Distributed fast loop-free transition of routing protocols (submitted version)

Nina Bekono, Nancy El Rachkidy, Alexandre Guitton
Université Clermont Auvergne, CNRS, LIMOS, F-63000 CLERMONT-FERRAND, FRANCE
Emails: {nina_pelagie.bekono, nancy.el_rachkidy, alexandre.guitton}@uca.fr

***Abstract*—In networks that operate during a long time, the routing protocol might have to be changed: this is the case when the network administrator plans a router change. Loop-free transition algorithms are used to ensure that there is no loop during the migration from the initial routing protocol to the final routing protocol. In this paper, we propose a distributed loop-free transition algorithm, called DLF (distributed loop-free heuristic). The algorithm is based on the fact that routing loops resulting from the removal of a node are localized, and can be detected efficiently. We show through simulations that DLF compares well with the existing centralized algorithms, and outperforms the existing distributed algorithm, in terms of migration duration.**

I. Introduction

In computer networks that have a long lifetime, the routing protocol may have to be changed without service interruption. This is the case when a security update of a routing protocol appears [1], when significant modifications of the topology or link metrics have to be taken into account [2], [3], [4], or to handle the apparition of urgent traffic in a wireless sensor network designed for energy efficiency [5]. Another example is when a change of router is planned: link metrics around the router can be increased until no route traverses these links anymore; then, the router can be safely removed [6].

Migrating from an initial routing protocol to a final routing protocol, without causing any loops is a complex task. Several heuristics have been proposed for this task. Currently, all heuristics are centralized. The centralized entity knows the whole topology and both routing protocols, and computes a sequence of steps called transition. In each step, some routers migrate from the initial routing protocol to the final routing protocol, in arbitrary order. The main goal of these heuristics is to reduce the number of steps of the transition, in order to shorten the migration.

In this paper, we propose a distributed heuristic called DLF (distributed loop-free heuristic) in order to achieve this task. DLF is specifically designed to handle the change of a single router. It is based on a mechanism that simultaneously allows for an efficient loop detection and a mechanism to solve them. It requires a limited control overhead (approximately two messages per router and per destination) and allows fast migrations.

The remainder of the paper is organized as follows. Section II describes relevant research works of the literature, and presents all heuristics exhaustively, to the best of our knowledge. Section III presents in details our proposed heuristic DLF, first for one destination, and then for several destinations. Then, we discuss how DLF can be used to cope with router failures, rather than with a carefully planned router removal. Section IV compares the performance results of DLF with the other heuristics. Finally, Section V concludes our work.

II. Related work on loop-free transition

In this section, we give details about heuristics that are used to remove transient routing loops from the network. These heuristics are centralized and we classify them into three categories.

A. Heuristics based on the final routing protocol

RTH (routing tree heuristic) [7] is the first protocol that allows a router to be removed without creating transient routing loops. RTH is centralized. It performs a pre-processing in which nodes that have the same next-hop on both routing protocols migrate immediately. Then, RTH states that a node is allowed to migrate to the new routing protocol when all its successors on the path to the destination have already migrated. RTH builds a constraint graph based on these constraints, and a topological sort produces the transition order.

Let us consider the example shown on Fig. 1. All nodes route packets toward the destination node 10, represented by a double-circle, using the initial routing protocol $\mathcal{R}_i$, represented by solid lines. $\mathcal{R}_i$ is computed based on the shortest-path algorithm from each node to the destination by using real link costs. We consider that node 5 fails. Thus, nodes recompute similarly the final routing protocol $\mathcal{R}_f$, represented by dashed lines, without considering the node 5. Note that if node 1 migrates before node 8, or if node 7 migrates before node 1, a transient routing loop occurs. RTH produces the following transition: $(\{4\},\{5\},\{10\},\{6\},\{9\},\{8\},\{2\},\{1\},\{3\},\{7\},\{0\})$. Note that in this transition, node 8 migrates before node 1, and node 1 migrates before node 7.

RTH-p (RTH with parallel changes) [8] is an improvement of RTH where some nodes can migrate in parallel. More precisely, nodes are regrouped into steps: nodes of the same step can migrate in arbitrary order, but all nodes of a given step have to migrate before that nodes of the next step start migrating. The main goal of RTH-p is to reduce the length of the transition (in terms of number of steps), and thus, to accelerate the migration.

