Graceful Convergence in Link-State IP Networks: A Lightweight Algorithm Ensuring Minimal Operational Impact

François Clad, Pascal Mérindol, Jean-Jacques Pansiot, Pierre Francois, *Member, IEEE*, and Olivier Bonaventure, *Member, IEEE*

***Abstract*—The use of real-time multimedia or mission-critical applications over IP networks puts strong pressure on service providers to operate disruption-free networks. However, after any topological change, link-state Interior Gateway Protocols (IGPs), such as IS-IS or OSPF, enter a convergence phase during which transient forwarding loops may occur. Such loops increase the network latency and cause packet losses. In this paper, we propose and evaluate an efficient algorithm aimed at avoiding such traffic disruptions without modifying these IGPs. In case of an intentional modification of the weight of a link (e.g., to shut it down for maintenance operations or to perform traffic engineering), our algorithm iteratively changes this weight, splitting the modification into a sequence of loop-free transitions. The number of weight increments that need to be applied on the link to reach its target state is minimized in order to remain usable in existing networks. Analysis performed on inferred and real Internet service provider (ISP) topologies shows that few weight increments are required to handle most link shutdown events (less than two intermediate metrics for more than 85% of the links). The evaluation of our implementation also reveals that these minimal sequences can be computed in a reasonable time.**

***Index Terms*—Convergence, IP networks, network theory (graphs), routing protocols.**

I. Introduction

THE GROWING popularity of real-time media services over Internet, such as TV broadcast, voice- or video-over-IP, and gaming, has changed the requirements of Internet service providers (ISPs) on the performance of routing protocols supporting those services. Non-Internet IP-based services such as VPNs have also led to ISPs facing ever more stringent Service Level Agreements (SLAs) [1]–[4], defining the performance of an ISP through various metrics such as packet losses and service availability.

Breaches in service availability are generally due to side effects of network topological changes [5]–[7], which are common events in large IP networks. In IP-over-optical networks, the topology can be regularly reconfigured according to the needs of the operators [8]. A study on the Sprint IP backbone [9] reports that 20% of topological changes are caused by maintenance operations. Moreover, several other studies reveal that such operations occur frequently [10]. Maintenance operations are mainly performed during nightly scheduled maintenance windows in order to reduce their impact on the traffic. However, this increases the cost of operating the network and reduces its flexibility at the time when it is actually most likely to undergo traffic engineering issues. Another possible kind of topological change is the intentional modification of Interior Gateway Protocol (IGP) weights for traffic engineering purposes [11] in order to optimize routing according to traffic fluctuations. Finally, nonintentional changes such as link or router failures are also a great source of transient convergence problems, but their impact on the data plane can be limited thanks to fast-reroute techniques [12]–[14].

All these changes affect the traffic passing through the network. Each topological change compels the routers to recompute and update their forwarding information bases (FIBs), putting the network into a transient state during which forwarding loops may occur [15]. Indeed, using routing protocols such as Open Shortest Path First (OSPF) or Intermediate System To Intermediate System (IS-IS), the order of FIBs updates is not controlled: It depends on the dynamics of the flooding of the link state packets and on the router performances throughout the network [16]. As a result, the global data plane can be transiently inconsistent among routers, potentially leading to forwarding loops in the network during the convergence.

Such performance drop during convergence is particularly unfortunate when the topological change is planned, hence when convergence does not happen around a failed component blackholing traffic. Several methods have been proposed at the IETF to solve this problem, notably [17] and [18]. However, they require extensions to the OSPF and IS-IS protocols, implying software and/or hardware modifications. Such changes would possibly take years before being actually deployed in a favorable perspective. Some ISPs have defined procedures [19] to "gracefully" reroute the traffic out of a link, by setting its weight to a pseudo infinite metric, before actually shutting it down. However, this does not prevent packet losses from occurring as a result of transient forwarding inconsistencies.

In this paper, we present an efficient solution relying on iterative updates of the weight of the link whose state is being modified [20], [21]. Each intermediate update is selected to not

Manuscript received June 18, 2012; revised January 14, 2013; accepted February 15, 2013; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor C.-N. Chuah.

F. Clad, P. Mérindol, and J.-J. Pansiot are with the ICube Laboratory, CNRS, Univeristé de Strasbourg, 67000 Strasbourg, France (e-mail: fclad@unistra.fr; merindol@unistra.fr; pansiot@unistra.fr).

P. Francois is with the IMDEA, Universidad Carlos III de Madrid, 28918 Madrid, Spain (e-mail: pierre.francois@imdea.org).

O. Bonaventure is with the ICTEAM, Université catholique de Louvain (UCL), 1348 Louvain-la-Neuve, Belgium (e-mail: olivier.bonaventure@uclouvain.be).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNET.2013.2255891

cause any forwarding loop. Preserving the requirement of solely relying on existing link state updates and standard processing by the IGP routers, we first prove that sequences of intermediate weight increments providing loop-free convergence always exist in an IGP network supporting shortest path routing. Francois *et al.* presented in [20] the proof of concept of this solution and proposed to implement it inside a network management tool. This paper proposes more efficient algorithms that compute sequences of minimum size in a polynomial time. This reasonable time complexity let us believe that a router could compute such sequences on its own, hence simplifying the involvement of the network operator in the deployment of the solution.

Minimizing the length of resulting loop-free sequences is a critical concern to limit the convergence period of a link state IGP. Our minimal sequences are computed in two main steps. First, considering independently each destination in the network, our proposal computes *reduced* destination oriented metric sequences. These sequences are pruned from most of their unnecessary metrics, but still contain at least one metric for each cycle that may appear during the transition. Second, we merge such reduced sequences for all destinations and then minimize the resulting union to provide a shortest global loop-free sequence. Our minimization algorithm sequentially checks if cycles occur between two given metrics. We give particular attention to its time complexity. We show that the time necessary to compute a minimal sequence is at worst on the order of a second for very large IGP networks (e.g., having more than 1000 nodes and 4000 edges).

While the oFIB technique proposed in [18] updates each router once and for all (for all destinations at one time), our approach may put a router into a partially updated state where old paths are still used for a subset of the destinations. The oFIB algorithm provides a global ordering of upstream node updates whatever the destination is. Unfortunately, this requires extensions to the standardized routing protocols, which is very difficult to achieve. In terms of number of intermediate convergence periods, oFIB would require at most $O(|N|)$ of them, while the metric increment technique could possibly generate $O(|N|^2)$ metrics long sequences (and as many subconvergence periods). However, in practice, our approach often simultaneously updates several routers that would have to be updated one by one with oFIB. Thus, it frequently results in fewer convergence periods without requiring any protocol extension.

Theoretically, our approach could easily interact with fast-rerouting techniques [12]–[14], making it possible to handle unplanned changes. It could even be adapted to benefit from those techniques. However, in practice, and according to the nature of the protection scheme, technical issues can arise considering both failures and metric injection. They have to be addressed case by case before such a combination is actually deployable. This is outside the scope of this paper.

The remainder of this paper is organized as follows. In Section II, we first illustrate the transient forwarding loop problem on a small gadget and then introduce the main properties on which we design our solution. In Section III, we propose and prove algorithms that efficiently compute global minimal metric sequences. In Section IV, we perform an analysis over actual and inferred ISP topologies, highlighting that, in most cases, our global optimized sequences are very short. Our analysis also reveals that they can be computed in a very reasonable time on current commodity hardware (in the order of a second in worst cases).

II. Transient Forwarding Loops

This section introduces and illustrates the problem of transient forwarding inconsistencies that may occur during the convergence of link-state routing protocols. We first provide some notations, supporting our subsequent proofs. We also provide some properties highlighting the order in which routers should update their data plane after the weight change. Without loss of generality, we focus on the case of a link removal. The activation of a link can be performed by reversing the sequence obtained for the case of its removal. A weight modification leads to a subset of the sequences obtained for the link removal (or activation). Note that for completely removing a link, we need to compute a sequence for each direction, but we can apply them independently. Indeed, in a stable network, a link can only be used in one direction in the routing graph toward a given destination.

A. Definitions and Notations

$G(N, E, w)$ is a directed weighted graph with a set of nodes N, a set of edges E, and w a positive valuation function for edges. The notation $l = (a, b)$ refers to an edge $l \in E$ connecting node a to node b. Let ϵ be the smallest weight increment/decrement that can occur on a link. In practice, w is an integral positive valuation and $\epsilon = 1$.

$P(s, d)$ denotes the set of shortest elementary paths linking node s to node d, while $C(s, d)$ is the cost of paths $\in P(s, d)$. $SPDAG(s)$ is the shortest path directed acyclic graph (SPDAG) rooted at node s. $RSPDAG(d)$ is the reverse shortest path DAG (RSPDAG) rooted at d. It is a directed acyclic graph (DAG) containing all the shortest paths from the nodes of the graph toward d. Note that all notations account for equal cost multipath (ECMP) paths—that is using, possibly simultaneously, several forwarding paths having the same cost.

When a link weight change is performed, we respectively denote the RSPDAGs of d before and after the change as $RSPDAG(d)$ and $RSPDAG'(d)$. We say that a change is *loop-free* for destination d if no forwarding loops can occur during the convergence triggered by the change. A convergence period starts when the link-state advertisement (LSA) is flooded; it ends when all routers have updated their FIB after receiving the LSA. Such forwarding loops can be detected by merging $RSPDAG(d)$ with $RSPDAG'(d)$: If the merged graph contains cycles, then forwarding loops might occur. Otherwise, the weight change is *loop-free* for destination d. We say that a change is *loop-free* if it is *loop-free* for all destinations in the network.

