Journal of Computer Science

Original Research Paper

A Conflict-Free Routing Tables Update Method for Persistent Multilink and Node Failures in SDN Architectures

[1]Yannick Florian Yankam, [2]Jean Frédéric Myoupo and [1]Vianney Kengne Tchendji

[1]Department of Mathematics and Computer Science, University of Dschang, Dschang, Cameroon
[2]Computer Science Lab-MIS, University of Picardie Jules Verne, Amiens, France

Article history
Received: 21-12-2018
Revised: 31-12-2018
Accepted: 11-03-2019

Corresponding Author:
Jean Frédéric Myoupo
Computer Science Lab-MIS,
University of Picardie Jules
Verne, Amiens, France
Email: jeanfrederic.myoupo@u-picardie.fr

Abstract: The large-scale network management abilities of centralized architectures, such as Software-Defined Networking (SDN), are attracting increasing interest from major computer networking companies. In this architecture type, the nodes follow the rerouting rules predefined by the control plane in the controller. However, when a link or node failure becomes persistent over the time, the controller must recompute the routing and rerouting rules and update the relevant nodes to maintain an acceptable quality of service (QoS). In this study, we propose a conflict-free mechanism for updates of routing and rerouting tables of nodes by the controller. This mechanism works without disrupting the current traffic if both persistent multilink and node failures occur. We describe an efficient strategy for choosing the nodes to update and define an update scheme for these nodes. We show through the simulations of our updating scheme on various networks that our strategy improves QoS by reducing packet routing delays and the data loss rate in case of persistent multilink and node failures in the network. We also compare our results to those of several existing studies.

Keywords: Network Virtualization, Software-Defined Networking (SDN), Centralized Architecture, QoS, Network Recovery

Introduction

For a long time, the difficulty of managing largescale network infrastructure (addressing, bandwidth, throughput, etc.) has been increasing in computer networks. The adoption of network virtualization technology has made this problem increasingly urgent and timely. The reason is that network virtualization allows easy creation of virtual networks using the virtual components built from an already existing physical IP network. However, in recent years, the Software-Defined Networking (SDN) approach has been presented as the most suitable solution for such a problem (Farhady *et al*., 2015; Hu *et al*., 2014). In this technology, the control plane (Hu *et al*., 2014) is decoupled from the data plane. The control plane is the part of the network that defines the network management policies, while the data plane merely forwards data according to the rules defined by the control plane. The latter is centralized within the main equipment called the controller and the data plane (Hu *et al*., 2014) is kept on network devices (e.g., switches and routers). Such decoupling of planes provide flexibility and programmability, satisfying the cloud automation needs and making it easier to add new services in communication networks (Azodolmolky *et al*., 2013). The controller that hosts the control plane defines network management policies, e.g., the routing rules and integrates them inside network devices. When a failure occurs in the network, the devices forward the packets according to the rules defined in their flow tables (Rothenberg *et al*., 2012; Pham, 2014; Papan *et al*., 2017) in advance by the controller. This is the IPFRR (IP Fast ReRoute) scheme known for its fast rerouting capabilities.

However, when a failure has been deemed persistent, these rules become obsolete and no longer allow for a good QoS in the network. The network QoS mostly suffers in case of multilink and node failures. A failure is determined to be persistent if its duration is more than ten minutes (Pham, 2014). In this case, the preceding routing rules are no longer optimal and become obsolete. Then, it is necessary to recalculate the routing paths and update the routing tables of the devices. The most important challenges to be overcome by the controller in this failure scenario are the following:

- Construct new routing and rerouting rules based on the existing rules, to handle the failures
- Select the nodes to update

- Choose the efficient update order that will be applied to the nodes; in particular, it needs to be decided whether all the involved nodes are to be updated simultaneously or sequentially
- Decide whether to send all the computed rules at the same time; this challenge concerns the packet size and the maximum transmission unit (MTU) value of the network's interfaces; in large-scale networks, the packet size of the updates can increase easily with the number of nodes

Related Studies

A review of the routing tables update problem in the literature yields few relevant ideas that have already been stated in basic routing protocols, such as Open Shortest Path First (OSPF) (Moy, 1998), Routing Information Protocol (RIP) (Perkins *et al*., 2003; Hedrick, 1988) and Enhanced Interior Gateway Routing Protocol (EIGRP). Perhaps this is why, to the best of our knowledge, many of the centralized (Muruganathan *et al*., 2005), distributed (Moy, 1998; Perkins *et al*., 2003) and even hybrid (Rothenberg *et al*., 2012; Lim *et al*., 2008) rerouting schemes provided in computer networks simply define the rerouting paths calculation and do not specifically address the routing tables update challenge. Nevertheless, the above distributed protocols are not individually efficient in centralized architectures, such as SDN using the OpenFlow protocol (Pham, 2014).

Lim *et al*. (2008) proposed a rerouting scheme based on a tree topology computation in Wireless Mesh Networks (WMN). In that tree-based routing scheme, numerous messages are exchanged between the root node and its child nodes when a routing table update is needed in case of a link or a node failure.

Moreover, each node that needs to update its routing table must send a Route Request (RREQ) message to the root first; the root replies with an update message packet called Route Set (RSET) (its structure is shown in Fig. 1). In a large-scale network, this approach could be very harmful.

In a centralized SDN architecture, when a persistent link failure has been identified, Pham (2014) suggests an update strategy based on an IPFRR (Papan *et al*., 2017) mechanism that avoids looping. In the cited study, the authors propose updating the nodes from the destination of a routing tree to the node that detected the failure. However, this scheme has the following limitations:

- Overloading of the rerouting path, which may affect the handling of other link failures and load balancing in the network
- Involving nodes for which an update is not necessarily needed; this increases the conflict risks to be managed in the rerouting path
- Only stating the order of updating the routing tables from one node to another, but not stating how the topology will be recomputed and how the update's information will be sent by the controller

Our Contribution

The main motivation of this paper is to improve the computer network's QoS by providing solutions to the drawbacks cited above for the approaches of (Pham, 2014; Lim *et al*., 2008). The contribution of this paper is the continuation of our work that previously appeared in Myoupo *et al*. (2018). More precisely, our contribution is a method of updating routing and rerouting tables of nodes by a controller. In contrast to the approach in Myoupo *et al*. (2018) in which only a single link failure is considered, this new approach also considers multilink and node failures; we describe it through three aspects:

- Determination of nodes involved in the update process: We show that by updating certain particular nodes, we can obtain lower packet routing delays in case of a persistent link and node failure than in case of a transient failure
- Construction of an update packet structure: We present a packet structure in the form of a vector of lists, inspired by the Link State Update (LSU) and Link State Advertisement (LSA) packets' structure of the Open Shortest Path First (OSPF) protocol (Moy, 1998). The structure of the update packet must be considered carefully to reduce the size of update packets which can facilitate easier forwarding of the updated routing information over the network
- Definition of a nodes' update scheme: We opt for both simultaneous and sequential updates of the involved nodes. Simultaneous update refers to particular nodes called critical nodes that are not directly linked by links; sequential update refers to the nodes on the original rerouting path used by the IPFRR strategy applied in Pham (2014) to deal with a transient failure. This approach helps reduce the rerouting paths' overloading

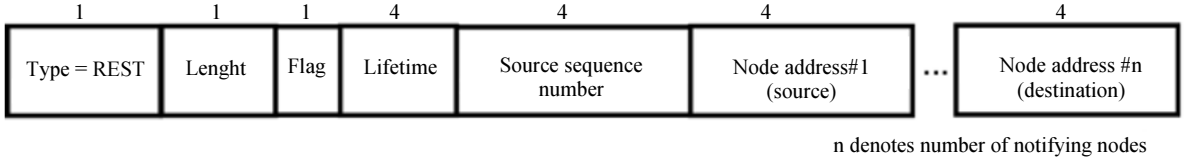

Fig. 1: Sample line graph using colours that contrast well both on screen and on a black-and-white hardcopy

The remainder of this paper is organized as follows. In section 2, we propose several hypotheses related to this study. Section 3 presents our update strategy for a single persistent link failure. Section 4 presents our update strategy for a persistent and non-simultaneous multilink failure. Section 5 presents our approach to the case of a virtual node failure. Our solution for the case of a physical node failure is presented in section 6. Section 7 describes numerical results of simulations. Finally, section 8 concludes the paper.