On the example of Figure 1, RTH-p produces a transition of three steps: $(\{2,4,5,6,8,9,10\},\{1\},\{0,3,7\})$.

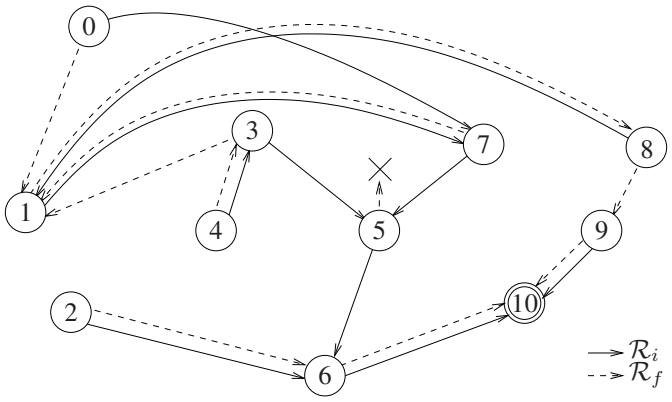

Figure 1. Routing entries for the Sprint topology, for destination 10. Solid lines represent the initial routing protocol $\mathcal{R}_i$, and dashed lines represent the final routing protocol $\mathcal{R}_f$, where node 5 is removed.

Let us denote by RTH-d (distributed RTH) a modification of RTH-p which allows the heuristic to be distributed. In RTH-d, there is no pre-processing (unlike RTH and RTH-p). The resulting constraints become simple: nodes migrate according to their depth on the tree of the final routing protocol.

On the example of Fig. 1, RTH-d produces a transition of five steps: $(\{5,6,9,10\},\{2,8\},\{1\},\{0,3,7\},\{4\})$.

B. Heuristics based on weight increments

GBA (greedy backward algorithm) [6] computes a sequence of weight increments to be applied around the node to be removed. GBA builds a set of constraints for each possible loop, in order to ensure that one node of the loop migrates before the other. The overall sequence is built greedily from the constraints. Authors tested GBA on realistic topologies. They proved that removing a node from the realistic topologies computes transient routing loops of size 2 nodes per loop.

On the example of Fig. 1, GBA identifies loops $\{1,7\}$ and $\{1,8\}$ for destination 10. It produces a sequence of weight increments, which we translate into a transition of three steps: $(\{2,4,5,6,8,9,10\},\{1\},\{0,3,7\})$. Note that, to remove the existing transient routing loops, node 8 migrates before node 1, and node 1 migrates before node 7.

AGBA (adjusted GBA) [6] was proposed to avoid intermediate forwarding loops, which might occur when applying the weight increments, if nodes do not migrate temporarily to a next-hop which is neither on the initial routing protocol $\mathcal{R}_i$, nor on the final routing protocol $\mathcal{R}_f$. In this paper, we decided to focus on GBA, since GBA produces smaller sequences of weight increments than AGBA, and since our algorithm that transforms sequences of weight increments into transitions avoids intermediate forwarding loops.

C. Heuristics based on strongly connected components

SCH-p (strongly connected components heuristic with parallel changes) [8] and ACH (avoiding cycles heuristic) [9] are both centralized heuristics based on strongly connected components. They both identify loops by computing strongly connected components in the graph having links of the two routing protocols. In each strongly connected components of two nodes or more, they determine which nodes can migrate first. To do so, SCH-p uses a greedy algorithm, and ACH uses an exhaustive enumeration of all loops.

On the example of Figure 1, SCH-p produces the following transition: $(\{0,2,3,4,5,6,8,9,10\},\{1\},\{7\})$.

The management of multiple destinations is also different in SCH-p and ACH. In SCH-p, the transition for all destinations is equal to the concatenation of the transitions of each destination. In ACH, the transition for all destinations is equal to the merge of the transitions of each destination: the i-th step of the transition for all destinations is equal to the union of the i-th step of each destination. In this paper, we decided to focus on SCH-p, since the running time of ACH is greatly impacted by the number of loops. We also decided to implement in SCH-p the merging of steps proposed in ACH. With this improvement, SCH-p and ACH yield similar results in terms of number of steps on realistic topologies.