Considering a given destination $d \in N$, let $G_l(d) \subset RSPDAG(d) \subset G$ be the subgraph impacted by the removal of a link $l = (a, b)$ such that each node in $G_l(d)$ must change its forwarding paths to d to avoid link l. Note that $G_l(d)$ forms an RSPDAG rooted at a: It contains all paths in $RSPDAG(d)$ that use the considered edge l minus paths $P(a, d)$. In the following, since we always consider the removal of a link

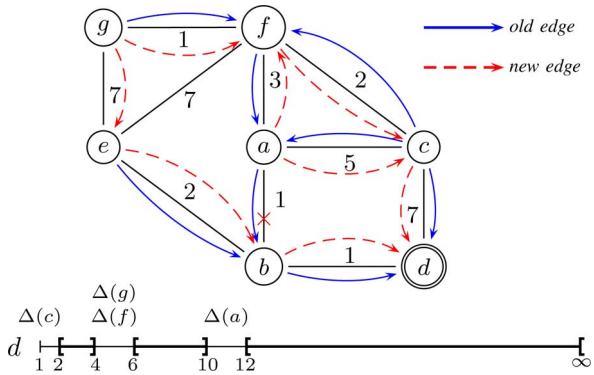


Fig. 1. Running gadget: paths toward d.

TABLE I
GENERAL NOTATIONS

$G(N, E, w)$	Directed weighted graph
$l = (a, b)$	Edge $l \in E$ connecting node a to b
ϵ	Smallest weight modification on a link
$P(s, d)$, $P(d)$	Shortest paths from s to d
$C(s, d)$, $C(d)$	Cost of the paths in $P(s, d)$
$P'(s, d)$, $P'(d)$	Shortest paths to d in $G(N, E \setminus l, w)$
$C'(s, d)$, $C'(d)$	Cost of the paths in $P'(s, d)$
$SPDAG(s)$	Shortest paths DAG rooted at s
$RSPDAG(d)$	Reversed SPDAG rooted at d
$RSPDAG'(d)$	RSPDAG rooted at d in $G(N, E \setminus l, w)$

$l = (a, b)$, we will refer to $G_l(d)$ as $G(d)$ to simplify our notations.

As all our notations are related to a destination d and a link l, we use $C(x)$ and $C'(x)$ to respectively refer to $C(x, d)$ and the cost of the best paths toward d in the graph $G(N, E \setminus l, w)$. In the same way, $P'(x)$ denotes the set of shortest paths linking nodes x and d in the graph $G(N, E \setminus l, w)$. Table I summarizes all the notations described above, which can be derived for any weight change.

B. Illustration on a Gadget

To illustrate how a transient forwarding loop can occur during the IGP convergence, let us consider the network shown in Fig. 1. If link (a, b) is shut down for maintenance reasons, it may cause several forwarding loops in particular for destination d. This link can either be shut down in one step by removing it from the link state database, or in two steps as proposed in [19] by first setting its weight to a pseudo infinite value and later removing it from the link state database. In both cases, after the first step, all routers must update their FIB. Before the removal of (a, b), routers g, f, c, and a in $G(d) \subset RSPDAG(d)$ use the link (a, b) toward d. After this change, their paths toward d form a subset of the routing graph, $RSPDAG'(d)$, represented with dashed arrows in Fig. 1 (instead of solid arrows for the pre RSPDAG, $RSPDAG(d)$). Before the removal of (a, b), router a was forwarding the packets destined to d via b, while it will use f and c to reach d after. This implies that if router a updates its FIB before router f (or c), at least one transient loop occurs. In this example, considering destination d, three nodes are possibly involved in four elementary cycles revealed by the merging of the two routing RSPDAGs. Formally, a cycle in a directed graph is elementary if each intermediate node appears only once.

Let us now show that there exists a sequence of metrics for link (a, b) that allows to shut it down without causing any transient loop nor packet loss. In the first stage, the weight of link (a, b) can be increased from 1 to 2: an atomic ϵ change. Among all routers receiving the LSAs, only router c actually modifies its FIB. It now only uses its direct outgoing link (c, d) instead of using both $(c, a), (c, f)$ and (c, d). Then, router f will definitively stop using its path via (a, b) when the weight of (a, b) reaches 6. Indeed, f has now a better path via c (which has already updated). Finally, router a will stop using b to reach d by using both c and f when $w(a, b) = 12$.

The sequence $\{2, 6, 12\}$ is a minimal loop-free sequence for destination d. If the delay between two successive increments is sufficiently long to allow all routers to update their FIB, this sequence provides a minimal loop-free convergence. The first two metrics allow for updating upstream nodes before downstream ones for paths in $RSPDAG(d)$. The last one is the first possible value that removes link (a, b) from all shortest paths. The two first changes avoid each elementary cycles in the merging of $RSPDAG(d)$ and $RSPDAG'(d)$ graphs (solid and dashed arrows in Fig. 1). Indeed, cycles $c - a$ and $f - a$ are removed thanks to subsequent updates $\{1, 2\}$ and $\{2, 6\}$, respectively. The cycles a–c and a–c–f can be avoided with one update inside range $[2, 10]$. This condition is already verified by each of the previous updates. Finally, note that node g is not involved in any cycle, so it can be ignored although it belongs to $G(d)$.

C. Existence of Loop-Free Metric Sequences

To prove that there always exists a sequence of weight increments that allows a loop-free convergence, we first demonstrate that incrementing the metric of a link by ϵ never causes transient loops. Hence, progressively incrementing the weight of a link, at worst by repeated increments of ϵ, can be performed without undergoing transient loops.

Theorem 1: In a stable network, incrementing the weight of a link by ϵ leads to a loop-free convergence process.

Proof of Theorem 1: Let us assume that there exists a forwarding loop between nodes $n_1, n_2, \ldots, n_i, \ldots, n_p = n_1$ during the convergence following the increase of the weight of link l by ϵ. We note $\mathscr{C}(n_i)$ as the distance according to which n_i is forwarding packets for destination d at the time the considered packet is forwarded by n_i along the loop. We will show that this loop is impossible because it would require $\mathscr{C}(n_1) > \mathscr{C}(n_1)$.

Let us observe that either n_i is not up to date and $\mathscr{C}(n_i) = C(n_i)$, or n_i is updated and $\mathscr{C}(n_i) = C'(n_i)$. Note that $C'(n_i) = C(n_i)$ or $C'(n_i) = C(n_i) + \epsilon$; the length of the shortest paths remains the same, not passing through l, or is increased by ϵ, passing through l. This gives $C'(n_i) \geq \mathscr{C}(n_i)$.

We then have the two following properties:

(P1): If n_i is updated, then $\mathscr{C}(n_i) = C'(n_i)$, by definition. If n_i is updated, then it forwards its packets toward neighbors that lie on its post-convergence paths toward d. From properties of shortest paths, this means that the post-convergence distance of these neighbors is strictly lower than the post-convergence

distance of n_i. We thus have $C'(n_i) > C'(n_{i+1})$ when n_i is updated, which gives $\mathscr{C}(n_i) > \mathscr{C}(n_{i+1})$.

(P2): If n_i is not updated, then $\mathscr{C}(n_i) = C(n_i)$, by definition. n_i is forwarding to neighbors that are lying along its old shortest paths toward d, which gives $C(n_i) > C(n_{i+1})$, as per properties of shortest paths. Hence, $C(n_i) \geq C(n_{i+1}) + \epsilon$. As per the definition of $C(n_{i+1})$, $C(n_{i+1}) + \epsilon \geq C'(n_{i+1})$. As per the definition of $\mathscr{C}(n_{i+1})$, we have $C'(n_{i+1}) \geq \mathscr{C}(n_{i+1})$. This gives $\mathscr{C}(n_i) \geq \mathscr{C}(n_{i+1})$, when n_i is not updated.

From *(P1)* and *(P2)*, we know that $\mathscr{C}(n_i) \geq \mathscr{C}(n_{i+1})$.

Note that $\mathscr{C}(n_i) = \mathscr{C}(n_{i+1})$ only when n_i is not updated, while n_{i+1} is updated and has increased its distance. If $\mathscr{C}(n_i) = \mathscr{C}(n_{i+1})$, we then have $\mathscr{C}(n_{i+1}) > \mathscr{C}(n_{i+2})$; if n_{i+1} is updated, we have *(P1)* for $i+1$ and thus a strict inequality. Therefore, we have $\mathscr{C}(n_i) > \mathscr{C}(n_{i+k}), \forall k \geq 2, \forall i + k \leq p$, which is in contradiction with the initial loop statement. A transient loop cannot occur for $p = 2$ and $k = 1$ either since a node never selects itself as its next hop. ■

When a link l has to be shut down, its metric can thus be progressively incremented by ϵ until it becomes sufficiently large to make l unused. However, such a technique can be inefficient, as a large number of weight increments would have to be issued when a wide range of metrics is used across the network. In order to overcome this limitation, we propose to perform larger increments of the weight when they are known to provide loop-free convergence. As the metric space may be wide in IS-IS and OSPF deployments, it is not realistic to completely explore it looking for a possible loop-free increment sequence for a given link metric transition. Indeed, many ISPs take advantage of the whole metric space width. For example, the weights assigned to the links of the European GEANT Research Network [22] are taken into the interval [1, 20 000]. Such a diversity in link valuation is also present in other ISPs we analyze in Section IV. In the following, we describe algorithms computing loop-free sequences without performing a complete exploration. Section II-D introduces main properties on which we build our computing approach.

