An automatic restoration scheme for switch-based networks

Jacques Carlier, Joël Lattmann, Jean-Luc Lutton, Dritan Nace, Thanh Son Pham

An automatic restoration scheme for switch-based networks

Jacques Carlier*, Joel Lattmann†, Jean-Luc Lutton†, Dritan Nace*,Thanh Son Pham†*,
*Sorbonne université, Université de technologie de Compiègne, UMR CNRS 7253, Heudiasyc, CS 60319, 60203 Compiègne cedex
†Orange Labs, Orange Gardens, 40 - 48 avenue de la Republique, 92320 Chatillon, France

***Abstract*—This paper presents a fully automated distributed resilient routing scheme for switch-based or new generation router based networks. The failure treatment is done locally and other nodes in the network do not need to undertake special actions. In contrast to conventional IP routing schemes, each node routes the traffic on the basis of the entering arc and of the destination. The resulting constraint is that two flows to the same destination entering in a node by a common arc have to merge after this arc. It is shown that this is sufficient for dealing with all single link failure situations, assuming that the network is symmetric and two-link connected. Two heuristic approaches are proposed to handle the corresponding dimensioning problem for large network instances. The proposed method generalizes some methods of literature [6], [8] and provides more cost-efficient solutions.**

I. Introduction

The Internet has been hugely successful in spreading its services all over the world. But the more firmly it becomes implanted, the harder it becomes to introduce new developments. This is the phenomenon of Internet ossification. A number of published works have attempted to rethink the architecture of the Internet. The context of this study is the total or partial replacement of conventional routers by switches or new generation routers which have extended features comparing to the conventional ones. In the following we will focus on switch-based networks keeping in mind that the new generation routers have the capacity to undertake the same actions in practice. Usually the routing in some node depends only of the destination. Networks based on switches controlled by an external controller may represent an interesting alternative when one deals with failure situations. With switches, we can also take into account the entering arc under the constraint that two flows to a same destination arriving from the same entering arc merge after this node. Otherwise we will say there is a conflict. A routing scheme with no routing paths in conflict is called a free-conflict scheme and it is formally defined in Section III. In the case of single link failure, for a given destination, only one of the two outermost nodes of the failed link needs to react by rerouting the disturbed traffic towards one of its neighbors. The traffic is then routed according to the filters programmed in each node of the network. The proposed scheme requires a local reaction only, making its implementation particularly easy in a distributed environment. This local reaction helps the network to operate normally and it can solve the problem of transient failures. We recall that a transient failure is a failure of short duration (less than ten minutes), while a persistent failure is longer. When it has been determined that a failure is persistent, the controller can recalculate the routing tables for all the nodes in the network. In order to prevent the rerouted traffic (following a failure) causing disturbances in another part of the network, additional capacities are assigned to all the arcs in the network. We first state and prove an existence theorem, ensuring that there exists a valid rerouting algorithm for a network based on switches if it is symmetric and biconnected. This leads to the first rerouting algorithm. We also propose a second rerouting algorithm which provides more flexibility in the choice of the rerouting paths. Both algorithms permit to introduce heuristics to deal with larger networks (please note that the problem of capacity assignment for the problem resilient network dimensioning in hand is NP-hard as it includes the integer multiflow routing problem as part of it). These heuristics are proposed and compared to previous methods introduced by Xi and Chao [8], and Wang and Nelakuditi [6]. Numerical results with respect to capacity assignment to a resilient switch-based network illustrate the performance of our methods comparing to those of literature. The paper is organized as follows. After this introduction, section 2 reports the related works. In section 3, we present our resilient rerouting scheme and its theoretical properties together with a complete example. Heuristic approaches deduced from the theoretical study are also proposed. Section 4 is devoted to numerical results.

II. Related Works

The problem of failure resilience has been widely investigated in literature. Many works have been proposed based on multi-protocol label switching (MPLS) [10] and devoted to real-time systems [11]. Nevertheless, when dealing with switch-based networks the literature is less abundant. Our restoration scheme is pre-calculated. As a result of its pre-calculated scheme, our protocol does not need any communication between network nodes, which guarantees a very short recovery time. We will present the principal works studying similar problems for IP networks and switch-based networks.

1) Rerouting in IP networks: Xi et Chao [8] propose a method for calculating backup paths which permits to reroute the traffic in case of link failure. The proposed scheme, called IPFRR (IP Fast ReRoute), uses two types of port, primary and backup ports. Normally, the node uses the primary port

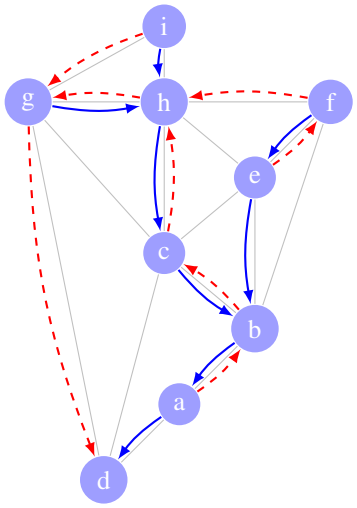
to route the outgoing traffic. When there is a failure on the primary port, the node will use the backup port to reroute the traffic. Also, the node uses the backup port when the node receives the traffic coming from its outgoing primary port. This packet forwarding policy makes the traffic follow the reverse routing path starting from the upper node of the failure link until a node where it can use its backup port. We will see that this method is similar to the first one we present in Section III.

Another similar method is presented in Wang and Nelakuditi [6]. It determines the next hop for traffic when the destination and entering interface of traffic are provided. On the other hand, it calculates the rerouting path using shortest paths with the metrics stored in each node. We note that in symmetric graphs, as in our study, the shortest rerouting path computed in [6] coincides with the rerouting path in the previous method. Consequently, this method becomes identical to the method of Xi et Chao [8].

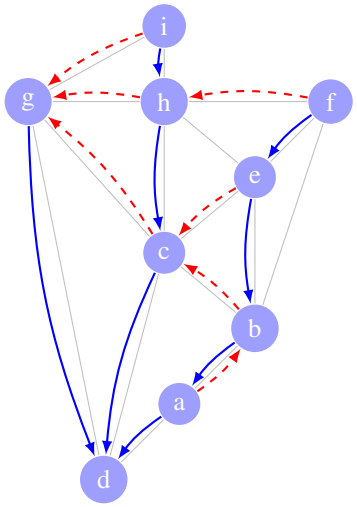
A number of methods have been proposed for IP fast rerouting, in order to solve the problem of transient failure. Nevertheless, these methods have the following limitations:

- With loop-free alternate mechanism based methods [1], there is no guarantee that traffic can be rerouted for all destinations. These methods can only help to reduce the number of lost packets in an IP network.
- Not-via addressing [2] and tunneling [3] mechanisms require the encapsulation and decapsulation of packets, while in multiple routing configurations mechanism [4], the packets need to carry configuration information. In the same line, Segment Routing (SR) paradigm [23] can be directly applied to the MPLS architecture with no change to the forwarding plane. A segment is encoded as an MPLS label. An ordered list of segments is encoded as a stack of labels. The segments are processed following the labels in the top of the stack and they are consecutively popped from it upon completion. All these methods are different to our approach because they propose modifiying the packet header.

Finally, some works dealing with multicast trees can be useful in treating IP transient failures. First, a method is proposed in [7] to create simultaneously two routing trees not sharing arcs between them, which are named B (Blue) and R (Red). In this method, after removing any edge in the graph, the source s remains connected to all vertices through B and/or R. We notice that reversing the solution of [7] can provide a routing scheme which may handle any single link failure. We can see an example of this method in Fig. (1a). The two routing trees are described by the bold arcs and the bold dotted arcs. However, this doesn't work when the nominal routing tree is fixed, which is the case in our study. This is explained in the Fig. (1b). When the nominal routing tree covers all the directed arcs to destination d, we cannot construct the two routing trees by using [7]. Recently, in the same spirit as above, Chiesa et al. [20] generalize this idea in k-connected networks. They conjecture that *for any k-connected graph, basic failover routing can be resilient to any k-1 failures*. In [21] the authors propose a similar approach to our first algorithm described



(a) An example inspired from [7], where two separated routing trees allow for resilient routing in case of single link failures.