Hypotheses

In this work, we assume that the network is represented by a graph with bidirectional links. Each edge is associated with a weight that represents the bandwidth or the flow on that edge. The weight of nodes is not considered and it is assumed that for any pair of nodes, there are at least two disjoint paths, making it possible to connect them.

We assume that there is already a rerouting scheme used for handle link and node failures. This scheme is based on Pham (2014), which exploits a shortest path tree called the nominal routing tree (Fig. 2a). It assumes that in case of a link failure, only one node detects the failure (node S in Fig. 2b) and reroutes packets according to the rules described in its routing table, as shown in Fig. 2b (black path). The node that detects failure is located in the red nominal routing tree that hosts the failure and the destination node (node D in Fig. 2b) of the original nominal routing tree is located in the blue nominal routing tree.

In case of node failure, all the links connected to that node fail. Then, multiple nodes detect the failure and each reroutes packets according to the rules provided for a single link failure. We consider a link to fail when flows can no longer use that link. A switch detects a link failure as unavailability of the port connected to that link.

Our Loop-Free Routing Tables Update Method using a Vector of Lists of Triplets in Case of a Single Link Failure

Here, we recall the results of Myoupo *et al*. (2018) that will be useful in the following.

Let us consider a network represented by graph G (N, L), where N represents the set of nodes and L is the set of links. Let us consider the following definitions before they are used later:

- G_r: The red part of the network that hosts a failure
- G_b: The blue part of the network that hosts the destination node of a nominal routing tree
- N_f: The node that detects a link failure when it occurs
- *border node*: A node of the red part connected directly to a node of the blue part
- *critical node*: A border node nearest to the node that detects a transient failure

Our Critical Nodes' Detection Mechanism

We consider a persistent link failure (p, q) (Fig. 3). Let $|N|$ = Cardinal (N) be the number of nodes and $|L|$ = Cardinal (L) be the number of links. We also assume that node D is the destination of the flows of G (N, L). The link failure (p, q) generates the subgraph Gr, which is a routing tree with node p as the root (Fig. 3). For each node of G_r, there is a path linking it to p because of the tree topology.

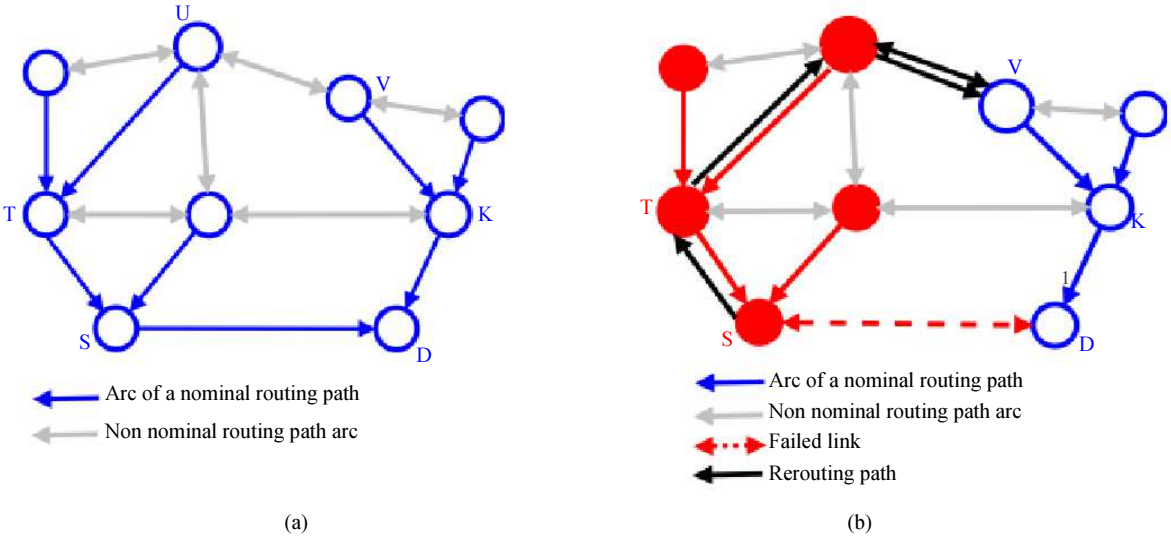

Fig. 2: Rerouting scheme used for link failure. D indicates the destination (a) Nominal routing tree (b) Rerouting path

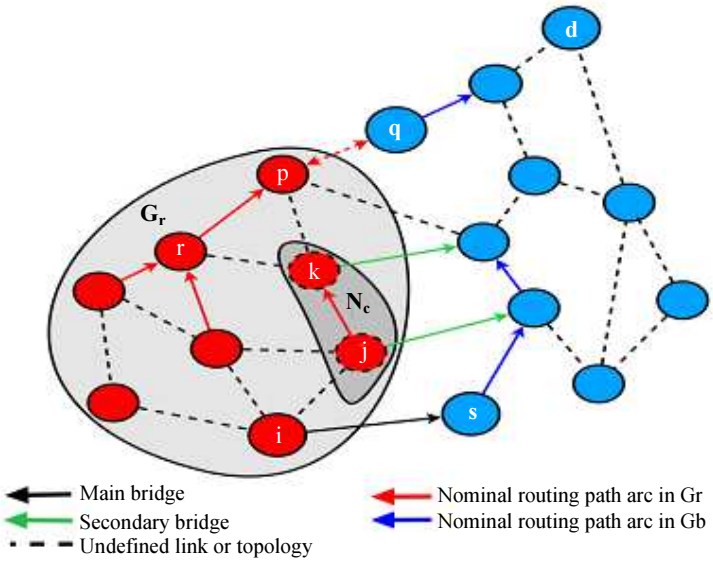

Fig. 3: Our mechanism for detecting the nodes to be updated

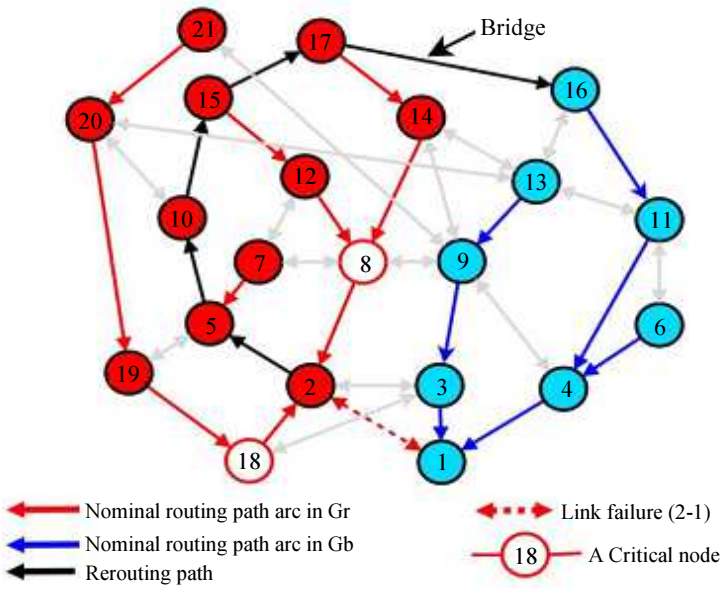

Fig. 4: Our mechanism of detecting the nodes to be updated

The dotted links indicate the possible presence of a topology. According to Pham (2014), there is at least one subgraph rooted in node r, which contains a set of nodes r_i and connects an extremity i of G_r to node s of G_b; this is the bridge (i, s). Similarly, according to the configurations, there are subtrees N_c of G rooted in nodes k_i (i = 1,2,…) and composed of nodes $k_j \neq i$ that directly connect G_r to G_b.