D. From Old to New Paths, Main Properties

Before introducing the methodology, we first discuss some properties among the nodes using each other to reach a destination d before and after the application of a metric increment. For each path $(n_1, \ldots, n_i, \ldots, n_j, \ldots, n_m)$ in $G(d)$, $\forall i < j, n_i$ is an upstream node of n_j and reciprocally n_j is a downstream node of n_i. Let us introduce a fundamental notation to our purpose

$$\forall x \in N, \quad \Delta(x) = C'(x) - C(x).$$

Note that this notation actually depends on the target destination and the considered link. For the sake of clarity, we ignore those cumbersome indexes and recall considered destination and link only when necessary. $\Delta(x)$ is the minimum relative increment allowing node x to reach d by using paths avoiding link (a, b). The relative increment $\Delta(x) + \epsilon$ sets x in a final state where it has definitively converged, i.e., it does not use the considered link anymore. The relative increment $\Delta(x)$ also triggers an intermediate change in the forwarding plane of x for d. Node x is in an ECMP *transient state* where it uses both its old and new paths toward d, before using only new paths with $\Delta(x) + \epsilon$. For example, in Fig. 1, nodes f and g enter an ECMP transient state when $w(a, b) = 5$. They use respectively outgoing edges $(f, a), (f, c)$ and $(g, f), (g, e)$. Note that $\Delta(x) = 0$ for all nodes not in $G(d)$, but a node $x \in G(d)$ may also verify $\Delta(x) = 0$. A node x verifies $\Delta(x) = 0$ if at least one of its initial shortest paths to d does not contain link (a, b). Either node x does not use (a, b) at all to reach d, or it uses equal-cost shortest paths that do not contain this link. This is the case for node c in Fig. 1. The properties discussed in the following, as well as the algorithms presented in Section III, solely base themselves on the Δ value of each rerouting router, accounting for the set of affected destinations.

Let us now introduce three fundamental properties related to Δ differences. We call an *old* (respectively *new*) edge an edge (x, y) that starts a subset of paths in $P(x, d)$ but not in $P'(x, d)$ (respectively starts $P'(x, d)$ but not $P(x, d)$). When an edge (x, y) starts a subset of paths $\mathscr{P} \in P(x, d)$, we have $\mathscr{P} = (x, y) \circ P(y, d)$ (it is the same for P'). A *common* edge starts both a path in $P(x, d)$ and one in $P'(x, d)$ (the two paths may differ further). Let (x, y) be an edge in G; we have the three following properties.

Property 1: If (x, y) is an *old* edge, then $\Delta(x) < \Delta(y)$.

Property 2: If (x, y) is a *new* edge, then $\Delta(x) > \Delta(y)$.

Property 3: If (x, y) is a *common* edge, then $\Delta(x) = \Delta(y)$.

Proofs are given in Appendix-A. Fig. 1 illustrates such properties. For $x = a, b, c, d, e, f, g$, we have $\Delta(x) = 5, 0, 0, 0, 0, 4, 4$. Along the ECMP path c–f–$a \in RSPDAG(d)$, we can notice that $\Delta(c) < \Delta(f) < \Delta(a)$. Reciprocally along a new forwarding path such as a–f–c, we have $\Delta(a) > \Delta(f) > \Delta(c)$; we also have $\Delta(g) > \Delta(e) = 0$ on the new edge (g, e) (since e is outside of $G(d)$: it is an "exit" that cannot generate loop). Finally, on a common edge such as (g, f), we have $\Delta(g) = \Delta(f)$.

A first interesting consequence of such properties is the increasing behavior of the Δ values along paths in $G(d)$ (old and common edges). Hence, sorting all Δ values for nodes in $G(d)$ yields a list of strictly increasing weight increments for each pair (l, d). Formally, we denote as $\Delta_i + \epsilon$, and call a *metric sequence*, the sorted union of $\Delta(n) + \epsilon$ for all nodes in $G(d)$. From an edge perspective, such a sequence ensures that each old edge (x, y) connecting an upstream node x to a downstream one y stops being used before a new, or transient (containing at least one new edge), subpath relinks y to x, thus creating a cycle. This sequence makes it possible to update upstream nodes strictly before downstream ones, or in the same convergence period if: 1) they share a common Δ value; or 2) if their Δ values only differ by $\epsilon (\Delta(y) = \Delta(x) + \epsilon)$.

Δ_i and $\Delta_i + 1$ being two consecutive Δ values in the metric sequence, a routing change occurs only once between $\Delta_i + \epsilon$ and Δ_{i+1}. Precisely, new edges start being used as the weight increment reaches Δ_{i+1}, but old ones used for $\Delta_i + \epsilon$ and Δ_{i+1} are the same. Thus, if a loop occurs during the transition from $\Delta_i + \epsilon$ to $\Delta_{i+1} + \epsilon$, it also occurs during the transition from Δ_{i+1} to $\Delta_{i+1} + \epsilon$. From Theorem 1, we know that no loop can occur while incrementing the link weight by ϵ, so that the transition from Δ_{i+1} to $\Delta_{i+1} + \epsilon$ is loop-free, as well as from

$\Delta_i + \epsilon$ to $\Delta_{i+1} + \epsilon$. Therefore, using the $\Delta_i + \epsilon$ metric sequence for a pair (l, d) provides a loop-free convergence toward d, given that the network converges between two consecutive metric increments.

Such properties on Δ values are also the basis of a new technique we propose in Section III. Since removing one edge from an elementary cycle is a sufficient condition to avoid it, we can deduce a necessary condition to avoid a transient loop. At least one node in the cycle has to be fully updated (it no longer uses its old edges) before the link weight is incremented by a value greater than or equal to the maximum Δ value of all nodes in the cycle. From this observation comes the notion of *metric interval*. For example, in Fig. 1, the elementary cycle of length 2 that may occur between nodes a and f could be avoided by incrementing the link weight by a value in $[\Delta(f) + \epsilon, \Delta(a) - \epsilon]$ before going further.

Two results arise from this section. First, there always exists a sequence of weight increases that provides a loop-free convergence. Second, this sequence can be minimized. The $\Delta_i + \epsilon$ sequence is not necessarily minimal because some transitions can still be removed if they are not necessary to avoid cycles. Such unnecessary transitions may result from the union of $\Delta_i + \epsilon$ of distinct sub-shortest paths upstream from the target link. Some transitions can be merged thanks to the flexibility offered by metric intervals if such an operation does not produce loops. Moreover, some $\Delta_i + \epsilon$ transitions resulting from the same sub-shortest path can also be removed by picking a single metric per elementary cycle. Section III presents the solution we propose to achieve an efficient minimization for all destinations.

III. Computing a Minimal Loop-Free Sequence

This section provides a detailed description on how we compute a global loop-free convergence sequence. Our goal is to efficiently compute a minimal metric sequence that is loop-free for all destinations. To this end, we compute a *reduced* loop-free sequence for each destination (Section III-A), merge all of them into a global sequence, and then prune this union to obtain a minimal sequence using a Greedy Sequential Walk (GSW) algorithm (Section III-B). This algorithm iteratively checks, for all destinations, if cycles occur in the union of two distinct RSPDAGs. The minimization process must be performed at the global level: Minimizing the merging of minimized destination oriented sequences does not necessarily lead to a global minimum. Instead, we will show that having, for each destination, the smallest Δ value of each potential cycle is sufficient to create a global sequence of minimum size.

A. Destination-Oriented Metric Sequences (DMSs)

First and foremost, let us introduce the notation D that gives the set of "affected destinations." This set contains all destination nodes in N that may be concerned by the weight change of the link (a, b). In practice, D is equal to all nodes $\in SPDAG(a)$ that have b as common root (first hop of the branch). Indeed, all $d \in N$ such that $\exists n, d | P(n, d) = (n, \ldots, a, b, \ldots, d)$ may be concerned by a metric change on (a, b). More formally, we have $D = \{d \in N | G(d) \neq \emptyset\}$. In our running example given in Figs. 1 and 2, considering the shutting down of the directed link (a, b), we have $D = \{d, e, b\}$, although we will only focus on the two first destinations for the sake of simplicity.

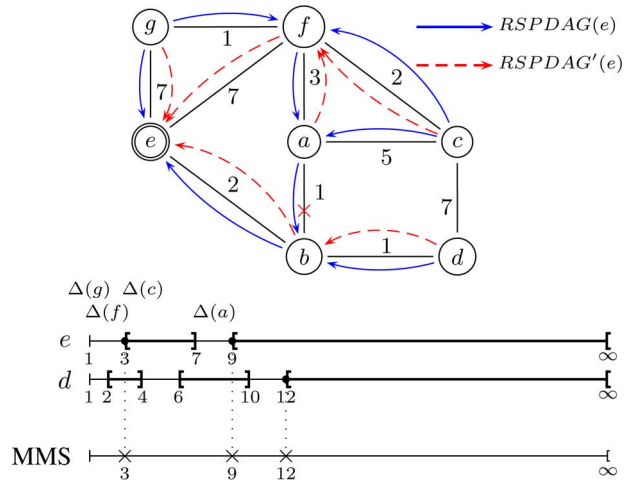


Fig. 2. Minimize a global metric sequence with GSW.

1) Computing Δ Values: As previously, let us focus on a given link (a, b), considering its initial and target weights, respectively m_i and m_t (as previously, we will focus on the case of shutting down a link such that m_t is equivalent to ∞), and a destination $d \in D$ initially reached through this link. As introduced in Section II, we know that there exists a sequence of $\Delta + \epsilon$ ensuring an incremental loop-free convergence. This sequence, named DMS, contains a sufficient set of weight increments within $[m_i, \infty[$. Each metric forces at least one node $\in G(d)$ to avoid its *old* paths going through (a, b) and so to use its *new* paths that do not contain the link (a, b).

