes. Finally, the maximal sequence size property given in [7] (the length of any link-shut sequence is lower than $|N|^2$) and Theorem 4 ensure that the number of iterations is at most $k|N|^2$, where k is the number of links of the router to be shutdown. ■

Theorem 5 is relative to the weight sequence computation macro-step. Of course, listing the loop-constraints must also have to be taken into account. With the efficient constraint technique described in Section IV-B, step 1) and step 2c) are modified, while step 2a) and 2b) are left unchanged. In particular, the efficient version of GBA computes constraints on a per-iteration/per-destination basis, using a modified cycle detection procedure that can be efficiently implemented (see Section IV-B). Hence, we were able to implement a GBA version that is polynomial in time.

Note that, in practical cases, long weight sequences are not interesting, as applying them would increase the convergence time and control-plane overhead too much. Thus, a given value p can be defined as maximum length of weight sequences. In this case, GBA can be even more efficient, by stopping the main loop after at most p iterations. Using a reasoning similar to the proof of Theorem 5, it is easy to show that stopping GBA after at most p iterations can be used to verify that there exists a safe sequence s for node n verifying $|s| \leq p$ and compute it (if it exists).

Due to space limitations, we cannot enter in all the details of GBA complexity. The version of GBA we design has a worst case complexity in $O(p \times |N|^3)$ if $p < N$, or $O(|N|^4)$ otherwise. Our implementation of the algorithm always results in running times that are on the order of a second (even for worst case nodes on our largest IGP topologies – about 1100 nodes).

C. Preventing Floops

Both the basic GBA algorithm and its time-efficient version only consider constraints extracted from the merging of the initial and final $RSPDAG$. As a consequence, they do not solve the floop problem (see Section II).

Nevertheless, GBA can be extended to take floops into account, by adding new transient constraints at each iteration of the algorithm. In particular, during the update of the set of unsatisfied constraints, the $RSPDAG$ corresponding to the last weight assignment computed by the algorithm is merged with the initial $RSPDAG$, and transient constraints may be identified on this merging. Unfortunately, adding constraints at each iteration possibly prevents GBA to find a minimal weight sequence. Fig. 2 provides such an example ($FF2$ versus $FF1$): removing a floop as a usual loop (i.e., in a greedy fashion) does not necessarily result in a minimal sequence because there exists other ways to avoid it. The issue of finding the minimal floop-free weight sequence with an efficient algorithm remains as an open problem. However, our experiments on real-world IP networks show that the overhead of the floop-preventing extension of GBA is almost negligible (considering the length of the sequence).

An alternative solution to this extension is to deal with the flooping problem at the technological or the protocol level, rather than at an algorithmic level. For example, introducing a local convergence delay on the router to be shut down and freezing data plane decisions in the meanwhile, as proposed in a recent Internet draft [12], would avoid the flooping problem. A similar mechanism can be directly supported into a new generation of link-state IGPs fully supporting loop-free convergence in case of network topology changes.

V. Evaluation: GBA sequences are short

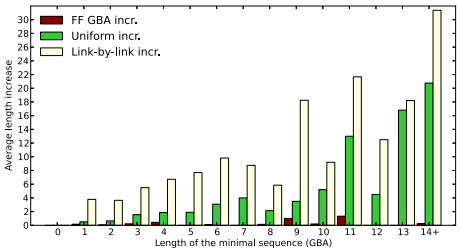
This section aims at evaluating the sequence length resulting from our algorithms on real IP networks. First, we briefly introduce the graphs we use, as well as our evaluation environment. We then present the results, in terms of sequence length, obtained with several algorithms. As in previous sections, we focus here on the node withdrawal for the sake of simplicity (as it gives maximal sequence sizes).