(b) An example showing that when a nominal routing tree is given in the network there may be not possible to build a separate routing tree by using [7].

Fig. 1: An example of network illustrating the problem of existence of two simultaneously routing trees.

in the following. It uses a precomputed next-hop in case of failure. It uses also tags in the packets to help the rerouting. In contrast to the above works, our model gives flexibility to choose the next hop and does not require modifying the packet headers. Also, our proposal works for the case when the nominal routing is fixed from the network manager. Finally, a computational method is given to compute a minimal link-cost network resilient to any single link failure while ensuring 100% traffic satisfaction.

From above we can state that two main points need to be addressed: first, proposing a generic routing scheme dealing with single link failures which works when a nominal routing scheme is given, and second, ensuring more flexibility in the choice of alternative routing paths. We will answer to these points in the next section.

2) Rerouting in networks using switches: In this section, we cite several works about failure resilience that are applied to switch-based networks. Some of these works are applied to specific types of networks as CDN (Content Delivery Network). They call for the use of multi-controllers or master-slave controller structures ensuring failure resilience when a controller fails, [14], [16], [19]. With respect to SDN (Software Defined Network), as remarked also in [22], we notice that OpenFlow is the main method dealing with resilience issues. In such networks, when a switch detects a link failure, it notifies it to the controller for taking the required action. All this may delay the restoration time. Then, the alternative methods proposed in the literature as [9], [15], [18] and others quoted in [22] allow reducing considerably the restoration time but they require header packet modification, which makes the main difference with our proposal.

In order to have an autonomous restoration system in switch-based network, we need to propose a method which is pre-calculated and which does not need to modify the packets. Our approach presented in next section satisfies the above conditions. It helps to reroute the traffic without intervention of the controller. The methods of [8] and [6] can also be applied to switch-based networks. However they do not perform so efficiently when solving the corresponding network dimensioning problem.

III. Theoretical study

In this section we detail our scheme for routing and rerouting in case of non-simultaneous single link failures. The considered network is modelled using a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, directed and symmetric (i.e., each link includes the two opposite arcs), composed of a set of n nodes $\mathcal{V}$ and a set of m links $\mathcal{E}$. For any destination in the network the traffic is routed to the destination through a tree, called nominal routing tree. This tree is constructed using specific criteria; for instance, it might be the shortest path tree. In our study we assume that the nominal routing tree is given. In the case of failure of an arc or link (both arcs composing the link are then concerned), the upstream extremity node will deviate the disturbed traffic to one of its neighbours. From this moment, any node traversed by the disturbed traffic will route it according to a free-conflict routing scheme. The notion of free-conflict routing is formally defined as follows:

Definition 1. *A routing scheme is said free-conflict if any two routing paths to the same destination sharing an arc are identical from this arc to the destination.*

In Figure 2, we report a free-conflict scheme. It can be noticed that all alternative paths to the same destination that are not in conflict (see above), can be embedded in a free-conflict scheme. Hence, there exist two free-conflict paths to destination G coming from A. Then, this situation can be represented at node A by two different inputs for destination G, one comes from input 1 and goes through output 2 and the other comes from input 4 and continues through output 3. Then, given a precomputed failure resilient free-conflict routing scheme, a rerouting procedure works as follows:

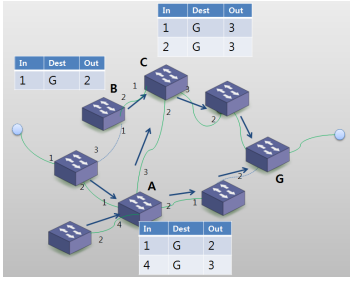


Fig. 2: A free conflict routing scheme

- In case of a (link) failure the upstream node initiates the rerouting procedure by deviating the traffic to some destinations to one of its neighbours. Such information relevant to failures of links is assumed known from each corresponding upstream node for any destination in the network.
- Other nodes in the network don't need to take any specific action. They will forward the traffic according to their routing tables. For any couple (input, destination) it finds the appropriate output and sends the traffic through it.

Nevertheless, the problem of existence of a valid routing/rerouting scheme, i.e. without conflict, for all possible non-simultaneous link failures and destinations is not straightforward.

A. Existence of a restoration scheme

In this section we study the question of whether there exists a rerouting solution without conflicts. We shall make the following assumptions:

- the graph is assumed to be directed with arcs in both directions for each link of the network;
- there exist at least two disjoint-arc paths between any two nodes of the graph;
- for each destination the nominal routing scheme follows a fixed tree;
- only one link failure can occur at a time.

Our goal is to achieve a fully free-conflict routing scheme handling both nominal and failure situations. This routing scheme is composed of a nominal routing scheme realized by a routing tree for each destination merged with the restoration scheme in such a way that there is no conflict with respect to filters. One can notice that routing for both nominal and failure situations is done with respect to a given destination l and there is no interaction between routing schemes for distinct destinations. This suggests restricting the study of existence of free-conflict scheme to the case of a fixed destination without loss of generality. The same can be reproduced for any other destination and merging them leads to a free-conflict scheme as well.

Theorem 1. *For single link failures perturbing the nominal routing tree there exists a rerouting scheme without conflict for any destination l.*

Proof. Let A be the routing tree to the destination l. l is therefore the sink of A. Let (p_1, q_1) be an arc of A subject

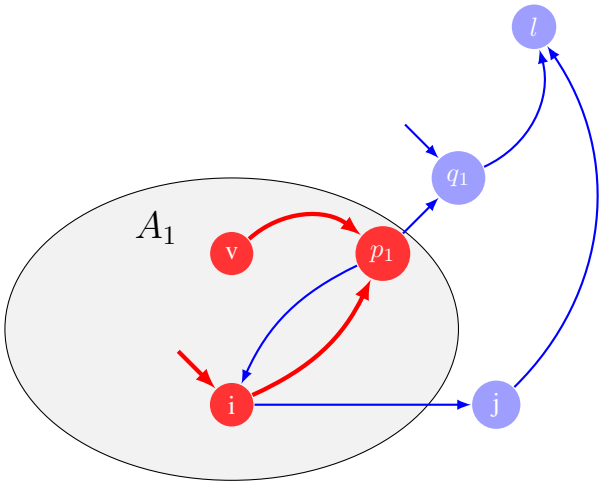


Fig. 3: Proof illustration of Theorem 1: in case of failure of arc (p_1, q_1), the rerouting of traffic originated from v is done by backtracking on the routing tree from p_1, until jumping out of A_1 through arc (bridge) (i, j).