To compute all Δ for a given destination d (see Algorithm 1) and then obtain DMS(d), we first need to compute the two RSPDAGs rooted at d with both the initial and target weights of the link (a, b) (lines 1 and 2 in Algorithm 1). For all source nodes in $G(d)$, a first costly approach would consist in inserting in $DMS(d)$ a new metric increment equal to $\Delta(s) + \epsilon$ (where $\Delta(s)$ is computed at line 4 of Algorithm 1). When the weight of the link (a, b) is set to $m_i + \Delta(s) + \epsilon$, router s would no longer use its *old* routes (via (a, b)), but only its *new* ones (avoiding (a, b)) to reach d. This value, $m_i + \Delta(s) + \epsilon$, is the first possible metric that forces s to avoid its old paths through (a, b) (without causing a loop). Note that a DMS may contain Δ metrics leading to transient ECMP states when there exists a couple of nodes n and m such that $\Delta(n) + \epsilon = \Delta(m)$.

Algorithm 1: Δ computation for destination d

1: Compute $RSPDAG(d)$
2: Compute $RSPDAG'(d)$
3: **for all** $s \in G(d)$ **do**
4: $\quad \Delta(s) \leftarrow C'(s) - C(s)$

As explained in Section II, such a complete DMS provides a loop-free convergence for each destination. However, the length of the resulting DMS may be large. In Section III-A.2, we focus on a specific subset of these metrics in order to limit the time

complexity of the global minimization stage. In our running example and considering destination e highlighted in Fig. 2, we will show why it is possible to only select metric $\Delta(f)+\epsilon$ instead of the complete sequence $\{\Delta(g)+\epsilon, \Delta(f)+\epsilon, \Delta(c)+\epsilon\}$.

2) Computing Reduced DMSs: While applying the DMS does lead to a loop-free convergence for a given destination, only a subset of the metrics are necessary to guarantee such a property: Only one intermediate metric for each elementary cycle is actually required. However, in order to preserve minimality at the global level, a specific metric has to be part of this sequence. This particular metric depends on whether we choose to avoid ECMP transient state using $\Delta+\epsilon$ or $\Delta-\epsilon$ values, being respectively the lowest or the greatest one. That is, to achieve our goals, a DMS containing at least the lowest metric of each elementary cycle is sufficient to ensure both loop-freeness and global minimality.

Algorithm 2 extracts the minimal metric of each elementary cycle from the DMS at a negligible cost. In return, it may produce false positives (i.e., extra values in the resulting sequence) when several subbranches of $G(d)$ are involved in the same elementary cycle. Since unnecessary metrics are pruned from the sequence at the global level, this algorithm focuses on pruning as many metrics as possible, without relying on any cycle detection or enumeration.

Algorithm 2: Compute a reduced $\mathrm{DMS(d)}$

```
1: for all v ∈ G(d) do
2:     for all u|∃(u,v) ∈ RSPDAG'(d) do
3:         If Δ(v) < Δ(u) then
4:             DMS(d)·insert(Δ(v) + ε)
```

Algorithm 2 has a linear cost and allows to insert a reduced subset of metrics that prevents loops and thus strongly limits the size of the DMS compared to an exhaustive Δ insertion. Lines 3 and 4 in Algorithm 2 allow to select all nodes that have a *new edge* pointing on them (in $RSPDAG'(d)$ such that the predecessor has a strictly greater Δ). Their associated Δ can be the minimal one of an elementary cycle.

Lemma 1: Algorithm 2 computes a sequence containing, at least, the minimal metric of each elementary cycle.

Lemma 2: Computing a minimal metric for each elementary cycle is sufficient to ensure a loop-free convergence.

These two lemmas, whose proofs are given in Appendix-B, justify our reduction approach. Looking at Fig. 2 (in this figure, our notations refer to destination e and link (a,b)), we notice that $G(e)$ includes nodes a, c, f, and g. Each of those nodes exhibits a $\Delta > 0$. However, only one of them, node f, has a new edge pointing onto it. Node f is the only one whose metric can prevent a loop. While node a is involved in the same (and unique) cycle as the one of f (f provides the minimal possible metric to avoid this cycle in the metric interval $[\Delta(f)+\epsilon, \Delta(a)-\epsilon]$), nodes c and g are not involved in any cycle, and their metrics are not required to avoid transient loops. If they were, we know from Properties 1–3 that they cannot provide the smallest metric of any cycle. We can ignore them to improve the efficiency of our global minimization.

When reducing DMS for all destinations, it generally comes out that sequences for some destinations become empty. Thus, the set of affected destinations D may be reduced to a new subset D_r. In practice, on real ISP networks, we usually have $|D_r| \ll |D|$. When a large set of affected destinations $d \in D$ concerns a limited set of source nodes (typically when $G(d)$ is small), it is likely that either all of them have their own rerouting next hop outside of $G(d)$, or they use the same rerouting node close to the link being removed. This large reduction comes from practical IP network design [23]. Indeed, for obvious redundancy reasons, operators generally deploy mesh patterns (such as triangles and squares) to make the network robust against link or router failures. We designed Algorithm 2 to be efficient on such patterns.

B. Global Metric Sequences (GMSs)

In practice, routers need to react to the change of a link weight by updating their FIBs for all affected destinations. Thus, computing a DMS for only one destination d is not sufficient to provide a complete solution. It only provides a loop-free convergence for packets toward d, but transient loops might still occur toward other destinations.

In this section, we show how to compute a valid and minimal sequence for all the destinations affected by a change. We call a valid global sequence a GMS. After minimizing it to the shortest valid possible global sequence, we call it a minimized global metric sequence (MMS). Our proposition to achieve this goal is based on a greedy algorithm consisting in merging DMSs obtained for each destination before minimizing the union thanks to a global GSW technique. We also develop efficient procedures scaling well with possibly long merged sequences and large D sets. Such improvements are detailed in Section III-B.2.

1) Merging the DMSs for All Destinations: Merging DMSs computed for each destination provides a valid loop-free GMS for all destinations affected by the change. More formally (the proof is given in the Appendix C), we have:

Lemma 3: The union of all DMSs for each destination in D_r results in a valid GMS for all destinations.

Such a basic merging does not necessarily provide an MMS. That is, applying this GMS for a given link does lead to a loop-free convergence, but it may contain unnecessary metrics. Indeed, a metric belonging to a given DMS may be removed if a metric from another sequence can play its role (as it can be also the case within a single DMS along distinct sub branches in $G(d)$). In the network illustrated in Figs. 1 (destination d) and 2 (destination e), picking a unique weight increase in $[3,4]$ is sufficient to avoid four elementary cycles: It avoids cycles c–f, a–f, and a–c–f for destination d and cycle a–f for destination e. However, it does not avoid cycle f–a for destination d. An additional weight increase must be picked to ensure global loop-freeness. We will show that using the sequence $\{3,9,12\}$ is one possible MMS to avoid any transient loop. Compared to the merged GMS $\{2,3,6,9,12\}$ resulting from the union of reduced $DMS(d)$ and $DMS(e)$, it allows for saving two metric changes and thus proportionally reduces the convergence time.

In order to minimize the length of merged GMS, we use a greedy approach. As it is applied on the set D_r, each cycle detection, along with RSPDAG computing and merging, has to be

performed for each destination $d \in D_r$. It may potentially result in a time-consuming algorithm. In Section III-B.2, we introduce an efficient merging and checking algorithm related to our GSW technique. Indeed, the core step of our GSW algorithm is to iteratively check for loops for a given pair of weight increments in order to avoid them with a minimal number of metrics. Note that the GMS we obtain after using Algorithm 2 is a subset of the one containing all possible Δ values. We denote it GMS_r. It contains at least all minimal metrics of each elementary cycle for each destination in D_r. In practice, in the same vein that we have $|D_r| \ll |D|$, we also have $|GMS_r| \ll |GMS|$.

2) Efficient Merging and Checking Algorithm: A first way to limit the required time complexity is to efficiently compute intermediate RSPDAGs. While cycle detection is a linear operation in $O(|N|)$, computing an RSPDAG is polynomial. Its complexity is in $|N|\log(|N|)+|E|$ with a Fibonacci Heap [24]. Then, computing $O(|N|)$ RSPDAGs per metric change can become quickly expensive. To limit such a complexity, we reuse previously computed Δ metrics. Indeed, intermediate RSPDAGs are composed of edges in either $RSPDAG(d)$ or $RSPDAG'(d)$. Either a node n still uses its old outgoing edges (until reaching $\Delta(n)$), or it uses only its new outgoing edges (after having exceeded $\Delta(n) + \epsilon$). Thus, it is not necessary to recompute each intermediate RSPDAG from scratch. The computation of $RSPDAG(d)$ and $RSPDAG'(d)$ is sufficient. In practice, rather than considering all intermediate RSPDAG, we directly compute the merging of a considered couple of RSPDAG according to the GSW walk. This merging allows for detecting the presence of cycles.

Another improvement consists in limiting cycle detection tests to a subset of currently affected destinations. When testing a pair of metrics to ensure a loop-free transition, it is not necessary to verify the absence of loop for a destination that cannot be affected by such a transition. Considering a destination d, and a given transition checked by GSW, if either the intermediate RSPDAG for d is equal to $RSPDAG(d)$ (no metric for d is reached), or if its intermediate RSPDAG is equal to $RSPDAG'(d)$ (all metrics for d have been exceeded), then destination d cannot generate a loop for such a transition. This basic idea allows us to save a significant amount of time because it limits the detection of cycles to a possibly very limited subset of destinations (in particular when DMS have metrics in disjoint intervals).