Our detection strategy uses only one bridge (instead of several) from each branch of the nominal tree G_r to retain the network structure. The search process starts from node $p = N_f$. We look for the first node of each routing tree's branch that offers a bridge to reach G_b. That node k is a critical node. To reduce the loop risk in the routing, only these critical nodes among the border nodes will be updated. The exploration of a tree branch stops when a critical node has been found. For instance, in the case of Fig. 4, the critical nodes are 8 and 18.

General Update Principle from one Node to Another

While the critical nodes are located in different tree branches, they are all independent from each other; then,

the controller can update those critical nodes simultaneously through packet-out messages, without the loop risk. In addition to these critical nodes, the nodes of the rerouting path used to handle the preceding transient failure also need to be updated because of rerouting. As to the nodes of the rerouting path, the update will be performed gradually, starting from the extremity node *i* of the bridge located in G_r towards node p that detected the link failure.

As the routing tables of the nodes in the rerouting path are used for transferring the rerouted flows, the update for these nodes will consist of applying these rerouting configurations in their routing tables as the new routing rules. The critical nodes' routing tables will be updated to allow for these critical nodes to use a secondary bridge; the rerouting paths of these nodes must also be rebuilt by considering these new routing tables' configurations. Consequently, the rerouting strategies previously developed by the controller will be recomputed by this method and reinstated in the various nodes of Gr. Otherwise, the routing tree would neither remain nominal nor be optimal, which would increase the packet routing delays.

Internal Updates of Nodes in the Rerouting Path

The internal updates of nodes are performed using a message containing a vector of lists of triplets that contains the updated data provided by the controller. This message's structure is similar to that of LSU messages of the OSPF protocol (Fig. 5). Each entry in the update vector is a list of triplets (*a*, *b*, *c*), where *a* represents the entry gate of the flow to be modified, *b* is the new exit gate of this flow and *c* is the exit gate that will be used for forwarding the remainder of the message to the next node in the vector. In case of a switch, *c* will be a MAC address and in the case of a router, it will be the next hop's IP address. For the nodes that need to update many entries in their routing table, only the last triplets in their update lists will have the value $c \neq 0$. While the update message can easily expand with the number of nodes to be updated, considering the MTU value, the update packet of the nodes will be fragmented into smaller packets by the controller before being transmitted. Each fragment will be assigned to its appropriate group of nodes, as shown in Fig. 6. Each group of nodes of the rerouting path that receive an update message use it only when it receives a permission message from the controller or from the preceding node in the rerouting path; to achieve this latency, the SDN switches or routers are made of agents, as it is the case in Pham (2014) with filters. Flow management, analysis of the ports' status and initiating the appropriate actions as specified by the controller are some features of these agents. When a node wants to use the update message, it reads the field *k*'s value, recovers its list of triplets, performs its updates, increments the value of *k* and returns the update message to the next node in the vector. The last triplet list has the value of *c* equal to 0; this signifies the end of update message transmission between the nodes of the rerouting path. When each node in a group receives its update triplets list, it checks if $c = 0$ in the last triplet and in such a case, updates its routing table and destroys the update message; otherwise, it forwards this update message to the next node.