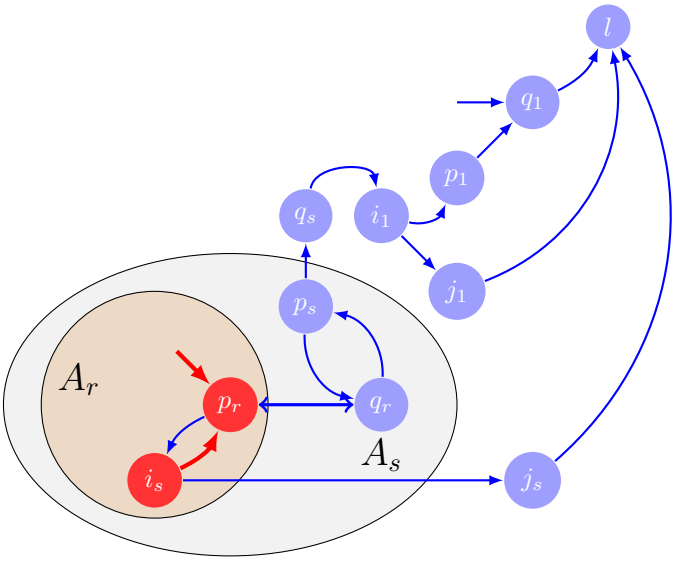


Fig. 4: Recurrence Hypothesis - Case 1. We assume that failure of link (p_r, q_r) creates sub-tree A_r which is included in sub-tree A_s for some $s < r$. We assume also that bridge (i_s, j_s) used in case of failure of (p_s, q_s) is such that node i_s is included in A_r. Then, the rerouting path used in case of failure of (p_r, q_r) is required to use bridge (i_s, j_s).

to failure. We assume that this arc fails and we must find a rerouting scheme without conflict. We remark that $(q_1,\ p_1)$ does not belong to A. This is clear as for a given destination only one of the two arcs (directions) can be part of the routing tree. Hence, this means that both link and arc failures have the same impact on traffic lost for a given destination and we can restrict ourselves to arc failures. Without loss of generality, we consider in this proof the problem of the failure of the arc (p_1, q_1). p_1 is the sink of a sub-tree A_1 whose nodes are denoted as red (i.e., in the Figure 3 nodes v, p_1 and i are part of tree A_1). The other vertices in the tree are colored in blue. Without loss of generality we assume that all vertices are part of the tree A ($|A| = n$) and this is true for any destination l.

We know that there are at least two disjoint paths in the initial graph going from p_1 to l. Given the assumption of two-link connectivity the upper part that includes p_1, but does not include the vertex l, must contain at least two outgoing arcs. Therefore there are at least two arcs going out of A_1 (Figure 3). Since one of these arcs is (p_1, q_1), there exists a path μ from p_1 that visits the vertices of A_1, and connects a vertex of A_1, which is red, to a blue vertex. So there is at least one arc (i, j) (we call this arc a bridge) of μ connecting the red vertices to the blue vertices (Figure 3). We associate with this arc a rerouting path for the failed arc. Let v be a red vertex of A_1 which is affected by the failure. Traffic to destination l and coming from v goes first to p_1, then it follows the reverse path on the nominal routing tree (it is assumed above that arcs in opposite directions are present for each link) from p_1 to i. It uses bridge (i, j), then from j to destination l follows the path on the nominal routing tree. According to the above choice of the rerouting path, traffic coming from different sources in A to destination l associated with the breakdown of (p_1, q_1) necessarily follows rerouting paths without conflict.

We now consider the $r - 1$ failures of arcs of the tree and choose the corresponding bridges (i, j). We number the arcs of the tree by decreasing order as we approach the sink l, and consequently choose the arcs subject to failure in successive order of increasing numbers. Let (p_r, q_r) be one of the arcs subject to failure under consideration, and let us suppose that we have chosen arcs $(i_1, j_1), (i_2, j_2) \ldots (i_{r-1}, j_{r-1})$ as bridges for constructing the rerouting paths with respect to previously examined failures. p_r is the sink of tree A_r. We consider two cases. The first case is when we have chosen for an arc (p_s, q_s), with s strictly smaller than r, a bridge (i_s, j_s) whose extremity i_s is in the tree A_r and the other extremity j_s outside of the tree A_s. Clearly in this case the tree A_r is contained within A_s (Figure 4). Indeed for any $s < r$ we have either $A_r \subset A_s$ (the path from p_r to destination l goes through p_s) either $A_r \cap A_s = \emptyset$. We therefore choose arc $(i_s,\ j_s)$ as bridge for the tree A_r. Note that there exists at most one bridge with this property. This can be deduced from the inclusion property of trees. Indeed, having two such arcs $(i_s,\ j_s)$ and $(i_t,\ j_t)$, with respect to failures (p_s, q_s) and (p_t, q_t) with $s < t < r$, means that $(i_s,\ j_s)$ is also a bridge for A_t and it should have been chosen instead of $(i_t,\ j_t)$. In the case when $A_s \cap A_r = \emptyset$, there is no rerouting arc with the above property. Then, we choose any arc (i_r, j_r) (Figure 5) that connects A_r to its complement.

We need to show that the rerouting has no conflict. We demonstrate this by recurrence on the number of rerouted arcs. We consider that we have already rerouted $r - 1$ arcs in the tree. By the recurrence hypothesis, we assume that there is no conflict for the first $r - 1$ reroutings. We verify that the r-th rerouting path built as above also has no conflict with the first $r - 1$ reroutings. Regarding the rerouting in the outside part of the tree A_r, that is to say the part that is in common with the nominal routing, there is no conflict by construction. Even if it uses the same arc with some previous rerouting path, in this part it will follow the same rerouting path as far

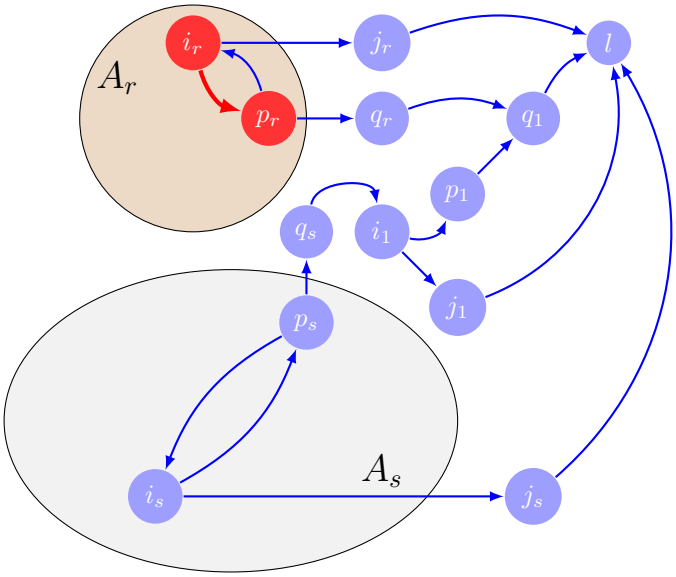
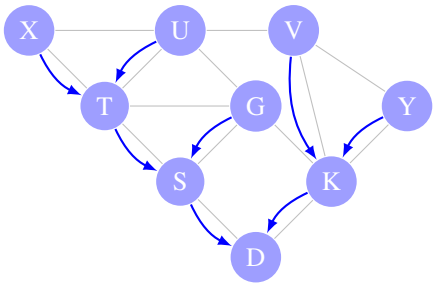


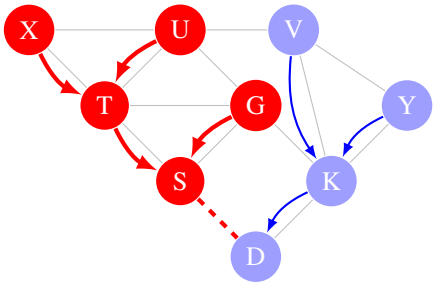
Fig. 5: Recurrence Hypothesis - Case 2. The bridge (i_r, j_r) used in this case is not used as a bridge for any $s < r$. The rerouting path in case of failure of (p_r, q_r) will backtrack from p_r until i_r, take bridge (i_r, j_r) and join l through the nominal routing tree.

as destination l, and so it is without conflict. We must also check that there is no conflict for the part where it goes in the opposite direction in the routing tree, which means verifying that there is no conflict in the two cases considered above. In the first case, where an arc (i_s, j_s) has been chosen in the tree A_r, clearly no conflict is present in that part of the tree because A_r will use the same arcs until i_s and continue with the same bridge (i_s, j_s) between its red part and its blue part. In the second case, the part that ascends the tree can have nothing in common with the other rerouting arcs, since this would imply the existence of (i_s, j_s)! Therefore, there is no conflict in this case either. We can conclude that the property remains true to the order r, and we have therefore demonstrated by recurrence the absence of conflict. □