Algorithm 3 implements both of these basic procedures, making it possible to check in linear time whether or not a metric transition is loop-free for a given destination. Our GSW algorithm technique uses such improvements to limit its time complexity. Function CHECKFUSION takes three arguments: d, the considered destination, and $m1, m2$, such that $m1 < m2$, the pair of metrics corresponding to the transition that is being tested. Line 2 eliminates destinations that the current transition cannot affect. That is, if the transition is outside the metric sequence of a given destination, this destination is ignored. Then, lines 4–10 compute a graph containing all possible routes that could appear during the transition, using Δ values rather than computing them from scratch. Eventually, line 11 checks whether a cycle exists in this graph. The function returns `TRUE` if the transition from m_1 to m_2 is loop-free for destination d (or `FALSE` if a loop occurs).

Algorithm 3: Merge and check for loops only if necessary

```
1:  function CHECKFUSION (d, m1, m2) GraphF ← ∅
2:      if m1 ≥ max(DMS(d)) or
        m2 ≤ min(DMS(d)) then
3:          return true
4:      for all n ∈ G(d) do
5:          if m1 ≤ Δ(n) then
6:              for all v|∃(n, v) ∈ RSPDAG(d) do
7:                  Insert (n, v) in F
8:          if m2 ≥ Δ(n) then
9:              for all v|∃(n, v) ∈ RSPDAG'(d) do
10:                 Insert (n, v) in F
11:     return ! checkLoop(F)
```

Let us illustrate these behaviors on the running example given in Fig. 2, as we check the metric transition from 10 to 12. First, since 10 is strictly greater than the greatest Δ value in the metric sequence for destination e, this destination is ignored. Then, for destination d, the merged graph is created by adding the old outgoing edge of node a, whose Δ value is greater than m_1, and new outgoing edges of all nodes since their Δ values are lower than m_2. Lastly, a cycle detection is performed over this graph, proving that the transition is loop-free.

3) Minimizing a GMS to Compute an MMS: To minimize a GMS into an MMS, we propose an algorithm called GSW (see Algorithm 4). We aim at discovering the largest metric increment that can be done from one necessary metric to another without causing loops. Note that the set of minimal sequences is a union of metric interval sequences. There exist several ways to explore them and then to optimally resolve the problem. Therefore, several greedy algorithms might be used: The sequence can either be explored sequentially from left to right, or opportunistically (largest increase first to possibly skip many metrics), or even with a dichotomic approach. All of them lead to sequences of minimal length, but which do not necessarily contain the same metrics. Computing times also vary depending on how distant the GMS is from an MMS. The sequential approach is more efficient when applied on a GMS that is already close to an MMS, while the opportunistic one is better if a large number of metrics has to be pruned. Eventually, the dichotomic approach would likely produce the best results for random graphs. Since we observed on real ISP networks that the merging of reduced DMS is often close to an MMS, we choose to use the first and simplest sequential approach. When applying one of the two other variants on a GMS that is already an MMS, it requires a polynomial-time search instead of a linear one with GSW.

Algorithm 4: Greedy Sequential Walk (GSW)

```
1: m ← m1 ← mi
2: for m2 = min(GMS) → max(GMS) do
3:     for all d ∈ Dr do
4:         if ! CHECKFUSION(m1, m2, d) then
5:             MMS· insert(m)
6:             m1 ← m
7:             break
8:     m ← m2
```

Basically, our GSW algorithm starts from the initial metric and sequentially constructs a minimal loop-free sequence. At each step, metric transitions from the last element in this sequence to the next possible values in the GMS are checked. As soon as a cycle is detected, the previous value is added to the sequence since it is the largest one to which the transition remains loop-free. Then, the procedure is restarted until the target metric is reached. Note that, practically, the first transition check of each step can be skipped, as we know that the transition between two consecutive values in the GMS is always loop-free.

Given a GMS $\{m_i = m_1, m_2, \ldots, m_j, \ldots, m_t\}$, GSW selects the largest metric m in that sequence, such that increasing $w(a,b)$ directly from m_i to m does not lead to forwarding loops. To do so we verify the absence of loop in the merging of $RSPDAGs_i$ and $RSPDAGs_m$, for each destination, by calling Algorithm 3. Metric m is selected as the largest possible metric if a cycle is detected at the subsequent metric (lines 4 and 5). Then, we restart the same procedure with $m_1 = m$ (line 6) until a new cycle is detected or m_t is reached. The walk is sequential (line 2). The m_2 values are picked in the GMS following an increasing order, and each value is checked only once.

Let us illustrate the operation of Algorithm 4 in Fig. 2 considering only destinations d and e. The input provided to GSW is the merged sequence $\{2, 3, 6, 9, 12\}$. When testing RSPDAG pairs $RSPDAG_i, RSPDAG_j$, for pairs $(i,j) = (1,2)$ and $(1,3)$, no loop appears. However, for the pair $(1,6)$, a cycle of length two between c and f occurs for destination d. Thus, GSW validates the transition $(1,3)$, the last correct state. It restarts with transition $(3,6)$ and finally validates $(3,9)$ because the largest transition $(3,12)$ implies a loop between f and a for destination d. The final MMS consists then in applying successive weight updates in $\{3, 9, 12\}$.

Now, let us demonstrate why our GSW algorithm provides an MMS. Note that a general proof of optimality for the greedy approach is given in [21]. A core step of the reasoning is based on the following lemma (whose proof is given in Appendix-C).

Lemma 4: If a metric transition for a link (a,b) from m to n, with $m < n$, is not loop-free, then we have the following.

- A metric transition from m to o for this link, with $o > n$, is not loop-free.
- A metric transition from k to n for this link, with $k < m$, is not loop-free.

From this lemma, we can prove that the sequence obtained with the GSW algorithm is an MMS.

Theorem 2: Using the GSW algorithm on a GMS containing all minimal metrics of each elementary cycle for all affected destinations provides an MMS.

Proof of Theorem 2: Let $s = \{\ldots, m_j, m_{j+1} \ldots\}$ be the sequence computed with GSW on GMS_r. If there exists a shorter GMS than s (this sequence may pick any metrics in $[m_i, m_t]$), say $s' = \{\ldots, m'_k, m'_{k+1} \ldots\}$, then at some step, s' differs from s. If s' is smaller than s, there necessarily exists $m_j \in s$ and $m'_k \in s'$ such that $m'_k \leq m_j$ and $m'_{k+1} > m_{j+1}$. Let us note $m_l > m_{j+1}$ the minimal metric of the next elementary cycle for any destination in D_r (by next, we mean the first one after m_{j+1} that GSW has not proven loop-free on GMS_r). If $m'_{k+1} \geq m_l$, it implies, from Lemma 4, that the transition from m_j to m_l is also loop-free. Then, by definition of our GSW algorithm, we have $m_{j+1} \geq m_l$ (which is contradictory with previous statements). Eventually, picking any metric $m'_{k+1} \in]m_{j+1}, m_l[$ cannot result in a shorter sequence since the metric m'_{k+1} cannot avoid any additional elementary cycle. It just enlarges the distance between the two subsequent metrics. ■

Several MMS solutions may exist. Depending on metrics given in the initial GMS and the algorithm itself, several MMSs of the same length may exist. Finally, the worst-case time complexity of the overall procedure is polynomial, in $O(|N|^3 \times |E|)$ for each MMS computation. It may appear high, but, as shown in Section IV, typical computing times are very limited in practice: The amount of input metrics in the GMS generally verifies $|GMS| \ll |N|^2$ on real use-case networking graphs. Moreover, we are currently investigating algorithmic improvements that would significantly reduce this worst-case complexity.

IV. Evaluation

This section aims at evaluating the typical MMS length and the time required to compute them on real IP networks. First, we present the topologies we used to evaluate the applicability of our algorithms. We then present the results of our evaluation, observing the reasonable length of the MMS (Section IV-B.1), as well as the efficiency of our optimizations on computing times (Sections IV-B.2 and IV-B.3). Instead of studying worst-case time complexity and theoretical maximum MMS length, we rather focus on typical order of magnitude of such quantity on real ISPs. Note that the memory consumption of our approach is only a few megabytes even for large networks, and we do not evaluate it in details in the following.

A. Environment Setup

Table II summarizes the main characteristics of our evaluation topologies. $|N|$ and $|E|$ respectively represent the number of nodes and edges in the graph. $\not\phi$ stands for the diameter, d_m is the maximum degree, and w gives the minimum, maximum, and cardinality values of the set of weights. In particular, we provide information about the range and number of IGP weights in each network. Networks 1 and 2 of Table II are two well-known small ISPs whose graphs and link weights are freely available (see respectively [25] and [22]). Networks 3–8 of Table II are Rocketfuel inferred topologies obtained with traceroute campaigns [26]. This set of IP graphs comes with inferred IGP weights. Networks 9–14 are real ISPs that we anonymized for confidentiality reasons (number of nodes and edges are also rounded).

Our algorithms are implemented in C for performance purposes and are freely available.[1] Particular attention has been given to the performances of our algorithms since the low time complexity of our methods (in real use cases rather than on generic graphs) is one of the main results presented in this paper. Computation times given in Section IV-B have been obtained on commodity hardware with a CPU clock rate of 2.53 GHz.