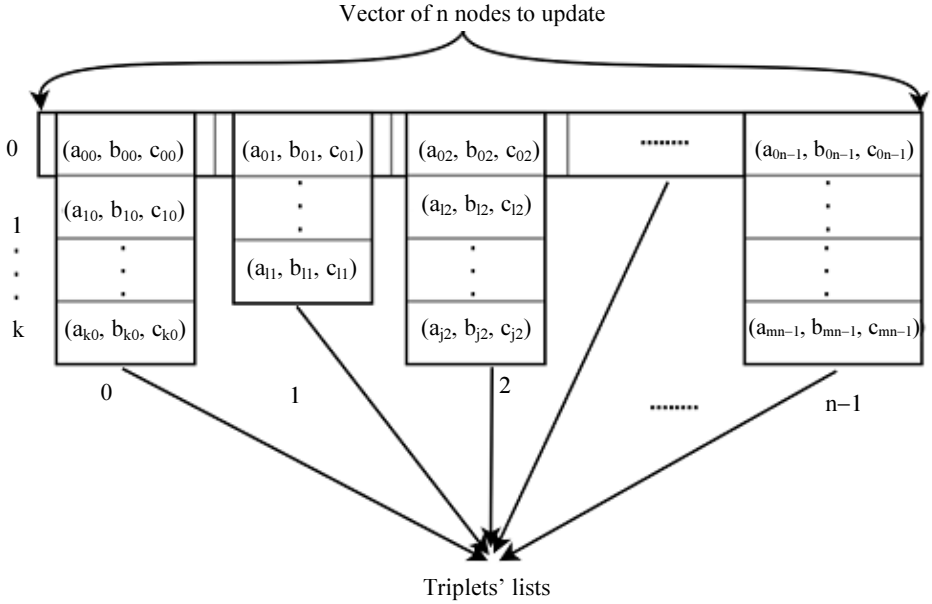

Fig. 5: Update structure of a vector of lists of triplets

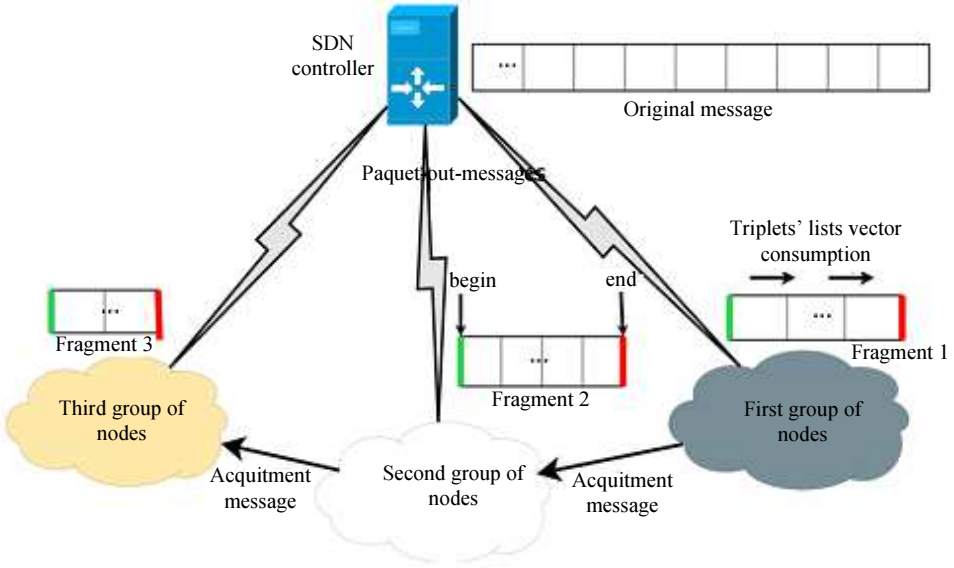

Fig. 6: Fragmentation of the update message

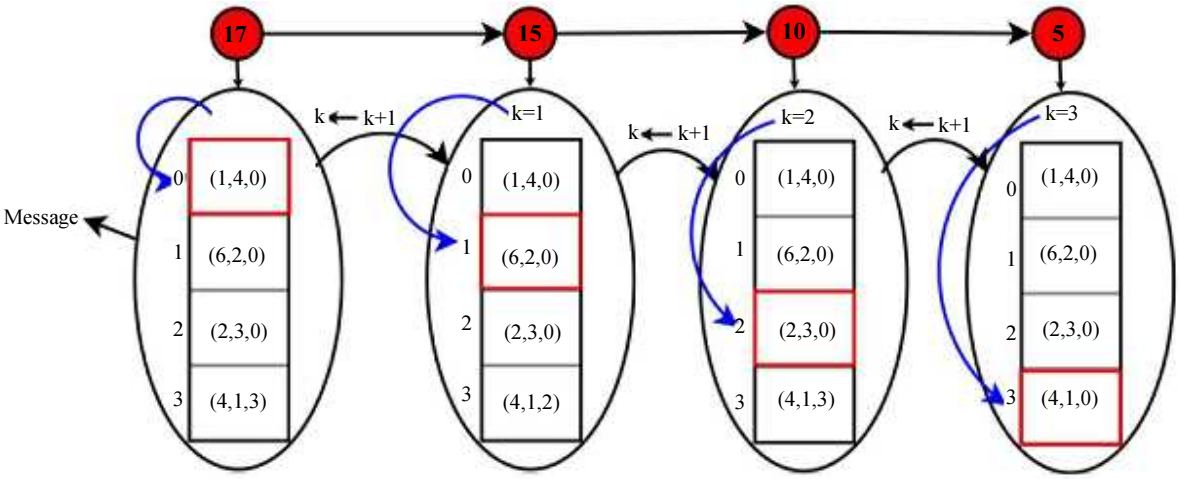

Fig. 7: Internal updates of nodes using a vector of lists of triplets

The use of this message structure is illustrated for the successive updates of a group of nodes 17, 15, 10 and 5 in Fig. 7. In that figure, node 5 is the last one to be updated in the group.

Reconstruction of Rerouting Tables' Rules using the New Routing Configurations

While rerouting rules are computed based on the new routing table's configurations, rerouting rules of each updated node must also be updated. Using the approach of Pham (2014) on a virtual nominal routing tree built by the controller, the controller computes the new rerouting rules; during this step, the network uses the prior rerouting rules as in the Cisco routers, resulting in poor QoS. To minimize the latencies during computations, the operations for determining the critical nodes and computations of the new routing and rerouting rules must all be done in advance in the controller before initiating any tables' update process. For the same reasons, the updates to the routing tables and rerouting tables in each node can be performed at the same time assuming that the controller has already provided the rules.

Our Routing Tables Update Approach for non-Simultaneous and Persistent

Multilink Failures Multiple link failures are called non-simultaneous when they occur sequentially. Each of these failures can persist over time (for instance, more than 10 min in Pham (2014)); then, they are deemed persistent failures. Persistence of each failure will be detected by the controller consecutively. In this section, we propose several

resolution approaches that could be considered for updating the routing tables and explain our strategy.

First Solution: Consecutive Update of Nodes as Link Failures are Deemed Persistent

This solution suggests solving persistent link failure cases as they are detected by the controller. Each detected case is solved by applying the proposed approach to the persistent single link failure case. This solution is inspired by Pham (2014), which is related to the rerouting rules' computation for cases of simultaneous and transient multilink failures.

Figure 8 illustrates this solution for persistent and nonsimultaneous failures of two links. In this figure, let us assume that the link failure (3, 1) is deemed persistent before (8, 2) is by the controller.

Then, the controller immediately recalculates the rerouting path to use the path 3-9-4 to reroute the flow and launches the update of nodes 3 and 9; this update introduces the new configuration presented in Fig. 8c. The controller starts the rerouting path computation (path 8-14-13) directly for link failure (8, 2) by considering this new configuration and sends the update message once the computation is complete.

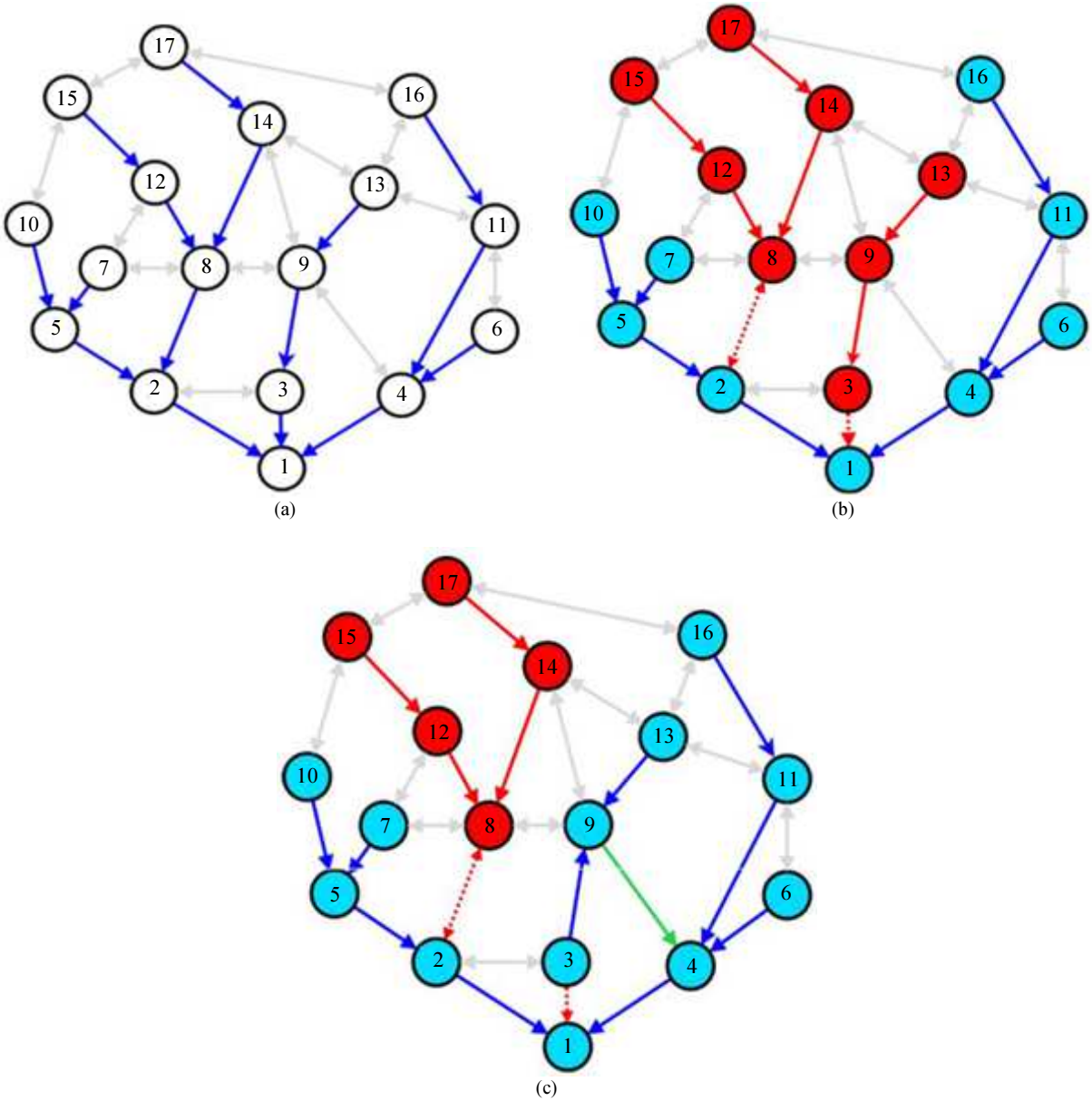

Fig. 8: Updates' structure of a vector of lists of triplets (a) Original nominal routing tree with node 1 as the destination (b) Persistent non-simultaneous multilink failure (c) New configuration after persistent failure management