III. Proposition: Distributed loop-free heuristic

We propose a distributed heuristic, called DLF (distributed loop-free heuristic), to perform a fast loop-free transition in the case of the removal of a node (as in GBA and AGBA). In Subsection III-A and Subsection III-B, we assume that both initial and final routing protocols, $\mathcal{R}_i$ and $\mathcal{R}_f$, are known in advance, for each node. In Subsection III-C, we explain a more realistic setting where the final routing protocol $\mathcal{R}_f$ is known only by nodes that have been previously informed of the topology changes.

A. DLF for one destination

DLF exploits the property that in real topologies, when a single node is removed, all loops produced between the initial and the final routing protocols, $\mathcal{R}_i$ and $\mathcal{R}_f$, have a size of 2 [6] (both GBA and AGBA are largely based on this property too). On each loop, both routing protocols are necessarily involved (as each routing protocol is loop-free when considered independently). Detecting all loops starting from node n becomes simple: a node n is on a loop if and only if $\mathcal{R}_i(\mathcal{R}_f(n)) = n$ or if $\mathcal{R}_f(\mathcal{R}_i(n)) = n$. Resolving a loop (x,y) requires the node closest to the destination on $\mathcal{R}_f$ to perform the transition before the other node. In other words, if $\mathcal{R}_f(x) = y$, it means that y is closer to the destination than x according to $\mathcal{R}_f$, thus y has to switch to $\mathcal{R}_f$ before x. Otherwise (if $\mathcal{R}_f(x) \neq y$), x has to switch before y.

Algorithm 1 describes the DLF heuristic. Initially, n sends a control message `checkingLoop1` to $\mathcal{R}_f(n)$ and waits for a reply: if $\mathcal{R}_f(n)$ detects that $\mathcal{R}_i(\mathcal{R}_f(n)) = n$, it sends a `loopDetected1` message to n, otherwise, it sends a `noLoop1` message. If $\mathcal{R}_i(\mathcal{R}_f(n)) \neq n$ or if the loop was resolved (see later), n determines whether $\mathcal{R}_f(\mathcal{R}_i(n)) = n$ or not by sending a `checkingLoop2` control message to $\mathcal{R}_i(n)$ and waiting for either a `loopDetected2` message or a `noLoop2` message. If n is on a loop with node $\mathcal{R}_i(n)$, n can migrate directly as it is closest to the destination according to $\mathcal{R}_f$ than $\mathcal{R}_i(n)$ (indeed, the next-hop of $\mathcal{R}_i(n)$ on $\mathcal{R}_f$ is

n). After its migration, node n has to inform $\mathcal{R}_i(n)$ with a `loopSolved` message. If n is not on a loop with node $\mathcal{R}_i(n)$, n can migrate directly and does not have to inform any node. The whole process requires four control messages by node, and one additional control message for each loop.

Algorithm 1: Distributed loop-free heuristic.

Data: n is the current node, $\mathcal{R}_i$ and $\mathcal{R}_f$ are known *a priori* by all nodes
if $\mathcal{R}_i(\mathcal{R}_f(n)) = n$ **then**
  n waits for $\mathcal{R}_f(n)$
end
if $\mathcal{R}_f(\mathcal{R}_i(n)) = n$ **then**
  n migrates to $\mathcal{R}_f$
  n informs $\mathcal{R}_i(n)$ of the migration
else
  n migrates to $\mathcal{R}_f$
end