[1]https://sourceforge.net/projects/metric-incr/

TABLE II
MAIN GRAPH PROPERTIES OF OUR SET OF EVALUATION NETWORKS

ID	Network	$\|N\|$	$\|E\|$	$\varnothing$	d_m	w	
1	Abilene	11	28	5	3	$[233, 2095]$	(14)
2	Geant	22	72	4	6	$[1, 20050]$	(18)
3	Exodus	79	294	10	12	$[1, 22]$	(17)
4	Ebone	87	322	11	11	$[1, 16]$	(14)
5	Telstra	108	306	8	18	$[1, 7]$	(6)
6	AboveNet	141	748	8	20	$[1, 20]$	(14)
7	Tiscali	161	656	10	29	$[1, 22]$	(20)
8	Sprint	315	1944	10	45	$[1, 16]$	(15)
9	ISP 1	25	55	6	6	$[1, 11]$	(4)
10	ISP 2	55	195	5	20	$[10, 50000]$	(8)
11	ISP 3	110	340	11	8	$[1, 9999]$	(32)
12	ISP 4	140	410	13	9	$[1, 9999]$	(32)
13	ISP 5	210	785	13	14	$[1, 66666]$	(55)
14	ISP 6	1170	7240	12	56	$[1, 100010]$	(105)

TABLE III
MMS LENGTH DISTRIBUTIONS

ID	MMS length distribution (%)					Max.
	$\|MMS\| = 0$	≤ 1	≤ 2	≤ 5	≤ 10	$\|MMS\|$
1	32.14	75.00	100	100	100	2
2	54.17	87.50	95.83	100	100	5
3	52.38	85.71	93.20	96.60	100	9
4	53.42	87.27	95.34	100	100	5
5	76.47	98.04	100	100	100	2
6	65.78	97.99	99.47	100	100	5
7	57.01	89.18	95.27	99.70	100	6
8	84.10	97.48	98.97	99.95	100	6
9	58.93	66.07	87.50	100	100	4
10	70.10	81.44	95.36	100	100	3
11	60.59	77.65	85.59	95.88	99.71	11
12	66.59	81.22	87.80	96.59	99.76	11
13	50.32	79.31	85.95	92.72	97.57	39
14	87.79	95.81	97.07	98.12	98.99	61

TABLE IV
MMS COMPUTING TIME DISTRIBUTIONS (ms)

ID	Link (percentile)			Router[2] (percentile)		
	50^{th}	80^{th}	100^{th}	50^{th}	80^{th}	100^{th}
1	0.03	0.05	0.08	0.08	0.10	0.13
2	0.05	0.09	0.23	0.18	0.21	0.28
3	1.02	1.85	6.21	3.98	5.33	8.95
4	0.52	2.94	5.89	4.46	6.02	8.87
5	0.67	4.53	6.16	4.65	5.79	8.34
6	0.78	5.80	12.89	14.40	17.77	32.67
7	1.51	10.55	22.14	16.24	22.51	39.63
8	6.19	34.60	176.68	104.60	124.26	518.01
9	0.15	0.32	0.35	0.33	0.38	0.47
10	0.09	1.07	1.79	1.22	1.74	2.03
11	1.15	4.94	6.03	6.06	7.51	11.02
12	1.79	8.87	10.16	9.59	11.79	18.04
13	3.49	19.34	98.88	30.16	39.38	101.64
14	2.78	993.53	*3.4 s*	*2.4 s*	*3.1 s*	*5.3 s*

B. Results

Tables III and IV respectively give an overview of MMS length and computing time observed on our set of evaluation networks. As in previous sections, we focus here on the case of a link removal that produces the largest sequences and requires worst computing times. Note that when removing a link (both directions of an edge), one has to compute and apply two MMSs, one for each direction. However, the two MMSs are independent and can be applied in any order.

1) Minimal Metric Sequence Length: Interestingly, the size of global sequences remains small in most of the studied cases. For all networks except Abilene, we observe that for more than half of the links, no intermediate metric is needed, as the size of the MMS is equal to 0. More specifically, between 12% (for ISP6) and 68% (for Abilene) of links require at least one intermediate metric to avoid transient forwarding loops during the convergence. We notice that for more than 90% of the cases, less than five intermediate metrics (before injecting the target one) are needed to ensure a loop-free convergence in case of a link removal. If one considers that increasing the convergence period by a factor of three is an upper bound (MMS length should be lower than 2), we observe that in at least 85% of link shutdowns for all networks, we verify such a requirement. In Table III, we can also notice that some graphs are less favorable than others. For example, in ISP3 and ISP5, almost 15% of the links require longer sequences while for ISP2 and ISP6 this proportion is lower than 5%.

For all networks, the MMS length distribution follows a power law such that the number of links requiring very large MMS is very limited. For the worst-case topology (ISP5) in terms of sequence length distribution, less than 3% of the links require sequences containing more than 10 intermediate metrics. In practice, except for ISP5 and ISP6 (with a maximum sequence size of 39 and 60), the maximum sequence size is around 10 metrics, and such sequences only concern a very small number of links. Those large sequences arise for several reasons: The dimension of the considered graph is a first factor as well as the number of distinct IGP weights. For instance, for a link that is used to reach many destinations and such that DMS are disjoint in terms of metrics (there are few intersections between metrics intervals), GSW will have little effect if there exists a cycle related to each metric interval. For a given destination d, the quantity of elementary cycles in the merging of $RSPDAG(d)$ and $RSPDAG'(d)$ impacts the number of necessary metrics: For example, if all nodes need to go backward on a ring network to avoid a failed link, it will produce a number of cycles of length 2 proportional to the size of the ring.

Although most of our sequences have a very reasonable size, it could be interesting to devise smart ways to cut too long sequences. This is out of the scope of this paper and is left for further work.

2) MMS Computing Time: We focus here on the time required to compute minimal sequences—our goal being to emphasize the feasibility of computing such sequences on common router architectures. We will show that computing MMS is not expensive, even for large graphs and in non-favorable cases. Since computing times obtained with our implementation are very low (and especially for small networks), instead of providing average values, Table IV gives computing time percentiles including worst-case results.

We can notice that for all topologies, except ISP6, the maximum computation time for shutting down a directed link is

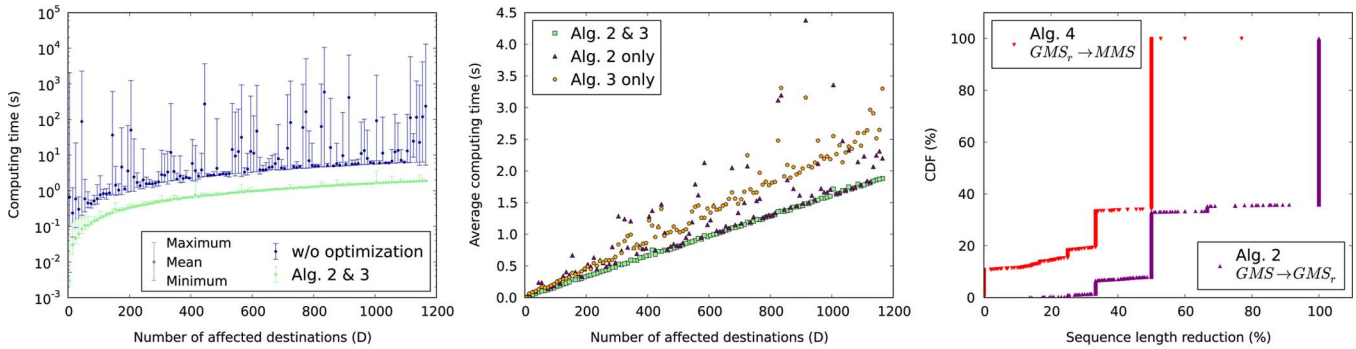


Fig. 3. Deeper analysis of ISP6: computing times, where the savings are. (a) Global improvements. (b) Individual improvements. (c) Length reduction.

below 200 ms, and about 500 ms for a router.[2] ISP6 being by far the largest network, we observe a worst case of 2.5 s for an edge and 5.3 s for all outgoing interfaces of a router. It seems that, on real IP networks, worst-case links are not concentrated on large degree routers. A core router having a large number of links generally benefits from numerous local rerouting options (such links "protect" each other among the set of destinations and thus do not require large MMSs and large computation times). These results raise the opportunity of letting routers compute metric sequence on their own. It can be performed either on the fly upon maintenance (for a given link) or as a low-priority background process (for all outgoing links of a given router).

Generally speaking, we observe that the overall minimization process does not cost too much: Reducing the merged sequence before minimization (Algorithm 2) and avoiding useless tests and RSPDAG computation (Algorithm 3) allows to limit the cost of the greedy minimization (GSW, Algorithm 4). The DMS reduction process comes at almost no cost, while it divides by at least a factor of three the time required for the minimization phase. This observation is particularly interesting as it highlights that, in practice, the only costly phase is the computation of the Δ's (which is not avoidable to discover potential forwarding loops). Note that computing the metric sequences for all links requires less than a second for small- and medium-sized networks, while it is about 1 h on ISP6. This last result might look huge, however dealing with total computation time is only relevant in the case of a server performing an exhaustive analysis of a topology. In such case, computation can be easily parallelized across multiple cores, thus significantly reducing this duration.

3) More Detailed Analysis of the Worst Computing Time Case: To better understand the impact of our implementation improvements on MMS computation, we plot in Fig. 3 results specific to ISP6, our largest case-study. This is the only network with both large MMS and computing times. In practice, most link-state IP networks are smaller than this one and provide more favorable results. ISP6 is one of the largest ISPs today.