Second Solution: Synchronized Update of a Node's Routing Tables as Persistent Link Failures are Gradually Detected

Consider a network subject to a set of nonsimultaneous link failures. The main drawback of the first solution is the difficulty of managing multiple update messages for different link failures when they arrive simultaneously at the same node. To solve this question, we prove the following theorem:

Theorem 1

For any pair of non-simultaneous link failures, there exists at least one common node in subgraphs *Gr* and *Gb* corresponding to these two failures.

Proof

Let VN be a virtual network represented by a graph G ($|N|$, $|L|$) where $|N|$ is the number of nodes and $|L|$ the number of links. Let d be the destination of the flows of a Nominal routing tree of G. Let (l_1, l_2) be a pair of link failures. Let n_1 be the node that has detected l_1 and n_2 the one that has detected l_2. l_1 subdivides G into two subgraphs G'_1 and G'_2, with $n_1 \in G'_1$ and $d \in G'_2$ such that $\cup_{i\in(1,2)} G'_1 = G$. Similarly, l_2 divides G into G''_1 and G''_2 such that $n_2 \in G''_1$ and $d \in G''_2$.

Let us consider $G'_1 \cap G''_1 = \phi$ and $G'_2 \cap G''_2 = \phi$. We have $d \in G'_2$ and $d \in G''_2 \Rightarrow d \in G'_2 \cap G''_2$; but $G'_2 \cap G''_2 = \phi$. So $d \in \phi$, which is absurd. Hence $G'_2 \cap G''_2 \neq \phi$. $G'_1 \cap G''_1 = \phi \Rightarrow n_1 \notin G''_1$ and $n_2 \notin G'_1$. But, the failure l_1 creates the subdivision $S_{1|G}$ with $G'_1 \subset S_{1|G}$ and $G'_2 \subset S_{1|G}$. Similarly, l_2 creates the subdivision $S_{2|G}$ such that $G''_1 \subset S_{2|G}$ and $G''_2 \subset S_{2|G}$. Since each failure induces a subdivision of the graph G into two subgraphs, we have: $S_{1|G} \subset S_{2|G}$ or $S_{2|G} \subset S_{1|G}$. Thus, $n_1 \in S_{2|G} \Rightarrow n_1 \in S_{2|G} \Rightarrow n_1 \in G'_1 \cap G''_1$ which is absurd because $G'_1 \cap G''_1 = \phi$.

Theorem 1 states the existence of common nodes in different subtrees generated by n non-simultaneous and consecutively persistent link failures. These common nodes are the most likely nodes to cause conflicts in the routing tables update' process between multiple messages. This theorem allows us to consider two approaches of solutions to this synchronization problem:

Approach 1

Wait for a packet-in message from the network before sending any update message for another link failure. When the updates' data for a given failure is being applied in the network, the controller waits to receive a packet-in message from the last node group to be updated, before sending the update packets for another failure. Considering Fig. 6 related to the updates' message fragmentation, the last group of packets is that of group 3, i.e., the packet-in message sent to the controller at the end of the update process for the handled failure will come from this group. In this group, it is the last node to be updated that will send this message, e.g., node 5 of Fig. 7 in section 3. This node will know that it is the last one by examining the value of c that is zero in the last update triplet. This approach has the advantage of avoiding the cases of conflicts and loss of update data for different failures; however, it has the major drawback of increasing latency.

Approach 2

Assign a label referencing the priority level of update messages for each failure.

The controller adds into the update message for a given failure a priority order in the set of update messages related to other persistent detected link failures. This priority order is a field that is a date or an integer, the value of which is incremented each time a new persistent failure is detected. Hence, there is no need to know in advance the existence of any potentially persistent failures in the network to use this field. To avoid information loss, the controller must keep up to date internally the current value of this field. This approach helps avoid the assignment of the same label to two update messages concerning different failures.

When multiple update messages are encountered within the same node, the value of the priority field is used to respect the precedence order; this is done especially when it is necessary to simultaneously apply many conflicting update data to a common entry of the relevant node's routing table. Then, the message with the smallest priority value is used before other messages with larger values. Non-conflicting update data related to the same entries are applied simultaneously. This approach reduces latency by allowing multiple update messages of different failures to be forwarded simultaneously; the preceding solution does not allow this.

Approach 2 is what we propose for a node's update and synchronization problems in cases of non-simultaneous and persistent multiple link failures. This approach enhances that used for the case of a single persistent link failure proposed in section 3, with several adjustments:

- A field is added to the update message structure to manage update synchronization in the common network parts delimited by several failures
- We assume at least two persistent link failures in the network

Our approach provides the following advantages:

- Preserving the consistency of paths in the network
- Maximizing handling of non-simultaneous link failures through the update synchronization method
- Allowing us to treat the cases of simultaneous failures of two non-adjacent links to the same node

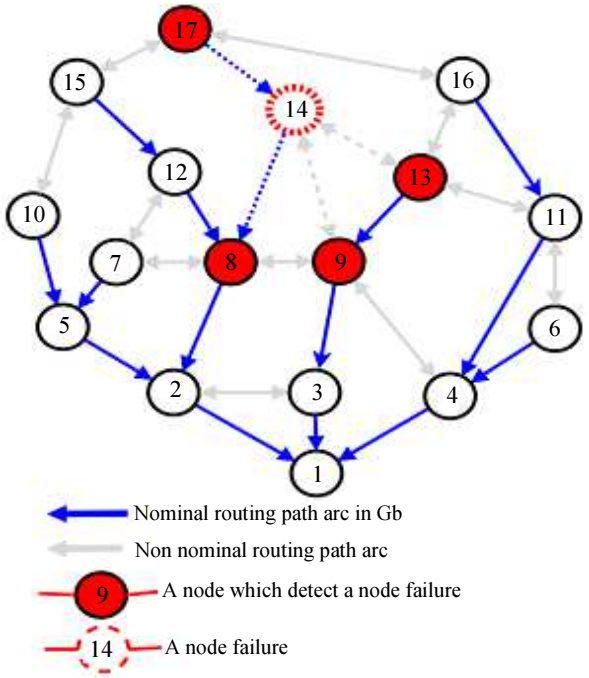

Fig. 9: Failure of node 14

The Persistent Virtual Node Failure Case

A node (router or switch) failure, whether physical or virtual, implies the impossibility of transferring data from any incoming port to an outgoing port of that node. This failure scenario directly leads to a simultaneous failure of several links. Figure 9 shows the failure of node 14 that induces failures of links (14, 17), (14, 13), (14, 9) and (14, 8).