Let us consider the example of Fig. 1, where node 5 is removed. To simplify the explanation, we consider here that nodes in DLF migrate in steps. The next step is initiated when all nodes have either migrated or are waiting for another node. There are two loops on Fig. 1: one between nodes 1 and 8, and one between nodes 1 and 7. $\mathcal{R}_i(\mathcal{R}_f(1)) = \mathcal{R}_i(8) = 1$, thus node 1 has to wait for node 8 to migrate. $\mathcal{R}_i(\mathcal{R}_f(7)) = \mathcal{R}_i(1) = 7$, thus node 7 has to wait for node 1 to migrate. $\mathcal{R}_i(\mathcal{R}_f(8)) = \mathcal{R}_i(9) = 10 \neq 8$ and $\mathcal{R}_f(\mathcal{R}_i(8)) = \mathcal{R}_f(1) = 8$, thus node 8 migrates immediately, and will inform node 1 that it can migrate with a `loopSolved` message. Thus, the first step is equal to $\{0, 2, 3, 4, 5, 6, 8, 9, 10\}$. During the second step, node 1 receives the `loopSolved` message from 8. Since $\mathcal{R}_f(\mathcal{R}_i(1)) = \mathcal{R}_f(7) = 1$, node 1 migrates immediately, and will inform node 7 that it can migrate with another `loopSolved` message. Thus, the second step is equal to $\{1\}$. During the third step, since $\mathcal{R}_f(\mathcal{R}_i(7)) = \mathcal{R}_f(5) = 5 \neq 7$, node 7 migrates immediately without informing any node. Thus, the third step is equal to $\{7\}$.

The number of control messages can be further reduced, as there is no need to send the `checkingLoop2`, `loopDetected2` and `noLoop2` control message. Indeed, when a loop is detected by $\mathcal{R}_f(n)$ due to the reception of a `checkingLoop1` control message, node $\mathcal{R}_f(n)$ stores that it will have to inform node n after its own migration. This new protocol requires two control messages by node (in order to detect loops), and one additional control message for each loop (in order to resolve loops).

B. DLF for multiple destinations

When there are multiple destinations, DLF processes each destination in parallel. In terms of steps (which allows a simplified explanation for distributed protocols), this corresponds to merging the steps for all destinations. Let us denote by $step(i, j)$ the set of nodes of the i-th step of DLF for the single destination j. Then, the i-th step of DLF for all destinations is equal to $\cup_{j=1}^{d} step(i, j)$, where d is the number of destinations.

C. Discussion on topology change notification

In Subsect. III-A and Subsect. III-B, we considered the same parameters settings used in [6] in order to perform a fair comparison: we considered that each node knows in advance its next-hop according to the initial routing protocol, $\mathcal{R}_i$, and according to the final routing protocol, $\mathcal{R}_f$.

In this subsection, we discuss how this assumption can be removed. Thus, we assume that all nodes know only $\mathcal{R}_i$ initially. When a router fails, the neighboring nodes detect this failure and have to inform all the other nodes of the network. DLF can be adapted for this setting in the following manner. When a node n detects the failure of its direct neighbor, it recomputes the new shortest path to the destination (that is, $\mathcal{R}_f(n)$). Then, it sends the `checkingLoop1` control message to $\mathcal{R}_f(n)$. If the new next-hop replies by a `noLoop1` message control, then node n can migrate immediately. Otherwise, node n delays its migration until it receives a `loopSolved` control message from $\mathcal{R}_f(n)$. In the meanwhile, n informs its own neighbor of the failure of the router.

Let us consider the example of Fig. 1 where node 5 fails. Initially, the neighbors of node 5, represented by the set $\{3, 6, 7, 9\}$, notice the link failure with 5. Nodes from $\{3, 5, 6, 7, 9\}$ are thus informed of the failure, and know both $\mathcal{R}_i$ and $\mathcal{R}_f$. Fig. 2 shows in gray these nodes. These nodes execute the DLF heuristic, and nodes 3, 6 and 9 are able to migrate to $\mathcal{R}_f$. However, node 7 is unable to migrate because it receives a `loopDetected1` message from $\mathcal{R}_f(7) = 1$. On the figure, it can be seen that among the informed nodes, only node 7 is still on $\mathcal{R}_i$. Fig. 3 shows the situation when all nodes are informed about the failure. Only nodes that are waiting for a `loopSolved` message are still on $\mathcal{R}_i$. On the example, this is the case for nodes 1 and 7. As soon as node 8 informs node 1 with the `loopSolved` message, node 1 will migrate to $\mathcal{R}_f$ (its next-hop becoming node 8) and will inform node 7. Then, node 7 will migrate to $\mathcal{R}_f$ (its next-hop becoming node 1).