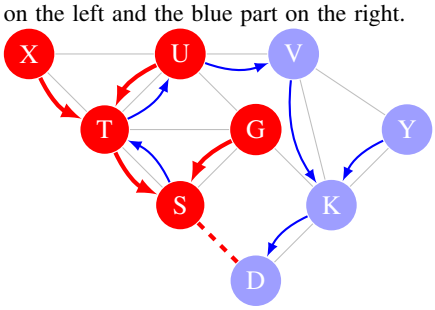
In the following we will present an example illustrating the above method. All the traffic having D as destination can be expressed as a directed tree which converges to destination D (Figure 6a). When the link (S, D) fails, the routing tree will be divided into two parts: the red part, and the blue part (Figure 6b). All the traffic to destination D that transit via node S can no longer use the link (S, D), and will instead be rerouted by the path (S, T, U, V, K, D) that was pre-calculated by our restoration scheme (Figure 6c). This alternative path connects the red part to the blue part and it will not interfere with the nominal routing. In this scheme, in case of failure of the link (S, D), node S is advertised of the failure and it will take the necessary action to restore the traffic through an alternative rerouting path while the other nodes will operate normally as programmed. We can see that the traffic towards destination D over the arc (T, S) is rerouted over the arc (S, T), but this will not cause any looping problem because of the configuration of filters. When the node T receives traffic coming from U and knows that the destination is D, T will transfer the traffic to node S. In the case of (S, D) link failure, node S will send back the traffic through the reverse routing path (the network is assumed directed with arcs in both directions for each link) to T. Knowing that traffic is coming from S and is heading towards D, T will transfer this traffic to U, which in its turn will transfer it to V. Then, the traffic will be transferred from V to K and on to destination D following the *nominal* routing tree for destination D. Therefore, there is no looping problem and the traffic is rerouted without causing any conflict.



(a) A nominal routing tree to destination D.



(b) A Failure in link SD disconnecting the nominal routing tree in the red part on the left and the blue part on the right.



(c) The traffic is rerouted from the red part to blue part via alternative paths using bridges like (U, V).

Fig. 6: An example of rerouting scheme with Algorithm 1.

B. Algorithm 1

The method employed to build rerouting paths ensuring the existence proof for Theorem 1 stated above is formalized in Algorithm 1 as follows:

With respect to the above example (Fig. (6c), running Algorithm 1 will give $D = q_r, S = p_r, U = i, V = j$. Hence, path $\mu_r^1 = \{S \to T \to U\}$ and $\mu_r^3 = \{V \to K \to D\}$ and bridge $\{U \to V\}$.

Algorithm 1:

Input: a directed, symmetric two-link connected network; a routing tree A, $|A| = n$, destination l which is the sink of the reverse tree A;
Output: a free-conflict routing scheme;

Number the arcs of the routing tree such that their number decreases in value as we approach the sink l: $r \in \{1...n-1\}$. Failures are considered in this order;
for *($r = 1$* **to** $n-1$*)* **do**
  Let $(p_r,\ q_r)$ be an arc of the tree and A_r the sub-tree of sink p_r;
  Compute a bridge $(i,\ j)$ connecting A_r to $A - A_r$. This bridge is chosen so that there is no conflict with $\mu_1, \mu_2, \ldots, \mu_{r-1}$;
  Compute the path μ_r^1 from p_r to i, following the inverse path of the nominal routing tree;
  Set the rerouting path of $(p_r,\ q_r)$ as : $\mu_r = \mu_r^1 + (i,\ j) + \mu_r^3$ where μ_r^3 is the path of the routing tree going from j to l;

C. Algorithm 2

The main drawback of Algorithm 1 is its lack of flexibility. Given the rerouting path, the only degree of freedom that it can offer is the choice of the bridge. Furthermore, the free-conflict constraint doesn't permit any alternative for the third part of the path. Hence, the only possible variation is in its first part. We show that under some assumption it is always possible to build the first part of the path in a more general way (not only following the inverse path of the nominal routing path). Nevertheless, the free-conflict constraint reduces at a certain point this choice as shown in the algorithm 2. In algorithm 2, we initialise a rerouting path which traverses only the nodes in the red part of the routing tree and does not use any arc in the red part of the routing tree (Figure 7). Then, this path follows the nominal routing tree in the blue part. Next, we look if this rerouting path has any conflict with rerouting ones get previously. In case of conflict, we modify this rerouting path by merging with those in conflict. At the end of this algorithm, we remove any directed cycle in the rerouting path to ensure the elementarity of this path. With respect to the example presented in Figure 6, one can apply Algorithm 2 by taking $D = q_r, S = p_r, Q = i, K = j$. Hence, path $\mu_r^1 = \{S \to T \to G\}$ and $\mu_r^3 = \{K \to D\}$ as shown in figure 7.

We have stated and proved the Theorem 2 to validate the above algorithm. We shall make the same assumptions as before: the graph is assumed to be oriented symmetric; there are at least two disjoint-arc paths between any two nodes of the graph; only one link failure can occur at a time.

Theorem 2. *The rerouting paths computed sequentially by algorithm 2 are elementary and without conflict.*

Proof. Let A be the routing tree to the destination l. l is therefore the sink of A. Let (p_k, q_k) be an arc of A. At first, we will explain how to build a rerouting path for the failure of arc (p_k, q_k). Next, we will modify this rerouting path appropriately in an iterative scheme. We assume that

Algorithm 2:

Input: a bidirected, symmetric two-link connected network; a routing tree A, $|A| = n$, destination l which is the sink of the reverse tree A;
Output: a free-conflict routing scheme;

Number the arcs of the routing tree such that their number decreases in value as we approach the sink l: $r \in \{1...n-1\}$. Failures are considered in this order;
for *($r = 1$* **to** $n-1$*)* **do**
  (Path initialization:)
  Let $(p_r,\ q_r)$ an arc of the tree and A_r the subtree of sink p_r;
  Compute a bridge $(i,\ j)$ of A_r where A_r is the subtree of sink p_r;
  Compute a path μ_r^1 from p_r to i, traversing only nodes of A_r and not borrowing arcs of A_r;
  Set the rerouting path of $(p_r,\ q_r)$ as : $\mu_r = \mu_r^1 + (i,\ j) + \mu_r^3$ where μ_r^3 is the path of the routing tree going from j to l;
  (Conflict avoidance:)
  if *μ_r is in conflict with one of* $\mu_1, \mu_2, \ldots, \mu_{r-1}$ **then**
    Denote μ_k the first rerouting path in conflict with μ_r;
    Let (a, b) be the first arc in common;
    Let denote with μ_r^5 and μ_k^5 respectively the sub-paths of μ_r and μ_k from b to l. These sub-paths bifurcate.
    Set μ_r^4 as the sub-path of μ_r from p_r to b;
    Set the rerouting path $\mu_r = \mu_r^4 + \mu_k^5$;
  Ensuring path elementarity:
  if *(μ_r is not elementary)* **then**
    Remove the cycle included in μ_r;

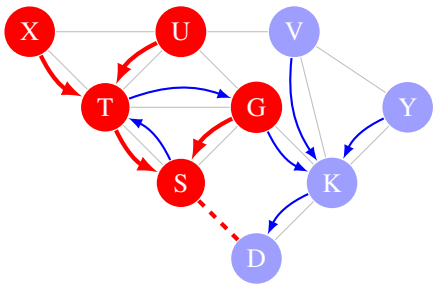


Fig. 7: Rerouting scheme - Algorithm 2: with respect to the example presented in figure 6, one can apply Algorithm 2 by taking $D = q_r, S = p_r, Q = i, K = j$. Hence, path $\mu_r^1 = \{S \to T \to G\}$ and $\mu_r^3 = \{K \to D\}$.

the link corresponding to this arc fails and we must find a rerouting scheme without conflict. p_k is the sink of a sub-tree A_k whose nodes are colored in red. The other nodes in the tree are colored in blue. Without loss of generality we assume that all nodes belong to the tree A and this is true for any destination l. We will also show that the constructed rerouting path is elementary.