We consider for the moment a single node failure in a virtual plane. Then, a node failure can be considered as:

- Simultaneous multiple link failures adjacent to the same node, as shown in Fig. 9
- Internal damage of the node

In both situations, it is impossible to transit flows from one input port of the equipment to another. The node failure's persistence is detected based on that of failures of links adjacent to it. For various reasons, the persistence of these failures can be detected and addressed by the controller in several ways:

Solution 1: Successive Detection of Persistent Adjacent Links' Failures

Persistence detection of multiple failed links is performed successively. In this case, the new rerouting paths are computed using the method of Pham (2014). The update mechanism used is approach 2 presented for the case of non-simultaneous multilink failures.

Solution 2: Simultaneous Detection of Persistent Link Failures Adjacent to the Same Node

In this case, the link failures that caused the node failure are all deemed persistent at the same time. According to the IPFRR strategy (Pham, 2014; Papan *et al*., 2017), the nodes that will detect the failure of node 14 are 17, 8, 9 and 13 (Fig. 9). These detector nodes will be the first to be updated to avoid the cycles in the rerouting; the next nodes are those of the rerouting path found by using the strategy of Pham (2014). The updates of the detector nodes will be primarily to divert flows for failed links to available links. For the case of Fig. 9 where node 1 is the routing tree destination, paths 17-1611-4-1, 8-2-1, 9-3-1 and 13-9-3-1 can be used to treat the link failures (14,17), (14,8), (14,9) and (14,13), respectively. The general update principle for this second solution is as follows:

1. The controller uses its global network view to detect the failed node from the list of failed adjacent links
2. For each persistent link failure detected, the controller builds the updates' messages based on our strategy proposed in section 2 to handle a single persistent link failure. A date field is included in each update message to manage synchronizations problems
3. THE controller sends the updates' messages to the nodes that detected the given node failure

Our Strategy for a Persistent Physical Node Failure

The network virtualization paradigm allows the sharing of a physical infrastructure via the creation of virtual networks that are hosted by this infrastructure (Mosharaf Kabir Chowdhury and Boutaba, 2010). Then, a physical node damage involves the failure of multiple virtual nodes belonging to the same plane or to different planes according to the mapping model that exists between the substrate network and the virtual networks being hosted (Nashid *et al*., 2016). Figure 10 shows a network virtualization environment including two virtual networks VN1 and VN2.

Nodes 3 and 4 of VN1 and node 5 of VN2 are all hosted within node 2 of the substrate network. When substrate node 2 fails, it directly induces the failure of nodes 3 and 4 of VN1 and node 5 of VN2; the physical and virtual links also fail at the same time.

In this type of configuration, the solutions proposed in Nashid *et al*. (2016) or Pham (2014) can be used for rerouting. In addition, various approaches to multilayer rerouting proposed in the framework of multilayer optical networks (Doumith, 2007) allow us to consider several routing tables update approaches:

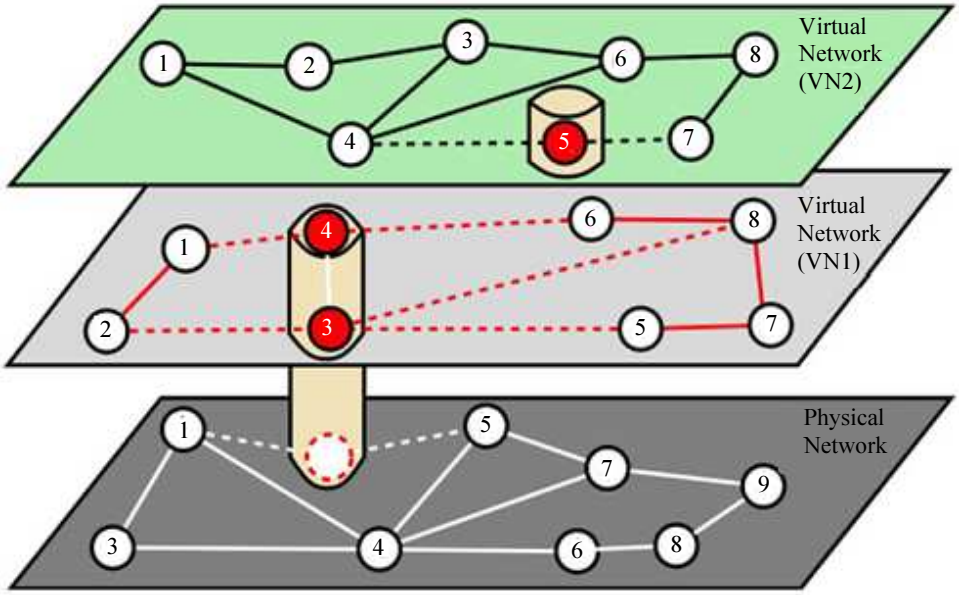

Fig. 10: Physical node failure

Solution 1: Uncoordinated Update

This solution uses our previously proposed strategy for virtual node failure in each virtual network. As a result, the controller triggers the update for each virtual plane affected by the failure of the physical node.

This approach suffers from an excessive protection bandwidth reservation that could decrease the network performance, especially in case of residual bandwidth deficiency in the network.

Solution 2: Coordinated Update

The aim is to define a sequential update process for the set of virtual networks affected by the substrate node failure. However, we first update the necessary nodes in the physical network (location of the physical node failure) before examining the hosted virtual planes: This strategy ensures the integrity of the substrate network, which affects the QoS in the network through its shared resources (bandwidth, throughput, etc). In the physical network, we can use our routing tables update scheme proposed for a virtual node failure.

Considering the dimensioning network problem and the virtual network mapping on the physical network, a physical node failure can lead to a simultaneous failure of multiple virtual nodes; e.g., this is the case for virtual nodes 3 and 4 of virtual network VN1 in Fig. 10. This failure's heterogeneity in the virtual networks makes the coordinated update process very difficult. Then, some update priority criteria for the affected virtual planes are needed to efficiently manage conflicts in the presence of multiple simultaneous persistent failures. These criteria are defined by the infrastructure provider according to the intended objectives. In this paper, we define two priority criteria: The number of failed virtual nodes and the traffic weight. Hence, depending on the infrastructure provider's objectives, we can propose the following solutions for managing update for a set of affected virtual planes:

- Prioritize the virtual nodes of the network with the fewest failed virtual nodes. This approach has the advantage of minimizing the routing conflicts. In the case of Fig. 10, the nodes of virtual network VN2 present only one node failure, while VN1 is affected by two node failures. Then, the nodes of virtual network VN2 will be updated before those of VN1
- Focus on the virtual network with the highest traffic weight: The advantage of this approach is the customers' loyalty through a traffic-oriented QoS

Implementation and Simulation Results

The OMNet++ network simulator environment has been used to implement and evaluate our routing table's update scheme. This network simulator provides an extensive graphical user interface (GUI), intelligent support, short computation times and an effective implementation of large-scale networks (Bilalb and Othmana, 2012). Moreover, the flexible NED language offered by OMNet++ allows full network topology customization even when the simulation is running, which meets the needs of our study. In the simulations, we consider the different networks in Table 1. These networks satisfy our hypotheses and have been randomly generated to

better assess the ability of our update strategies to apply to any network that satisfies our hypotheses.

The simulations have been performed on a computer with the following configuration: Core i5 2.40 GHz CPU, 4.00 GB of RAM and 12 MB of cache.