First, in Fig. 3(a) we plot the computing times as a function of the number of affected destinations. Before interpreting such a plot, note that for more than half of the links, the number of affected destinations is equal to 1, while it tends to be uniform for larger numbers of affected destinations. On Fig. 3(a), we can start by noticing that the MMS computing time linearly depends on D, while the size of the initial sequence has a limited impact. Green dots correspond to our approach, while blue ones show computing times without our improvements. We observe that the time saving is particularly relevant for links used to reach many destinations (the scale is logarithmic). For some links, the computing time is divided by a factor of 50. Central links that are used to reach a large part of the network are more likely to be "expensive." When minimizing a sequence, for each metric, it is necessary to verify the absence of loops for all affected destinations. This is the key factor in the overall computation time. Thus, removing a metric and, most of all, focusing only on destinations in D_r allow to avoid costly tests and hence drastically reduce computation time. Note that, on average, D_r is significantly lower than D. This explains a large part of the gain obtained by Algorithm 2.

To better understand the benefits of our optimizations, we plot in Fig. 3(b) the computing time obtained with either Algorithm 2 or 3. First, we can notice that Algorithm 2 provides the largest gain for costly links. Moreover, it is interesting to observe that when an improvement fails to strongly reduce the computing time, the other succeeds to limit it. Indeed, when Algorithm 2 is not able to reduce the GMS, it is even more useful to use Algorithm 3. Our two improvements are dependent: The more metrics we test, the more Algorithm 3 is relevant and reciprocally. We also observe that the behavior of Algorithm 3 is more linear than the one of Algorithm 2. Indeed, while the efficiency of Algorithm 2 strongly depends on underlying network patterns (the redundancy of each specific part of the network), the efficiency of Algorithm 3 is almost systematic: Avoiding useless tests and, in any cases, incrementally computing RSPDAG strongly improve the efficiency of our approach.

Finally, Fig. 3(c) gives details about the GMS reduction and minimization. We plot the cumulative distribution function of the two stages of the sequences reduction (Algorithms 2 and 4). The gain in sequence reduction is plotted relatively to the respective previous stage. First, this figure shows that our pruning phase strongly reduces GMS lengths. In more than 60% of the cases, GMSs become empty (the large purple bar on the right). Algorithm 2, by removing a large number of useless metrics, allows to set D_r to 0 for more than half of the links. For approximately 30% of the links (ignoring the ones resulting in

[2]These columns provide the time distribution required for a router to compute independent MMS for each of its outgoing links. They illustrate the time that a router would spend at the computation of the metric sequences of its own links

empty sequences), Algorithm 2 reduces the initial GMS by a factor larger than 2. Moreover, even after having removed 60% of the initial GMS, among the remaining GMS_r, we can notice that a significant subset of them ($\approx 10\%$) is already equal to an MMS (the small red bar on the left). It comes that the remainder of GMS_r is still reduced by a maximum factor of 2 except for three links. This reduction mostly concerns very short GMS_r (generally for $|GMS_r| = 2$). These are sequences that do not require a long computing time anyway. Longer MMSs with many affected destinations take advantage of a large number of metrics being pruned from DMSs with Algorithm 2. Since Algorithm 3 performs a cycle detection for each destination and intermediate metric in the GMS, the more destinations are affected, the greater the savings are for each metric pruned from a DMS. Such a reduction thus generally results in a significant time saving for worst-case links.

From our experiments performed on real and inferred network use cases, we can conclude that our minimization approach generally produces sequences of limited size in a reasonable time. These results encourage us to consider the use of these techniques on production routers. Also, since most sequences only require two additional metrics, the convergence time is only increased to the same extent. The benefits of avoiding transient loops seems sufficient to balance such a little disadvantage in case of planned operations.

V. Conclusion

The distributed nature of link-state routing protocols implies that transient forwarding loops can occur after each topological change. This is a problem in networks that support mission-critical applications with stringent SLAs. This paper demonstrates that by splitting any link weight change in a series of metrics increments/decrements, it is possible to avoid all these forwarding loops without any change to the routing protocols.

The two key concerns for the applicability of our proposed techniques are the number of metrics increments/decrements that are required for a topological change and the time to compute them. Our simulations on various ISP topologies show that the metric sequences remain small. For most link shutdown scenarios, the sequences are short. Across all topologies, 85% of the links can be shut down with less than three metric changes. Concerning the computation time for the metrics sequence, previous work showed that this computation could be included in network management or traffic engineering tools [20]. Thanks to the improvements proposed in this paper, it is now possible to compute the metrics sequence directly on the routers. Our evaluations shows that in most networks, the metric sequence for a link shutdown can be computed in a few tens or hundreds of milliseconds on commodity hardware. Only a few cases in one of the largest ISP network require a few seconds of computation time.

Appendix A

A. Proofs From Section II

Proof of Property 1: On the one hand, if node x is not updated and continues to use its old neighbor y, we have by definition $C(x) = C(y) + w(x, y)$. On the other hand, if node x does not use y anymore once updated we have $C'(x) < C'(y) + w(x, y)$.

Thus, $\Delta(x) = C'(x) - C(x) = C'(x) - (C(y) + w(x, y)) < C'(y) + w(x, y) - C(y) - w(x, y) = \Delta(y)$. ■

Proof of Property 2: When node x updates its path toward d and decides to go through a new neighbor y, we have by definition $C'(x) = w(x, y) + C'(y)$. We also have $C(x) < C(y) + w(x, y)$: x was not using y as a next hop toward d before the update.

Thus, $\Delta(x) = C'(x) - C(x) = w(x, y) + C'(y) - C(x) > C'(y) - (C(y) + w(x, y)) + w(x, y) = \Delta(y)$. ■

Proof of Property 3: When node x updates its path toward d and still uses its old neighbor y toward d, we have by definition $C'(x) = w(x, y) + C'(y)$. We also have $C(x) = C(y) + w(x, y)$: x was using y as a next hop toward d before the update.

Thus, $\Delta(x) = C'(x) - C(x) = w(x, y) + C'(y) - C(x) = C'(y) - (C(y) + w(x, y)) + w(x, y) = \Delta(y)$.

B. Proofs From Section III-A

Proof of Lemma 1: Let us consider an elementary cycle $c = \{n_1, n_2, \ldots, n_i, \ldots, n_p = n_1\}$ in the graph resulting from the merging of the initial and target RSPDAGs toward a destination d.

Given a node n_i and its direct predecessor n_{i-1} in c, we know from Properties 1–3 (in Section II-D) that if the edge (n_{i-1}, n_i) does not belong to $RSPDAG_t(d)$, then $\Delta(n_{i-1}) < \Delta(n_i)$ so that $\Delta(n_i)$ cannot be the minimal metric of c. Consequently, our algorithm focuses on edges that are used after the target metric change. Moreover, if $\Delta(n_{i-1}) = \Delta(n_i)$, we do not need to add $\Delta(n_i)$ either. Indeed, $\Delta(n_i)$ is the minimal Δ value of c if and only if $\Delta(n_{i-1})$ is. Eventually, we only add a metric $\Delta(n_i)$ to the sequence if its value is strictly lower than $\Delta(n_{i-1})$. Since there exists at least one edge in c that does not belong to $RSPDAG_i(d)$ (otherwise c cannot be a cycle), we know that there exists at least one node n_i in c such that $\Delta(n_i) < \Delta(n_{i-1})$. ■

Proof of Lemma 2: Let us reconsider the elementary cycle c used in the previous proof and denote as $e = (n_i, n_{i+1})$ an edge in c that becomes unused as the link metric increases. As soon as such an edge e becomes unused, c no longer exists. Therefore, using any metric of a node in c except the largest one is sufficient to avoid the potential transient loop corresponding to c. In particular, using the metric $\min_{\forall n \in c}(\Delta(n))$ is the first possible increment that allows to avoid c. From the proof of Lemma 1, we know that the associated edge $e \in c$ that turns first unused then verifies $\Delta(n_i) = \min_{\forall n \in c}(\Delta(n))$.

Increasing the weight of link (a, b) by $\Delta(n_i)$ removes at least the edge e from the forwarding plane. Thus, c neither appears in the metric transition from $\Delta(n_i) + m_i$ to m_t nor in the metric transition from m_i to $m_i + \Delta(n_i)$. The loop corresponding to c appears if and only if a metric greater or equal to $\max_{\forall n \in c}(\Delta(n))$ is injected while all other metrics in c are skipped. Consequently, generalizing the same reasoning for all elementary cycles, we know that considering only the minimal metric of each elementary cycle results in a loop-free convergence. ■

C. Proofs From Section III-B

Proof of Lemma 3: Given a DMS for link (a, b) and a destination d, $\{m_i, \ldots, m_j, m_k, m_l, \ldots, m_t\}$. We need to prove that adding metrics in that sequence also gives a loop-free metric sequence for destination d.

Let us consider the extended sequence $\{m_i, \ldots, m_j, m_s, m_k, m_l, \ldots, m_t\}$, with $m_s \in]m_j, m_k[$. As m_j and m_k are consecutive metrics in the initial MKMS, we know that $RSPDAG_j(d) \bigcup RSPDAG_k(d)$ does not contain cycles. We also know that $RSPDAG_j(d) \bigcup RSPDAG_s(d) \subset RSPDAG_j(d) \bigcup RSPDAG_k(d)$. Since k is greater than s, then all paths in $RSPDAG_s(d)$ not via (a, b) are also in $RSPDAG_k(d)$. Since j is lower than s and k, paths that are not anymore in $RSPDAG_k(d)$ (but the ones still going through (a, b) in $RSPDAG_s(d)$) are in $RSPDAG_j(d)$. This implies that $RSPDAG_j(d) \bigcup RSPDAG_s(d)$ does not contain any cycle either.