We know that there are at least two disjoint paths in the initial graph going from p_k to l. Given the assumption of two-link connectivity, the upper part that includes p_k, and does not include the vertex l must contain at least two outgoing arcs. Therefore there are at least two arcs going out of A_k (Figure 8). Since one of these arcs is (p_k, q_k), there exists a path μ_k from p_k that visits the vertices of A_k, and connects a vertex of A_k, which is red, to a blue vertex. So there is at least one arc (i, j) of μ connecting the Red vertices to the Blue vertices (Figure 8). This arc is a bridge of A_k. We associate with this arc an elementary rerouting path for the failed arc. Let v be a red vertex of A_k which is affected by the failure. Then, the traffic to destination l and coming from v goes first to p_k, then it follows a rerouting path to destination l. This rerouting path is composed by 3 parts: a path μ_k^1 from p_k to vertex i, a path μ_k^2 that contains only the bridge (i, j) and a path μ_k^3 from j to destination l follows the original

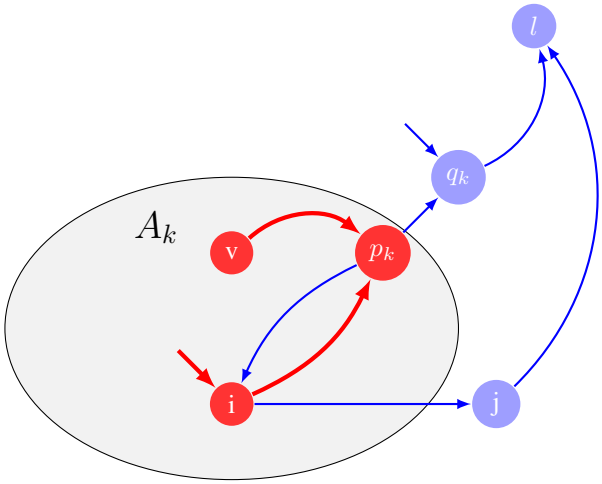


Fig. 8: Path initialisation - Algorithm 2: the rerouting path is composed of three parts: a path joining p_k to i, arc (i, j) and the nominal routing path from j to l.

We explain first how to modify the rerouting paths to avoid any conflict. We number the arcs of the tree so that their numbers decrease in value as we approach the sink l, and consequently choose the rerouting paths in successive order of increasing numbers. For the first failed arc (p_1, q_1), we construct the rerouting path μ_1 like above, so μ_1 is elementary by construction. We consider that we have already computed the rerouting paths for the first $r-1$ arcs in the tree $(r \geq 2)$. By the recurrence hypothesis, there is no conflict for the first $r-1$ reroutings and these paths are elementary. We construct now the r-th rerouting so that it has no conflict with the first $r-1$ reroutings and that it is elementary. Let (p_r, q_r) be the arc under consideration. p_r is the sink of tree A_r. Now we

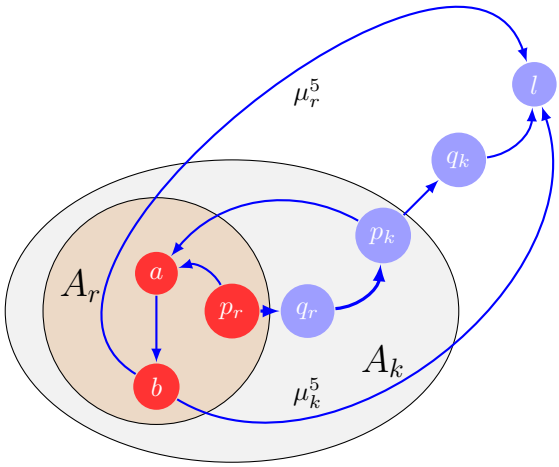


Fig. 9: Conflict avoidance - Algorithm 2: there is a conflict when traffic rerouted from p_k and the one rerouted from p_r go through arc (a, b) and bifurcate next. The rerouting path associated with failure of arc (p_r, q_r) is then composed of sub-path of μ_r from p_r to b and μ_k^5.

initialize μ_r as explained above. There are two cases to be considered with this new path. In the first case, μ_r has no conflict with the paths $\mu_1, \mu_2 \ldots \mu_{r-1}$ chosen before. We have nothing to do with μ_r in this case. In the second case, there is a conflict, so there is an existing rerouting path that has an arc in common with μ_r. Let $(a,\ b)$ be the first arc in common between μ_r and the other rerouting paths and $\mu_k, (k < r)$ be the first rerouting path among them having this property.

Notice first that there are only two possibilities for subtrees A_k and A_r; either $A_r \subset A_k$ (the arc (p_k, q_k) is included in the nominal routing path from p_r to l), either A_r and A_k are disjoint. In the last case (a, b) is included in μ_k^3 and μ_r^3 and the rerouting paths are necessarily without conflict. In fact, the two rerouting paths will follow the original routing tree to destination l so there will be no conflict between them and the path is elementary. Consider now the case $A_r \subset A_k$. If (a, b) doesn't belong to μ_k^1, then the same reasoning as above applies and the rerouting path is elementary and without conflict.

Let consider the case when (a, b) belongs to μ_k^1. This implies that (a, b) doesn't belong to μ_r^3. We define then μ_r^4 the part of μ_r from p_r to arc (a, b) and μ_r^5 the rest of μ_r from b to destination l. In the same way, we define μ_k^4 and μ_k^5. If there is no vertex in common between μ_r^4 and μ_k^5, we reconstruct then the new path μ_r' by concatenating μ_r^4 and μ_k^5. We use this new path to reroute the traffic for the failure $(p_r,\ q_r)$ (Figure 9). This path is without conflict from definition of (a, b) as the first arc in common with other rerouting paths. This path does not borrow the failed arc (p_r, q_r) because $A_r \subset A_k$ and the path μ_k is supposed not to borrow any arc of A_k. In case when there is a vertex in common in μ_r^4 and μ_k^5, let m be the first one for example, we construct a new path μ_r' that uses μ_r^4 from p_r to m

and uses μ_k^5 starting from the vertex m to destination l. Clearly, there is no vertex in common between the part of μ_r^4 and μ_k^5 used by the new path μ_r'. So, μ_r' is elementary. We notice that this rerouting path uses the bridge of rerouting path μ_k. We can show also that this contraction will yield to no conflict. In fact, we only need to consider if there is a conflict for the arc of this new path μ_r which has m as the end vertex, we suppose this arc is (n, m). Let suppose, by absurdity, that μ_r has conflict with another rerouting path at this arc (n, m), which means (n, m) is the arc in common between μ_r and another rerouting path. This is in contradiction with the hypothesis that (a, b) is the first arc in common with other rerouting paths. Then, we can conclude that this contraction of μ_r is without conflict. We have therefore demonstrated by recurrence the absence of conflict and the property of elementarity of the rerouting paths. Finally, the path does not borrow the failed arc $(q_r,\ p_r)$ because it is elementary. □

Difference between our method and the related works

We have presented the method of Xi et Chao and of Nelakuditi et al. in the related works section. Now that our method has been presented, we explain the difference between our method and this method. It has some similarity with our first method. The method of Xi et Chao chooses the first bridge that is available. Consequently, their method has no flexibility concerning the bridge choice. The method of Nelakuditi et al. is the particular case of our second method where the shortest paths are chosen. Consequently, it is identical to Xi et Chao method. Their limitation will be seen clearly in the numerical results presented in the section below. Concerning other works in relation with SDN, we notice that they require modifying the packet headers which is not our case.