The objective of our simulations is to compare QoS of the network in the absence of persistent link failures and in the presence of such failures to show the efficiency of our routing tables update strategy for the QoS improvement in the presence of failures. To this end, we performed simulations on high data rate networks (1.2 Mbits/sec to 512 Mbits/sec) of various scales and focused on packet routing delays and the data loss rate. We also compare our results to several existing studies. For both link and node failures, the data were obtained by performing several tests on the networks in the following order:

- In the absence of failures
- In the presence of a transient link failure
- In the scenario where the above link or node failure was deemed persistent

The analysis of packets' routing delays in various nodes and various network instability scenarios allowed us to appreciate the QoS variability. Consequently, in the case of a single persistent link failure, the average of packet routing delays in network 3 is presented in Fig. 11. We note that the average packet routing time in the presence of a persistent link failure is lower than those observed in the presence of a transient failure; this occurs because some packets are forwarded through the critical nodes. In addition, there is a high gap between the TBR (Tree-Based Routing) approach and the packet routing delay in the network without a failure. This significant difference could be explained by the TBR approach relying on the data of neighbours to construct its updating information; since data of certain neighbours might not reach the destination node, the update data can be wrong. Our approach actually uses all the information from neighbouring nodes to build its update data, allowing optimal paths that reduce the packet transit time.

Figure 11 reveals the results for a large-scale network (60 nodes); however, this result is similar to those for other tested networks (network1 and network 2) that are small. Network 3 is interesting to us because its size is more compatible with the SDN scale.

Considering the node failure problem, Fig. 12 presents the packet transit delays in the network. We note that the previous observations made for a persistent single link failure also apply here. The packet transit delays with our routing tables update method are lower than those in the literature. Then, in case of a persistent node or multilink failure, our method improves the packet transit delays.

Table 1: Simulation networks

Network	Number of nodes	Number of links
Network1	10	18
Network2	20	31
Network3	60	90

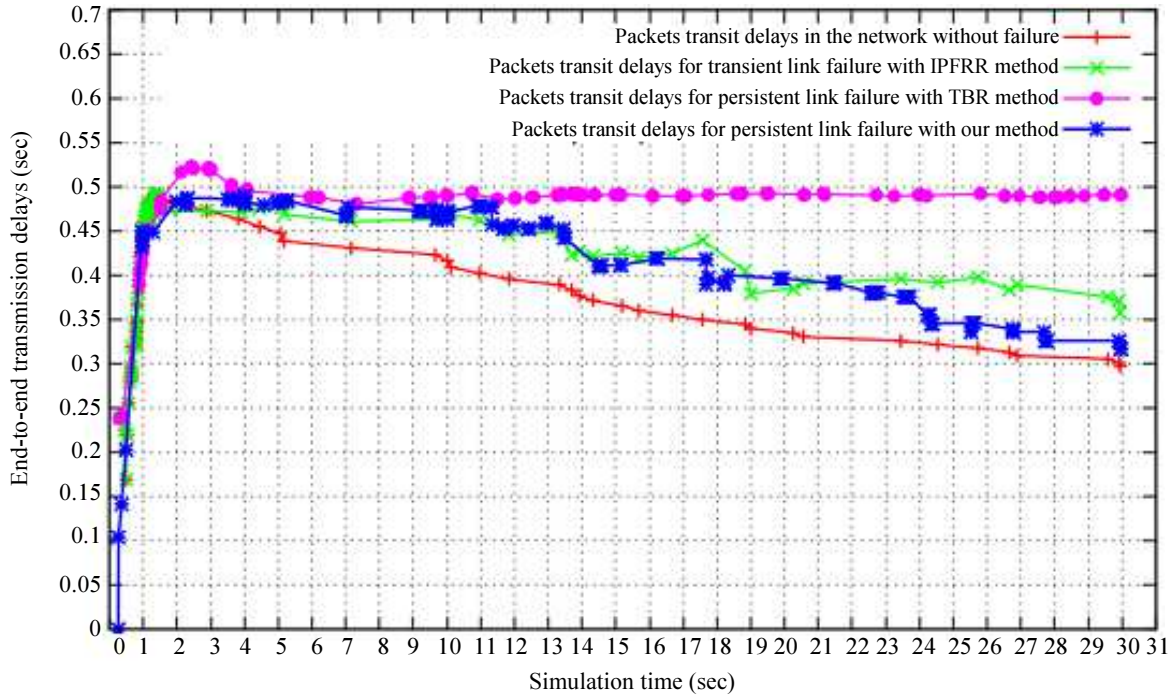

Fig. 11: Packets routing delays' variations for a persistent single link failure in network3

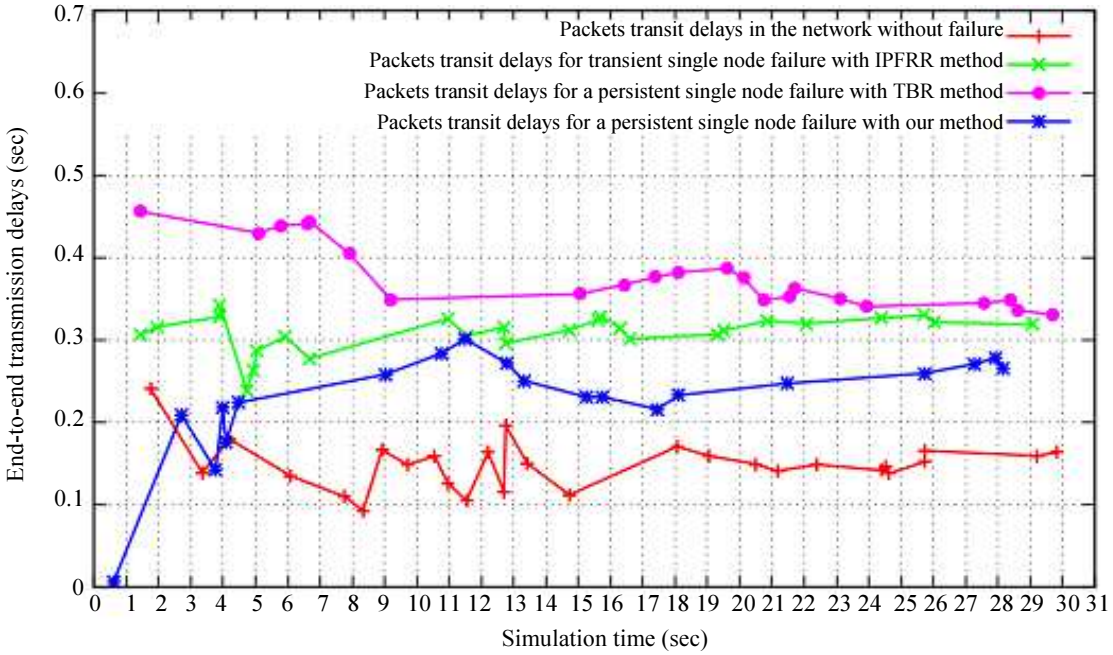

Fig. 12: Packets routing delays' variations for a persistent single node failure in network3