We evaluated our algorithms on a wide set of real IP network graphs of various shape and size. Table 4a provides a general overview: Abilene and GEANT networks are two well-known IP networks whose weighted graphs are freely available [13], [14]. The last six networks are real ISP that we anonymized for confidentiality reasons. Evaluated algorithms (including GBA) are implemented in C and are available on demand. On most networks, GBA takes a few milliseconds to compute a minimal increment sequence, and up to $5s$ on ISP6 (which is by far the largest network) on the worst case node. Table 4a provides key points in the distributions of sequence lengths. In particular, it gives the percentage of empty sequences as well as, for each algorithm, the proportion of sequences containing 5 intermediate increments or less and the length of the longest sequence.

Interestingly enough, we observe that, apart from the ring architecture of Abilene, no intermediate increment is required for most of the nodes of a network. Such situation appears mainly on the rim of the network, for nodes that are used to reach a few number of destinations. Such nodes can be safely and easily removed from the network without worrying about transient loops. On the other hand, core nodes on largest networks may require up to dozens of intermediate increments. Performing each increment of long sequences is not realistic as it would result in too much convergence time and IGP churn. Hopefully, such extreme cases seldom happen and most of the sequences are much smaller. If one considers 5 intermediate metrics to be a reasonable upper bound for a maintenance operation on a node, we observe that at least 80% of the nodes for all topologies meet such requirement. Besides, this proportion increases to 100% for topologies with less than 100 nodes. We also notice that uniform sequences are often longer (up to three times) than those produced with both versions of GBA. The floop free version only slightly increases the sequence sizes, so that the proportion of *short enough* sequences is almost the same as with the standard version.

Topology	#nodes / #edges	$S = \emptyset$	Uniform		Std GBA		FF GBA	
			$\|S\| \leq 5$	max	$\|S\| \leq 5$	max	$\|S\| \leq 5$	max
Abilene	11 / 28	36.4 %	100 %	3	100 %	3	100 %	3
GEANT	22 / 72	63.6 %	100 %	5	100 %	3	100 %	4
ISP1	25 / 55	69.2 %	100 %	4	100 %	4	100 %	4
ISP2	55 / 195	81.5 %	94.4 %	7	100 %	3	100 %	3
ISP3	110 / 340	59.1 %	81.8 %	21	90.9 %	10	89.1 %	11
ISP4	140 / 410	67.4 %	85.8 %	21	92.9 %	10	91.5 %	11
ISP5	210 / 785	56.7 %	74.8 %	63	82.9 %	33	82.9 %	36
ISP6	1170 / 7240	84.2 %	92.1 %	147	93.2 %	57	93.1 %	57

(a) Global result table



(b) Sequence length increase on ISP6

Fig. 4: Evaluation results

On Fig. 4b, we evaluate the efficiency of non-optimal algorithms, by representing their average length increase compared to the minimal referent sequence, obtained with GBA. We first notice that the floop free GBA, in red, remains quite close to the standard version with less than one extra increment on average, even for large sequences. In the event that introducing a local convergence delay on the failing node is not possible, this technique would hence allow to prevent flooping issues at a negligible overhead. On the other hand, the length of uniform sequences, in green, rapidly grows for larger sequences, reaching more than twice the minimal length for 11 and 13 (the number of additional increments is proportional to the GBA sequence length). Thus, while the uniform heuristic is a nice alternative to GBA for short sequences, as it completely prevents flapping, it should be avoided for longer ones. One can also notice the low performances of a link-by-link technique. Our results clearly indicate that link based heuristics are not sufficiently efficient compared to GBA, which is able to compute shortest sequences for node operations.

VI. Conclusion

To enable graceful router-wide updates in link-state routing networks without changing protocol specifications, we designed an optimal metric update algorithm, called GBA. In the case of a router addition or removal, our algorithm progressively shifts of the traffic flows in or out of the updated router, while provably avoiding transient forwarding loops. The need for such a router-wide solution is motivated by the inefficiency of link-based algorithms in terms of number of intermediate steps. On the contrary, GBA induces minimal operational impact and we show that graceful update sequences are short in practice. Also, GBA is a polynomial time algorithm that only requires few seconds on large real-world IP network topologies. Finally, we studied how to prevent specific loops related to the graceful router update problem. They occur in case routers does not support features like local convergence delay. We thus proposed a straightforward extension of GBA that guarantees their prevention at the cost of a negligible overhead on the operational impact.