IV. Numerical results

Let us note first that the routing scheme proposed in this study requires the single path routing of traffic demands both in the nominal situation and in situations of failure. An additional difficulty for this strategy comes from the conflict avoidance constraint. All this makes modelling and solving using an arc-path flow formulation extremely hard. We have therefore opted for an adapted variant of the arc-node formulation that is better suited to expressing this type of constraints. At this stage, the mathematical formulation concerns only the spare-capacity assignment problem, (that is computing the minimum capacities to be added for recovery needs, the routing being given). The full model, given in the appendix, deals with link failure situations, node failure situations can be handled in a similar way.

Given the above algorithms we have built two heuristic approaches to solve the dimensionning problem. The main idea behind the heuristics is to compute the minimum capacity added to each arc such that the traffic routing and failure rerouting can be done simultaneously. To do this we have examined all destinations following a decreasing order of the amount of traffic transported to each destination. Of course, these approaches give non optimal results. Then in order to assess the performance of both heuristics we have also written and run a MIP (Mixed Integer Program) model for the dimensioning problem (the detailed formulation is given i Appendix). We ran our program on IBM ILOG OPL IDEA using IBM ILOG CPLEX 12.1.0. The calculations were performed on a virtual machine with the following configuration: Quad Core 1.8GHz, 8.00 Go RAM, 12Mb cache.

Network	Nodes	Links	Demands
Test	7	9	42
Polska	12	18	66
Atlanta	15	22	210
Nobel-Germany	17	26	121
France	25	45	300
India35	35	80	595
Pioro40	40	89	780
Germany50	50	88	662

TABLE I: Network instances.

Network Instance	Exact Method	Heuristic 1	Heuristic 2	Xi and Chao
Test	62	66	68	69
Polska	19110	21449	21949	25093
Atlanta	308171	333480	343969	330745
Nobel-Germany	1862	1980	1940	2744
France	OM	261529	260451	416670
India35	OM	7874	7784	11689
Pioro40	OM	289168	279046	431332
Germany50	OM	7339	7453	9847

TABLE II: Network cost.

Network Instance	Exact-method without "free-conflict constraints"	Exact method
Test	0.83	0.96
Polska	0.87	0.9
Atlanta	1.06	1.08
Nobel-Germany	1.08	1.16

TABLE III: Evaluating the additional cost due to requiring free-conflict constraints.

In order to evaluate the effectiveness of the above heuristics, we tested them on 8 network instances presented in Table I. Table II gives the numerical results obtained with the exact model, two heuristics and the method of Xi and Chao [8]. Notation OM (Out of Memory) stands for cases when the program stops without reaching a solution because of memory overflow. We can remark from this table that the results for both heuristics are close to those obtained with the MIP model (the gap is between 6% and 12%). We notice that Heuristic 2 performs generally better than Heuristic 1 for large instances (except for the last one). Heuristic 1 finally performs quite well given its simplicity. Borrowing arcs of the inverse path results to be a good strategy as such arcs will be taken by all rerouting paths used for failures in the same routing path. We have discussed the similarity of our method and the method of Xi et Chao [8] in previous sections, so now we compare our method with this method. We recall that the rerouting path always borrows the first bridge in [8] and when two rerouting paths have the same node in common, they will fusion until they reach the destination. Obviously, their method is more rigid than our methods and it did not take into account capacity

optimization. We can see from the numerical results that our methods perform better than [8] except for Atlanta network. Nevertheless, the difference between the two solutions for the Atlanta network remains small (around 1%), while our methods perform much better for all other instances. There is no clear reason why such behavior happens for the Atlanta network. Still, we are comparing here between heuristics, the stability of the results is not guaranteed, so no firm conclusions can be drawn from a single instance while the tendency is clear in favor of our proposed algorithms. What is encouraging is that it performs much better with gaps going from 25% to 37% for the last (5) larger network instances.

The second serie of tests is concerned with specificity of the model which stands mainly in forcing the rerouting to satisfy the *free conflict cosntraint*, which makes the main difference with the conventional single path rerouting. Hence, we have tested the impact of this constraint in terms of link capacity cost. Table III has 4 columns: the network instance, the ratio capacity of exact method without "free-conflict constraints", the ratio capacity of our mathematical model. The ratio capacity column describes the ratio between the added capacity and the installed routing capacity. The tests were run on the above networks and the routing was considered to be fixed. We used the shortest paths to fix nominal routing for our method. We can see from the results that the cost of the free-conflict constraints is not highly expensive for our network dimensioning, while it leads to a very simplified rerouting scheme in terms of management cost.

V. Conclusion

We have presented a rerouting approach that can handle link failures in a network of switches. The proposed method is based on the local reaction of nodes at the extremities of the failed link, while the other nodes do not need to know or take any particular action. This makes implementation particularly easy. We have proved that there exists a restoration scheme without conflict in the network and have also enhanced this resilient routing scheme. Then we have provided a mathematical model capable of calculating the rerouting scheme and optimizing the sum of additional capacities. Moreover, we have proposed the heuristics which permit to solve this issue for larger instances. Our method can be extended for single node failure issues in switch-based network but not for multi-link failures [5]. Future work is needed to extend computational methods for single node failure problem and evaluate them. It could be interesting to study the influence of network topology for single link failure problem.

Appendix

We propose below an integer linear porgramming formulation of the problem.

Parameters:

- Arc: set of arcs of the graph.
- E: set of links.
- V: set of nodes.
- Triple: all triples (i, k, j), where i, k, j are nodes of the graph and (i, k) and (k, j) are two adjacent arcs, i being different from j.
- 0: a fictitious node used to divert traffic in case of failure. We introduce the fictitious node 0 that will be used for all failures. For a given failure $(v,\ l)$, the traffic to l will be rerouted along a single path from 0 to l and starting with the arc $(0,\ v)$.
- T_v^l: total traffic to l that passes through the node v, v being the node that detects the failure. In fact, the failure is characterized by a source v and a destination l. This is because the nominal routing is done through a tree, and we need to reroute only the nominal traffic passing via this tree. Hence, for a destination l, each node in the tree is concerned with only one (failure) arc in the tree, and this node is necessarily the origin of the failed arc.
- A^l: set of arcs of the routing tree to the destination l.
- red_v^l: sub-tree of tree A^l with sink v. Recall that in the case of failure the tree is divided into two parts: the isolated part, i.e. the red part, and the blue part. The alternative path will reroute traffic from the node initiating the rerouting in the red part to the destination in the blue part.
- $blue_v^l$: $A^l - red_v^l$.
- α_{ab}^{lv}: a binary coefficient equal to 1 if the arc (a, b) belongs to the nominal routing path from v to l excluding the failed arc $(v,\ w)$.

Decision variables:

- y_{ikj}^{lv}: A binary variable that indicates whether the alternative path coming from v and going to destination l contains arcs (i, k) and (k, j), where node v is the node that detects the failure.
- x_{ikj}^{l}: A binary variable taking the value 1 if there exists a failure whose alternative path to destination l contains arcs (i, k) and (k, j). In other words, this variable takes the value 1 if there exists v that y_{ikj}^{lv} is equal to 1.
- r_{ab}: additional capacity assigned to the arc (a, b).
- $N(k)$: set of nodes which are neighbours of node k.