IV. Simulation Results

In this section, we compare the performance of our proposition DLF with the performance of RTH-d, GBA and SCH-p. Recall that only DLF and RTH-d are distributed, while GBA and SCH-p (as well as RTH, RTH-p, AGBA and ACH, which are not shown here) are centralized.

A. Performance metrics

Our main performance metric is the duration of the migration. For centralized heuristics, this is defined as the number of steps of the transition. For distributed heuristics, as mentioned previously, we consider that all nodes that can migrate without waiting for other nodes do so during the same step; when a node n is waiting for a node n', they migrate in two different steps.

We also compute the control overhead in terms of number of messages. For centralized heuristics, we assume that the centralized entity has to notify independently each node to

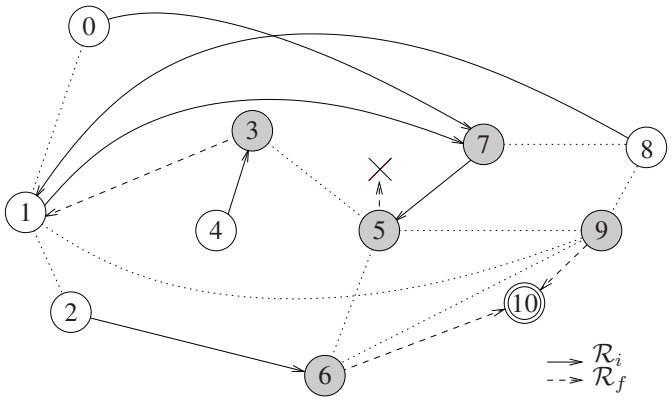

Figure 2. Node neighbors of 5, which are nodes of $\{3, 6, 7, 9\}$, notice the disconnection with node 5 when it fails. They recompute the shortest-path to the destination without producing routing loops. Thus, the next-hop of 3 becomes 1, the next-hop of 6 becomes 10, and the next-hop of 9 becomes 10. However, node 7 does not change its next-hop to avoid a routing loop with 7.

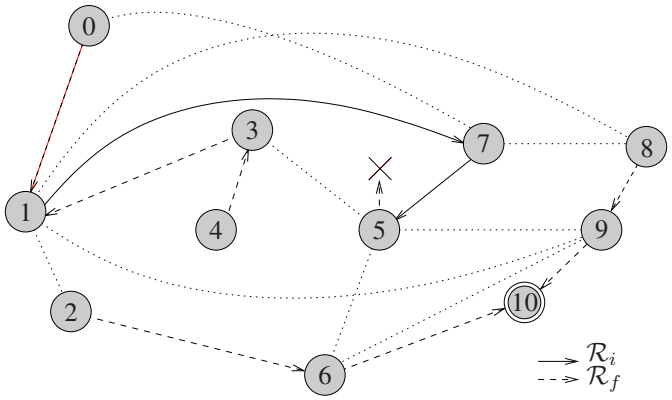

Figure 3. Once all the nodes are informed about the node failure, they compute the new shortest-path to the destination. Only nodes that do not yield to routing loop have migrated on this example. The others are waiting for a `loopSolved` message.

migrate, for each destination and at the beginning of each step. Each notification requires a number of control messages which is equal to the distance between the centralized entity and the node. Note that the centralized entity is chosen at the most central location of the network, in order to minimize the number of control messages. On the example of Figure 1, if we assume that the centralized entity is node 10 (which is not the most central, but simplifies the computation of paths, and thus eases our presentation), the overall number of control messages for destination 10 is the sum of the number of hops of all paths from nodes to node 10, which is equal to 29 on the initial routing protocol of Fig. 1. For distributed heuristics, we assume that both routing protocols are known in advance. We take into account the number of messages to identify loops (for DLF) and to inform that a node has to migrate (for both RTH-d and DLF), for each destination.

B. Simulation settings

Our simulations are performed on real topologies collected from the Internet Topology Zoo from University of Adelaide [10]. We used topologies of size varying from 9 to 278 nodes. We consider that all nodes, except the one that fails, are destinations. Simulations results are averaged over 100 repetitions and confidence intervals are of 95%.