The same reasoning can be used to show that the merging of $RSPDAG_s(d)$ and $RSPDAG_k(d)$ is cycle-free, so that the metric sequence that adds s is still loop-free for destination d. Thus, inserting an arbitrary number of metrics within a loop-free metric sequence for a given destination preserves the loop-freeness property. ∎

Proof of Lemma 4: If the transition from metric m to n is not loop-free for destination d, then there is a cycle in the merging of $RSPDAG_m(d)$ and $RSPDAG_n(d)$. The first proposition is true if there exists a cycle in $RSPDAG_m(d) \bigcup RSPDAG_o(d)$. When setting the weight of the link (a, b) from m to n, some nodes may not use (a, b) toward d anymore, which leads to the possibility of a loop. Let us denote this subset of nodes by $\mathcal{N}$. If $w(a, b)$ is set to a step $o > n$, instead of being set to step n, each node in $\mathcal{N}$ still uses the same set of paths as in the step n. Thus, paths from each node in $\mathcal{N}$ included in $RSPDAG_n(d)$ are the same paths as in $RSPDAG_o(d)$. When merging $RSPDAG_o(d)$ with $RSPDAG_m(d)$, we obtain at least the same cycles as when merging $RSPDAG_m(d)$ with $RSPDAG_n(d)$. Hence, a loop also occurs in the transition $m \rightarrow o$.

The same reasoning can be applied to prove the second proposition. Then, obviously, we also know that the transition $k \rightarrow o$ is not loop-free either. ∎

REFERENCES

[1] J. Martin and A. Nilsson, "On service level agreements for IP networks," in *Proc. IEEE INFOCOM*, New York, NY, USA, Jun. 2002, vol. 2, pp. 855–863.

[2] AT&T, Dallas, TX, USA, "AT&T managed Internet service," 2007 [Online]. Available: http://www.att.com/gen/general?pid=6622

[3] Sprint NEXTEL, Overland Park, KS, USA, "Sprint NEXTEL service level agreements," 2007 [Online]. Available: http://www.sprint.com/business/support/serviceLevelAgreements.html

[4] NTT America, New York, NY, USA, "NTT communications global IP network service level agreement," 2013 [Online]. Available: http://www.us.ntt.net/support/sla/network/

[5] H. Pucha, Y. Zhang, Z. M. Mao, and Y. C. Hu, "Understanding network delay changes caused by routing events," *Perform. Eval. Rev.*, vol. 35, no. 1, pp. 73–84, Jun. 2007.

[6] Y. Zhang, Z. M. Mao, and J. Wang, "A framework for measuring and predicting the impact of routing changes," in *Proc. IEEE INFOCOM*, Anchorage, AK, USA, May 2007, pp. 339–347.

[7] F. Wang, Z. M. Mao, J. Wang, L. Gao, and R. Bush, "A measurement study on the impact of routing events on end-to-end internet path performance," *Comput. Commun. Rev.*, vol. 36, no. 4, pp. 375–386, Aug. 2006.

[8] P. Pongpaibool, R. Doverspike, M. Roughan, and J. Gottlieb, "Handling IP traffic surges via optical layer reconfiguration," in *Proc. Opt. Fiber Commun. Conf. Exhib.*, Anaheim, CA, USA, Mar. 2002, pp. 427–428.

[9] 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 Trans. Netw.*, vol. 16, no. 4, pp. 749–762, Aug. 2008.

[10] N. Dubois, B. Fondeviole, and N. Michel, "Fast convergence project," presented at the RIPE 47, Jan. 2004.

[11] B. Fortz and M. Thorup, "Optimizing OSPF/IS-IS weights in a changing world," *IEEE J. Sel. Areas Commun.*, vol. 20, no. 4, pp. 756–767, May 2002.

[12] P. Pan, G. Swallow, and A. Atlas, "Fast reroute extensions to RSVP-TE for LSP tunnels," IETF, RFC 4090, May 2005.

[13] A. Atlas and A. Zinin, "Basic specification for IP fast reroute: Loop-free alternates," IETF, RFC 5286, Sep. 2008.

[14] A. Kvalbein, A. F. Hansen, T. Čičic, S. Gjessing, and O. Lysne, "Multiple routing configurations for fast IP network recovery," *IEEE/ACM Trans. Netw.*, vol. 17, no. 2, pp. 473–486, Apr. 2009.

[15] U. Hengartner, S. Moon, R. Mortier, and C. Diot, "Detection and analysis of routing loops in packet traces," in *Proc. 2nd ACM SIGCOMM Workshop Internet Meas.*, Marseille, France, Nov. 2002, pp. 107–112.

[16] M. Goyal, M. Soperi, E. Baccelli, G. Choudhury, A. Shaikh, H. Hosseini, and K. Trivedi, "Improving convergence speed and scalability in ospf: A survey," *IEEE Commun. Surveys Tut.*, vol. 14, no. 2, pp. 443–463, 2nd Quart., 2012.

[17] M. Shand and S. Bryant, "A framework for loop-free convergence," IETF, RFC 5715, Jan. 2010.

[18] P. Francois and O. Bonaventure, "Avoiding transient loops during the convergence of link-state routing protocols," *IEEE/ACM Trans. Netw.*, vol. 15, no. 6, pp. 1280–1292, Dec. 2007.

[19] R. Teixeira and J. Rexford, "Managing routing disruptions in Internet service provider networks," *IEEE Commun. Mag.*, vol. 44, no. 3, pp. 160–165, Mar. 2006.

[20] P. Francois, M. Shand, and O. Bonaventure, "Disruption free topology reconfiguration in OSPF networks," in *Proc. IEEE INFOCOM*, Anchorage, AK, USA, May 2007, pp. 89–97.

[21] H. Ito, K. Iwama, Y. Okabe, and T. Yoshihiro, "Avoiding routing loops on the internet," *Theory Comput. Syst.*, vol. 36, pp. 597–609, 2003.

[22] GEANT, Cambridge, U.K., "GEANT network topology," 2012 [Online]. Available: http://www.geant.net/Network/The_Network/Pages/Network-Topology.aspx

[23] C. Filsfils, P. Francois, M. Shand, B. Decraene, J. Uttaro, N. Leymann, and M. Horneffer, "Loop-free alternate (LFA) applicability in service provider (SP) networks," Internet Engineering Task Force, RFC 6571 (Informational), Jun. 2012 [Online]. Available: http://www.ietf.org/rfc/rfc6571.txt

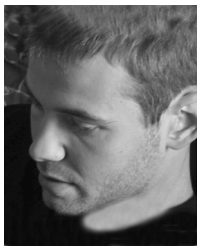
[24] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson, *Introduction to Algorithms*. New York, NY, USA: McGraw-Hill, 2001.

[25] Internet2, Ann Arbor, MI, USA, "Abilene network topology," [Online]. Available: http://www.internet2.edu/info/#maps

[26] R. Mahajan, N. Spring, D. Wetherall, and T. Anderson, "Inferring link weights using end-to-end measurements," in *Proc. ACM SIGCOMM Internet Meas. Workshop*, Marseille, France, Nov. 2002, pp. 231–236.

Francois Clad received the M.Sc. degree in computer science from the University of Strasbourg, Strasbourg, France, in 2011, and is currently pursuing the Ph.D. degree in computer science with the Network research team of the ICube Laboratory, University of Strasbourg.

His research interests include routing in ISP networks and Internet measurements.



Pascal Mérindol received the Ph.D. degree in computer science from the University of Strasbourg, Strasbourg, France, in 2008.

Then, he spent two years with the Université catholique de Louvain, Louvain-la-Neuve, Belgium, as a Post-Doctoral Researcher. He is now an Assistant Professor with the Network research team of the ICube Laboratory, University of Strasbourg. His main research topics are routing and Internet topology discovery.

Jean-Jacques Pansiot received the M.Sc. degree in computer science from Nancy University, Nancy, France, in 1972, the Ph.D. degree in computer science from Cornell University, Ithaca, NY, USA, in 1976, and the "Doctorat d'Etat" from the University of Strasbourg, France, in 1983.

He joined the Department of Computer Science, University of Strasbourg, in 1984, where he was successively appointed as Assistant Professor, Associate Professor, and Full Professor. He is a member of the Network research team of the ICube Laboratory, University of Strasbourg. His research interests include routing, multicasting, traffic engineering, and Internet cartography.

Pierre Francois (S'06–M'11) received the Master's degree in computer science from the Facultés Notre Dame de la Paix, Namur, Belgium, in 2003, and the Ph.D. degree in computer science, on improving the convergence of IP routing protocols, from Université catholique de Louvain, Belgium, in 2007.

He is now a Staff Researcher with Institute IMDEA Networks, Madrid, Spain, where he notably carries out research in collaboration with ISPs on network management. He leads a Cisco Systems University Research Program grant on BGP. His work includes several papers published in top conferences and journals within the networking field, as well as multiple Internet Engineering Task Force (IETF) Working Group documents and RFCs, in various working groups of the IETF Routing and IETF Operations and Management areas.

Olivier Bonaventure (M'92) graduated from the University of Liège, Liège, Belgium, in 1992, and received the Ph.D. degree in 1999.

He spent one year with Alcatel, Antwerp, Belgium. He is now a Full Professor with the Université catholique de Louvain, Louvain-la-Neuve, Belgium, where he leads the IP Networking Laboratory. He has published more than 80 papers and was granted several patents.

Prof. Bonaventure serves on the Editorial Board of the IEEE/ACM TRANSACTIONS ON NETWORKING. He currently serves as Education Director within ACM SIGCOMM and is a member of the CoNEXT Steering Committee. He received several awards including the IEEE INFOCOM 2007 Best Paper Award and the 2012 USENIX NSDI Community Award.