Given the above the mathematical model follows:

The objective function which minimizes the sum of additional capacity allocated to each arc, follows:

$$\min \sum_{(a,b)\in Arc} r_{ab} \tag{1}$$

(1) will allow us to evaluate the ratio between the additional capacity and the installed capacity.

a) Flow and rerouting constraints: The constraints in this paragraph are the constraints of rerouting. They will be able to construct a rerouting path from the node fictive 0 to destination l (Figure 10).

$$\sum_{vv_1\notin A^l,\ v_1\in N(v)} y_{0vv_1}^{lv} = 1,\ v \in V,\ l \in V \tag{2}$$

Constraints (2) implie that there exists exactly one arc coming out of v to be used for the rerouting of the disturbed traffic from v to l. As we describe above, all the traffic will be rerouted by only one path. This path begins by the fictive node 0, then it uses the node v which is the node that detects the failure and has to reroute the traffic (Figure 10). These constraints allow to control the number of outgoing arcs from

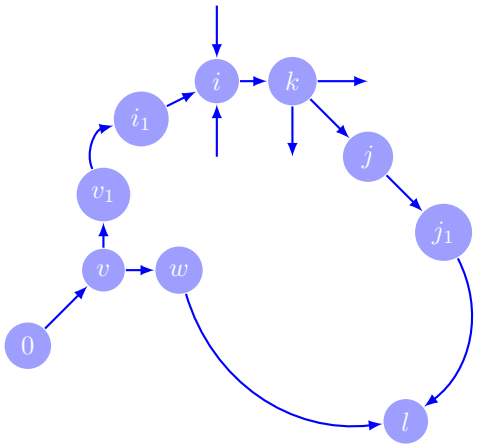


Fig. 10: Model illustration: there is a fictive node 0 associated with arc failure (v, w). Then the path will go through v to $v_1, \ldots, l$..

the node v. If arc (v, v_1) is not in the nominal routing tree to destination l, this arc is not the failed one. This ensures that the rerouting path doesn't pass by the failed arc.

To avoid the problems of looping and conflict, the alternative path should not contain any arc of nominal routing in the red part of the network.

Constraints (3):

$$y^{lv}_{ikj} = 0,\ l \in V,\ v \in V,\ i \in red^l_v,\ (i,\ k) \in A^l,\ (i,\ k,\ j) \in Triple \tag{3}$$

ensure that this condition holds. Indeed, when the rerouting path uses one arc of the nominal routing tree, it has to continue to use the arcs of the nominal routing tree until it reaches the destination l. If this arc is in the red part of the network, the rerouting path will follow the nominal routing tree to the failed arc. This will cause the looping problem and the traffic will not be rerouted to the destination. We add the following constraint to avoid this problem:

$$\sum_{i \in V, j \in V | (i,k,j) \in Triple} y^{lv}_{ikj} \leq 1, l \in V,\ v \in V,\ k \in V \tag{4}$$

Constraints (4) ensure that there will be no looping in the network, since the alternative path can transit once at the most via any given node.

We will introduce then the flow constraints that ensure the continuity of the alternative path.

$$\sum_{i_1 \in N(i)} y^{lv}_{i_1 ik} = \sum_{j \in N(k)} y^{lv}_{ikj},\ l \in V,\ (i,k) \in Arc, i \neq 0\ , i \neq v, k \in V \backslash \{l\} \tag{5}$$

$$\sum_{j \in N(k)} y^{lv}_{vkj} = y^{lv}_{0vk},\ l \in V,\ v \in V,\ (v,k) \in Arc \tag{6}$$

$$y^{lv}_{kjj_1} - y^{lv}_{ikj} \geq 0,\ j \in blue^l_v \backslash \{l\},\ (j, j_1) \in A^l,\ (k,j) \in A^l, \forall l \in V, \forall (i,k,j) \in Triple \tag{7}$$

Constraints (5) are the flow conservation constraints. With respect to the traffic passing by arc $(i,\ k)$, all traffic entering node i equals the total traffic leaving node k (Figure 10). For the node v, the entering traffic is supposed to come from the fictive node 0. We therefore have a flow constraint (6) for this case. In the blue part, if the path uses an arc of the initial routing, it must continue to destination l, and we thus have constraints (7).

Next, we need to guarantee that the rerouting path reaches destination l.

$$\sum_{(i,\ k,\ l) \in Triple} y^{lv}_{ikl} = 1,\ l \in V,\ v \in V \tag{8}$$

Constraints (8) ensure that there is a single rerouting path that will reach the destination l. These constraints consequently ensure the single path routing requirement.

Given the complexity of the above LP formulation, we believe that it will be useful to briefly discuss the validity of the model. We recall first that the traffic to be rerouted is characterized by the couple $(l,\ v)$. We list below all the requirements that a rerouting path from v to l must satisfy:

1) The path does not use the nominal routing when it goes through the red sub-tree of the network.
2) If the path uses an arc of the nominal routing when it is in the blue sub-tree of the network, it must continue to use the nominal routing until it reaches the destination l.
3) It is an elementary path.

Clearly, constraint (3) ensures the first requirement and constraint (7) ensures the second. The third requirement comes first from constraints (2) and (8) specifying constraints for the origin (v) and the destination (l), secondly from constraints (4) that ensure the elementarity of the path, and finally from the Kirchoff constraints.

b) Avoiding conflict constraints: After the construction of the rerouting paths, we have to assure that these rerouting paths have no conflict between them.

$$\sum_{v \in V} y^{lv}_{ikj} \geq x^l_{ikj} \geq \frac{\sum_{v \in V} y^{lv}_{ikj}}{|V|},\ (i,k,j) \in Triple,\ l \in V \tag{9}$$

$$\sum_{j \in neighbor\ of\ k} x^l_{ikj} \leq 1,\ (i,\ k) \in Arc,\ l \in V \tag{10}$$

$$x^l_{ikj} \in \{0,\ 1\}, \forall (i,k,j) \in Triple, \forall l \in V \tag{11}$$

$$y^{lv}_{ikj} \in \{0,\ 1\}, \forall (i,k,j) \in Triple, \forall l \in V, \forall v \in V \tag{12}$$

Constraints (9), (10), (11), (12) ensure the absence of conflict. The constraints (11), (12) express that the variables x^l_{ikj} and y^{lv}_{ikj} are binary. Constraint (9) expresses the fact that $x^l_{ikj} = 1 \Leftrightarrow$ There exists $v : y^{lv}_{ikj} = 1$. Indeed, if $x^l_{ikj} = 1$, it is obvious that there exists a v such that $y_{ikj} = 1$, because $\sum_v y^{lv}_{ikj} \geq x^l_{ikj}$. And vice-versa, if there exists $v : y^{lv}_{ikj} = 1$, then $\sum_v y^{lv}_{ikj} \geq 1$ and consequently $\frac{\sum_v y^{lv}_{ikj}}{|V|} \geq \frac{1}{|V|}$. We can

deduce that $x^l_{ikj} \geq \frac{1}{|V|}$. Because $x^l_{ikj} \in \{0,1\}$, we have $x^l_{ikj} = 1$.