For each destination d and for each repetition, we assigned a random weight chosen uniformly at random within $[1; 100]$ to each link. The initial routing protocol $\mathcal{R}_i^d$ is built as an inverse shortest-path tree rooted at d. Then, we chose a random node $r \neq d$ to remove. The final routing protocol $\mathcal{R}_f^d$ is built as an inverse shortest-path tree in the topology where node r is removed.

C. Results

In the following, we first present the average number of loops we obtained on the topologies, as it has an impact on the number of steps for all heuristics. Then, we present our main performance metrics: the duration of the migration, which is represented by the number of step needed to switch from $\mathcal{R}_i$ to $\mathcal{R}_f$, and the control overhead.

1) Number of loops: Figure 4 shows the average number of loops (using a logarithmic scale for the y-axis), as a function of the network size. The number of loops is computed over all destinations, and averaged over all repetitions. The number of loop increases with the topology size, and reaches several hundreds for larger topologies. Note that each loop has a size of 2, as expected from [6].

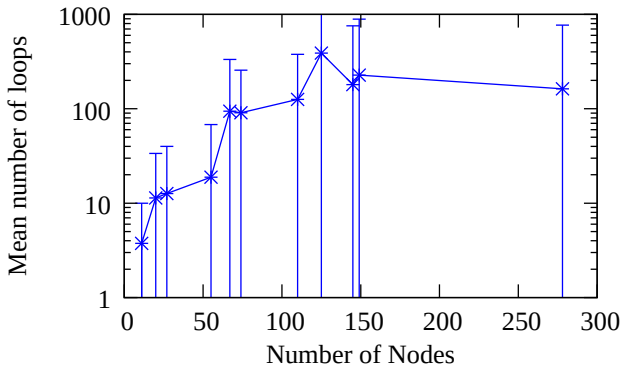

Figure 4. The number of loops increases with the number of nodes in the network and with the number of destinations considered.

2) Duration of the migration: Figure 5 shows the duration of the migration, which is the number of steps of the transition from $\mathcal{R}_i$ to $\mathcal{R}_f$, for all heuristics, as a function of the network size. For RTH-d, the migration duration is equal to the depth of the final routing protocol. For large topologies, this yields to transitions with several steps, and thus RTH-d yields long migrations. For GBA, the computation of the steps depends on the overall number of loops computed. When the number of loops is large, GBA builds transition steps that are not optimal due to the greedy construction of the steps. SCH-p is able to yield transitions with a very small number of steps, due to

the fact that the strongly connected component concept allows small loops to be solved efficiently. DLF is able to achieve the same performance results as SCH-p due to the fact that the numerous small loops can be processed independently very efficiently. For the large network topologies, the gain of DLF with respect to RTH-d is up to 84%, and the gain of DLF with respect to GBA is up to 57%.

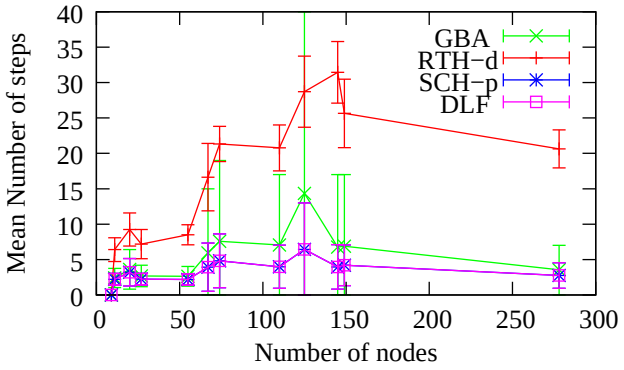

Figure 5. The migration duration is much shorter with SCH-p and DLF than with GBA and RTH-d.

3) Control overhead: Figure 6 shows the control overhead in terms of number of control messages, as a function of the number of nodes in the network. RTH-d produces a small number of control messages, as all nodes are informed by a single flooding on the (reversed) final routing protocol. GBA and SCH-p require a similar number of control messages, which is the cost due to the notification of all steps by the centralized entity. Since the number of loops is much smaller than the number of control messages for each node and for each destination, DLF requires about twice the number of control messages of RTH-d.