In the future works, we plan to define a general solution for the problem of the transition from a given set of links metric configuration to another one. Such an approach could be used in conjunction with traffic-engineering tools suggesting link metric reconfigurations at independent locations. We also envision evaluating and extending our solutions on other type of networking environments, such as data center topologies, where router states may be dynamically switched according to the traffic load.

Acknowledgments

We thank Olivier Bonaventure for insightful comments on a preliminary version of this work. This work has been partially supported by the European Community's Seventh Framework Programme (FP7/2007-2013) Grant No. 317647 (Leone).

References

[1] P. Pongpaibool, R. Doverspike, M. Roughan, and J. Gottlieb, "Handling IP Traffic Surges via Optical Layer Reconfiguration," in *Proceedings of the Optical Fiber Communication Conference and Exhibit (OFC)*, Anaheim, CA, USA, March 2002, pp. 427 – 428.

[2] A. Markopoulou, G. Iannaccone, S. Bhattacharyya, C.-N. Chuah, Y. Ganjali, and C. Diot, "Characterization of Failures in an Operational IP Backbone Network," *IEEE/ACM Transactions on Networking*, vol. 16, pp. 749–762, August 2008.

[3] A. Medem, R. Teixeira, N. Feamster, and M. Meulle, "Joint analysis of network incidents and intradomain routing changes," in *Proceedings of the International Conference on Network and Service Management (CNSM)*, Niagara Falls, Canada, October 2010, pp. 198–205.

[4] H. Pucha, Y. Zhang, Z. M. Mao, and Y. C. Hu, "Understanding Network Delay Changes Caused by Routing Events," *SIGMETRICS Performance Evaluation Review*, vol. 35, no. 1, pp. 73–84, June 2007.

[5] Y. Zhang, Z. Morley Mao, and J. Wang, "A Framework for Measuring and Predicting the Impact of Routing Changes," in *Proceedings of IEEE INFOCOM'07*, Anchorage, Alaska, USA, May 2007, pp. 339 –347.

[6] H. Ito, K. Iwama, Y. Okabe, and T. Yoshihiro, "Avoiding Routing Loops on the Internet," *Theory of Computing Systems*, vol. 36, pp. 597–609, 2003.

[7] F. Clad, P. Merindol, J.-J. Pansiot, P. Francois, and O. Bonaventure, "Graceful Convergence in Link-State IP Networks: A Lightweight Algorithm Ensuring Minimal Operational Impact," *IEEE/ACM Transactions on Networking*, 2013, to appear.

[8] L. Vanbever, S. Vissicchio, C. Pelsser, P. Francois, and O. Bonaventure, "Lossless Migrations of Link-State IGPs," *IEEE/ACM Transactions on Networking*, vol. 20, no. 6, pp. 1842–1855, December 2012.

[9] C. Hounkonnou and E. Fabre, "Enhanced OSPF Graceful Restart," in *Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management (IM)*, 2013, to appear.

[10] J. Martin and A. Nilsson, "On Service Level Agreements for IP Networks," in *Proceedings of IEEE INFOCOM'02*, vol. 2, New York, NY, USA, June 2002, pp. 855 – 863.

[11] L. Vanbever, S. Vissicchio, L. Cittadini, and O. Bonaventure, "When the cure is worse than the disease: the impact of graceful igp operations on bgp," in *IEEE INFOCOM*, 2013.

[12] S. Litkowski, B. Decraene, and P. Francois, "Microloop prevention by introducting a local convergence delay," IETF, Internet-Draft, 2013.

[13] "Abilene Network Topology." [Online]. Available: http://www.internet2.edu/info/#maps

[14] "GEANT Network Topology." [Online]. Available: http://www.geant.net/Network/The_Network/Pages/Network-Topology.aspx