Next, the constraint of absence of conflict can be expressed by (10):
$\sum_{j \in N(k)} x^l_{ikj} \leq 1$, because if we use arc $(i,\ k)$ for the alternative path, we need only to use at most one arc $(k,\ j)$. Indeed, when two rerouting paths use the same entering arc $(i,\ k)$, the constraint ensures that it could use at most one outgoing arc $(k,\ j)$, which means there is no conflict.

c) Capacity constraints: In this paragraph, we present the constraints that concern the capacity added to the arc for any link failure $(v,\ w)$ in tree A^l.:

$$\begin{aligned} &\sum_{l \in V|(v,w) \in A^l} \sum_{i \in N(k),\ i \neq j} y^{lv}_{ikj}.T^l_v \\ &+ \sum_{l \in V|(w,v) \in A^l} \sum_{i \in N(k),\ i \neq j} y^{lw}_{ikj}.T^l_w \leq r_{kj} \\ &+ \sum_{l \in V|(v,w) \in A^l} \alpha^{lv}_{kj}.T^l_v + \sum_{l \in V|(w,v) \in A^l} \alpha^{lw}_{kj}.T^l_w, \\ &(k,j) \in Arc,\ k \neq v,\ k \neq w,\ (v,w) \in E \end{aligned} \tag{13}$$

$$\begin{aligned} &\sum_{l \in V|(v,w) \in A^l} y^{lv}_{0vj}.T^l_v \leq r_{vj} + \sum_{l \in V|(v,w) \in A^l} \alpha^{lv}_{vj}.T^l_v, \\ &(v,\ j) \in Arc,\ j \neq\ w,\ (v,w) \in E \end{aligned} \tag{14}$$

$$\begin{aligned} &\sum_{l \in V|(w,v) \in A^l} y^{lw}_{0wj}.T^l_w \leq r_{wj} + \sum_{l \in V|(w,v) \in A^l} \alpha^{lw}_{wj}.T^l_w, \\ &(w,j) \in Arc,\ j \neq\ v,\ (v,w) \in E \end{aligned} \tag{15}$$

For each failure of link $(v,\ w)$, constraints (13) consider rerouted paths for both arcs $(v,\ w)$ and $(w,\ v)$, and only trees that contain the arc failure are involved. They also take into account the released bandwidth on the initial routing paths. Indeed, the first term of the left part of the constraint (13) signifies the capacity of traffic rerouted by the failure of arc $(v,\ w)$ that passes by arc $(k,\ j)$ and the second one signifies the capacity of traffic rerouted by the failure of arc $(w,\ v)$. In addition, the first term of the right part of the constraint (13) signifies the capacity added to arc $(k,\ j)$ while the second one and third one signifies the capacity released for arc $(k,\ j)$ if this arc is on the nominal routing paths for the failure of arc $(v,\ w)$ and $(w,\ v)$ correspondingly. In brief, this constraint gives us the bound of the capacity added to arc $(k,\ j)$. (14) and (15) are special cases of (13) for the nodes that detect the failure, that is v and w.

REFERENCES

[1] RFC 5286, *Basic Specification for IP Fast Reroute: Loop-Free Alternates*, http://tools.ietf.org/html/rfc5286, 2008.

[2] Internet-Draft, *IP Fast Reroute Using Not-via Addresses*, https://tools.ietf.org/html/draft-ietf-rtgwg-ipfrr-notvia-addresses-11,2013.

[3] Ho, K.-H., Wang, N., Pavlou, G., Botsiaris, C., *Optimizing Post-Failure Network Performance for IP Fast Reroute using Tunnels*, QShine'08, Proc. of the 5th International ICST Conference on Heterogeneous Networking for Quality, Reliability, Security and Robustness, Hong Kong, 2008.

[4] Kvalbein, A., Hansen, A., Cicic, T., Gjessing, S., Lysne, O., *Fast IP Network Recovery using Multiple Routing Configurations*, INFOCOM 2006, pp. 1-15, April, 2006.

[5] Pham, T.S., PhD Thesis, *Autonomous management of quality of service in virtual networks*, University of Technology of Compiegne, Nov. 2014.

[6] Wang, J., Nelakuditi, S., *IP Fast Reroute with Failure Inferencing*, Proceedings of the 2007 SIGCOMM workshop on Internet network management, pp. 268-273, ACM New York, USA, 2007.

[7] Medard, M., Finn, S.G., Barry, R.A., Gallager, R.G., *Redundant trees for preplanned recovery in arbitrary vertex-redundant or edge-redundant graphs*, IEEE/ACM Trans. on Networking, vol. 7, issue 5, pp. 641-652, Oct, 1999.

[8] Xi, K., Chao, J., *IP Fast Rerouting for single-link/node failure recovery*, BROADNETS 2007, Fourth International Conference on Broadband Communications, Networks and Systems, pp. 142-151, Raleigh, NC, USA, Sept, 2007.

[9] Kamamura, S., Shimazaki, D., Hiramatsu, A., Nakazato, H., *Autonomous IP Fast Rerouting with Compressed Backup Flow Entries Using OpenFlow*, IEICE TRANSACTIONS on Information and Systems, Vol.E96-D, No.2, pp.184-192, February, 2013.

[10] Stern, T.E., Bala, K., *Multiwavelength Optical Networks: A Layered Approach*, Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1999.

[11] Zheng, Q., Shin, K.G., *Fault-tolerant real-time communication in distributed computing systems*, IEEE Trans. Parallel Distrib. Syst, vol. 9, issue 5, pp. 470-480, May 1998.

[12] Yang, H., Zhang,J., Zhao, Y., Li,H.,Huang,S.,Ji,Y.,Han,J., Lin,Y.,Lee,Y., *Cross stratum resilience for OpenFlow-enabled data center interconnection with Flexi-Grid optical networks*, Optical Switching and Networking, pp. 72-82, Volume 11, Part A, January 2014.

[13] Raeisi, B. and Giorgetti, A., *Software-based fast failure recovery in load balanced SDN-based datacenter networks*,6th International Conference on Information Communication and Management (ICICM), pp. 95-99, Hatfield, 2016.

[14] Chen, X., Zhao, B., Ma, S., Chen, C., Hu, D., Zhou, W., and Zhu, Z., *Leveraging master-slave OpenFlow controller arrangement to improve control plane resiliency in SD-EONs*, Opt. Express 23, Issue 6,pp. 7550-7558, 2015.

[15] Capone, A., Cascone, C., Nguyen, A. Q.T., and Sanso, B., *Detour Planning for Fast and Reliable Failure Recovery in SDN with OpenState*, DRCN 2015, Kansas City, March 2015.

[16] Mijiddorj, D., Tarigan, I.D.F., Kim, D.-S. *Fast-Failover Mechanisms using Parenthood Distribution Controllers*, KICS Summer Conference 2016,pp.212-213, Jeju Island, South Korea,June 2016.

[17] Chu, C. Y. ,Xi, K., Luo, M. and Chao, H. J.,*Congestion-aware single link failure recovery in hybrid SDN networks*, IEEE Conference on Computer Communications (INFOCOM), pp. 1086-1094, Kowloon, 2015.

[18] Borokhovich, M., Schiff, L. and Schmid, S., *Provable data plane connectivity with local fast failover: introducing openflow graph algorithms*, The third workshop on Hot topics in software defined networking (HotSDN '14). ACM,pp. 121-126, New York, USA,2014.

[19] Katta, N., Zhang, H., Freedman, M. and Rexford, J.,*Ravana: controller fault-tolerance in software-defined networking*, The 1st ACM SIGCOMM Symposium on Software Defined Networking Research (SOSR '15). ACM, New York, USA, 2015.

[20] Chiesa, M., Nikolaevskiy, I., Mitrovic, S., Gurtov, A., Madry, A., Schapira, M., and Shenker, S., *On the Resiliency of Static Forwarding Tables.* IEEE/ACM Trans. Netw. 25, 2, 1133-1146, April 2017.

[21] Holterbach, Th., Vissicchio, S., Dainotti, A., and Vanbever, L., *SWIFT: Predictive Fast Reroute.* In Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '17). pp. 460-473, ACM, New York, NY, USA, 2017.

[22] Kreutz, D., Ramos, F. M. V., Verissimo, P. E., Rothenberg, C. E., Azodolmolky, S., and Uhlig, S., *Software-Defined Networking: A Comprehensive Survey,* in Proceedings of the IEEE, vol. 103, no. 1, pp. 14-76, Jan. 2015.

[23] C. Filsfils, S. Previdi, L. Ginsberg, B. Decraene, S. Litkowski, R. Shakir., *Segment Routing Architecture.* Request for Comments: 8402, July 2018, https://www.rfc-editor.org/authors/rfc8402.txt.