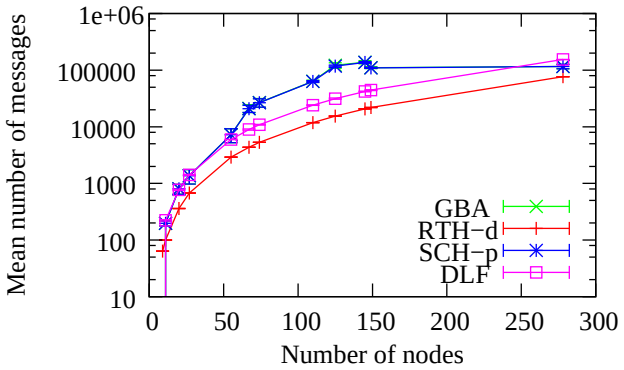

Figure 6. The number of control messages of the heuristics depend mainly on the number of nodes and on the number of destinations.

Globally, DLF is able to yield migrations of short duration, in a distributed way, and with a reasonable number of control messages (that is, slightly more than two control messages per node and per destination). When compared to the centralized heuristics (GBA and SCH-p), DLF shows a gain of about 70% for large network topologies. However, DLF sends a large number of control messages compared to RTH-d. RTH-d shows a gain of about 50% compared to DLF.

V. Conclusions

When there is a significant change in the network topology, such as when a router change is planned, transient routing loops might occur in the network. In this paper, we proposed a distributed loop-free transition algorithm called DLF. DLF exploits the property that loops caused by the removal of a node are localized: thus, DLF is able to detect and solve loops with a limited number of control messages. Simulation results show that DLF performs as well as the best centralized heuristics in terms of migration duration. Although DLF requires about twice more messages than the other distributed heuristic called RTH-d, DLF is able to perform the migration in a significantly smaller duration (with a gain of up to 84% in terms of number of steps of the transition for large networks).

Acknowledgment

This work has been sponsored by the French government research program "Investissements d'avenir" through the IMobS3 Laboratory of Excellence (ANR-10-LABX-16-01), by the European Union through the regional program competitiveness and employment 2014-2020 (ERDF - Auvergne region), and by the Auvergne region.

References

[1] K. Butler, T. R. Farley, P. McDaniel, and J. Rexford, "A survey of BGP security issues and solutions," *Proceedings of the IEEE*, vol. 98, no. 1, pp. 100–122, 2010.

[2] Z. Zhong, R. Keralapura, S. Nelakuditi, Y. Yu, J. Wang, C. N. Chuah, and S. Lee, "Avoiding transient loops through interface-specific forwarding," in *IEEE/ACM IWQoS (International Symposium on Quality of Service)*, ser. Lecture Notes in Computer Science, vol. 3552. Springer, 2005, pp. 219–232.

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

[4] S. Nelakuditi, Z. Zhong, J. Wang, R. Keralapura, and C. N. Chuah, "Mitigating transient loops through interface-specific forwarding," *Computer Networks*, vol. 52, no. 3, pp. 593–609, 2008.

[5] L. Le Guennec, N. El Rachkidy, A. Guitton, M. Misson, and K. Kelfoun, "MAC protocol for volcano monitoring using a wireless sensor network," in *NoF (International Conference on Network of the Future)*, 2015.

[6] F. Clad, S. Vissicchio, P. Mérindol, P. François, and J. Pansiot, "Computing minimal update sequences for graceful router-wide reconfigurations," *IEEE/ACM Trans. Netw.*, vol. 23, no. 5, pp. 1373–1386, 2015. [Online]. Available: http://dx.doi.org/10.1109/TNET.2014.2332101

[7] L. Vanbever, S. Vissichio, 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, 2012.

[8] N. El Rachkidy and A. Guitton, "Changing the routing protocol without transient loops," *Computer Communications*, 2016.

[9] N. Bekono, N. El Rachkidy, and A. Guitton, "Fast loop-free transition of routing protocols," *Vehicular Technology Conference*, 2016.

[10] Internet Topology Zoo. Last updated on 18th of July 2012. [Online]. Available: http://www.topology-zoo.org/dataset.html