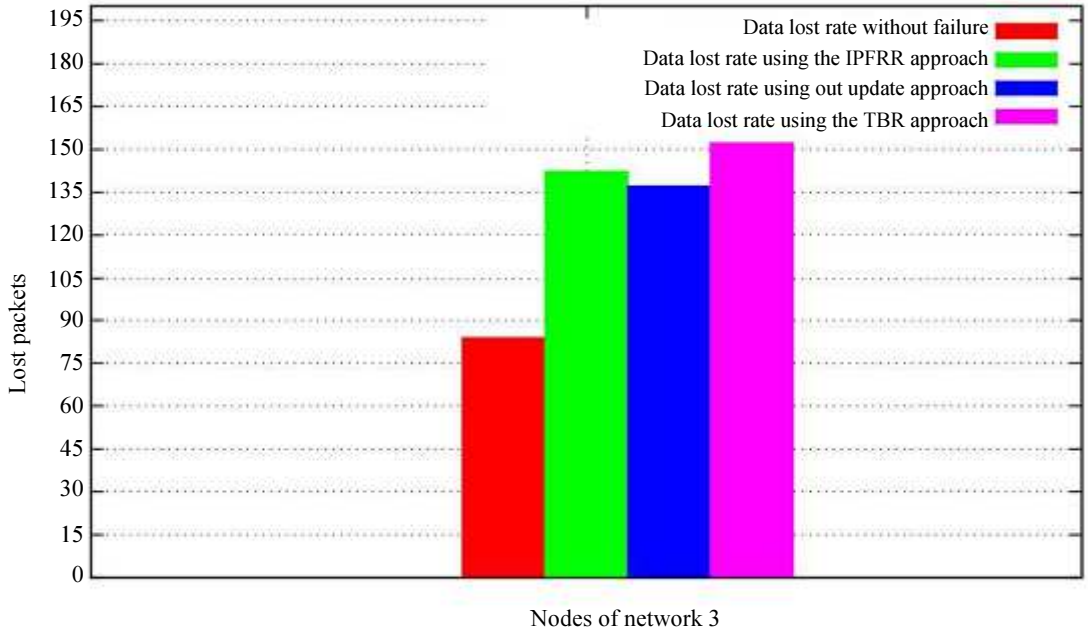

Fig. 13: Comparison of the data loss rate of our update strategy

Considering the data loss rate, we obtain the results displayed in Fig. 13 for a single link failure and Fig. 14 for a single node and multilink failures. Both figures show that a link or node failure highly increase the data loss rate in the network. This data loss rate is more important for the case of a persistent single node failure than the persistent single link failure. We note a strong increase by approximately 60% of this data loss rate (Fig. 14) compared to those observed in the absence of failures. This phenomenon could be explained by the lack of resources needed to reroute the traffic. Nevertheless, our approach helps decrease this data loss rate to 24% (compared to the IPFRR method) and 43% (compared to the TBR method). This improvement means that our approach is better than TBR and IPFRR in terms of the data loss rate.

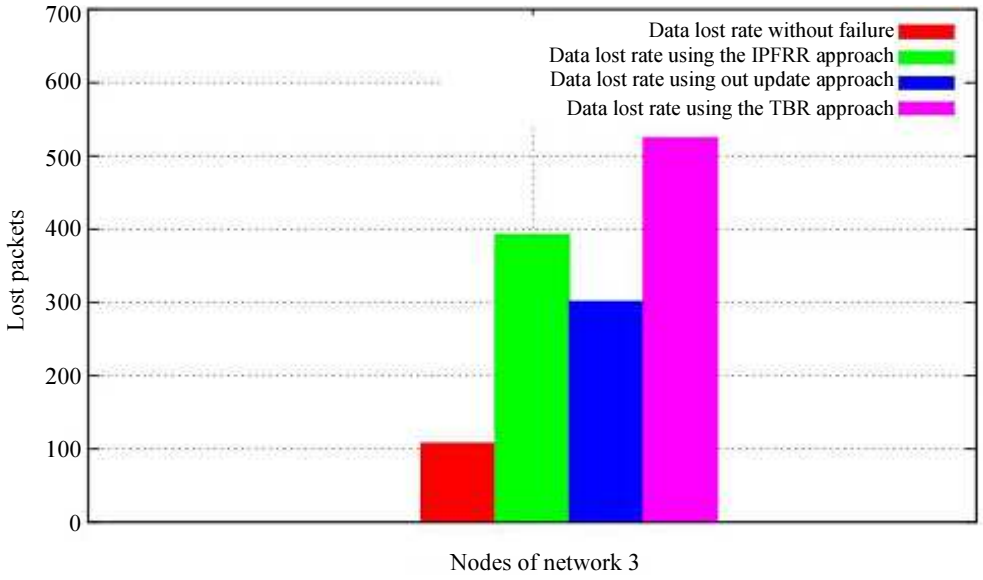

Fig. 14: Comparison of the data loss rate of our update strategy to those of Pham (2014) and TBR for network3 in case of node failure

Conclusion

In this paper, our aim was to propose a routing and rerouting tables' update mechanism to overcome nodes' reconfiguration challenges in case of persistent single link, multilink and node failures in a virtual network. Hence, we proposed an update strategy that efficiently identified the nodes to be updated and defined an update scheme for those nodes. This scheme, originally built on a vector of lists of triplets inspired by OSPF's LSU packet structure has been adapted to treating the cases of persistent multilink and node failures. The simulations we performed show that our routing tables update method helps reduce packet routing delays and the data loss rate, which are two important metrics in computer networks. Consequently, this study provides solutions to the routing tables' update challenge in the SDN architectures, to offer a good QoS at all times to the endusers and avoid losing customers to infrastructure providers (InP). In addition, the ideas presented in this paper and related to the management of multiple virtual planes in case of a node failure can help the InP better allocate resources for network recovery.

To improve this work, further research could specifically address the flow congestion problem in the network when there are numerous active users' requests. Even though our update approach is effective, update messages can be stopped by certain network congestion cases. In these cases, our method will not be very useful.

Acknowledgement

We thank the anonymous reviewers whose valuable comments and suggestions have significantly improved the presentation and the readability of this work.

Author's Contributions

Yannick Florian Yankam: The author contributed to the review the various published articles in the field, contributed to the hypothesis the date analysis, writing of the manuscript and final approval.

Jean Frédéric Myoupo: The author designed the research proposal, organized the study, designed of the proposed work plan, contributed to the hypothesis, writing of the final manuscript and final approval.

Vianney Kengne Tchendji: The author contributed as the research guide, technical corrections, reviewing it critically and final approval.

Ethics

This work is original and not published elsewhere. The authors confirm that they have read and approved the manuscript and there is no conflict of interest. Also, the authors confirm that there are no ethical issues involved.

References

Azodolmolky, S., P. Wieder and R. Yahyapour, 2013. SDN-based cloud computing networking. Proceedings of the 15th International Conference on Transparent Optical Networks, Jun. 23-27, IEEE Xplore Press, Cartagena, Spain, pp: 2181-2206. DOI: 10.1109/icton.2013.6602678

Bilalb, S.M. and M. Othmana, 2012. A performance comparison of open source network simulators for wireless networks. Proceedings of the IEEE International Conference on Control System, Computing and Engineering, Nov. 23-25, IEEE Xplore Press, Penang, Malaysia, pp: 34-38. DOI: 10.1109/iccsce.2012.6487111

Doumith, E., 2007. Agrégation et routage de trafic dans les réseaux WDM multicouches. Ph.D Thesis, École Nationale Supérieure des Télécommuni-cations (Télécom ParisTech), France.

Farhady, H., H.Y. Lee and A. Nakao, 2015. Software defined networking: A survey. Comput. Netw., 81: 79-95. DOI: 10.1016/j.comnet.2015.02.014

Hedrick, C.L., 1988. Routing information protocol. RFC Editor.

Hu, F., Q. Hao and K. Bao, 2014. A survey on software-defined network and OpenFlow: From concept to implementation. IEEE Commun. Surveys Tutorials, 16: 2181-2206. DOI: 10.1109/comst.2014.2326417

Lim, A.O., X. Wang, Y. Kado and B. Zhang, 2008. A hybrid centralized routing protocol for 802.11s WMNs. Mobile Netw. Applic., 13:117-131. DOI: 10.1007/s11036-0080038-4

Mosharaf Kabir Chowdhury, N.M. and R. Boutaba, 2010. A survey of network virtualization. Comput. Netw., 54: 862-876. DOI: 10.1016/j.comnet.2009.10.017

Moy, J.T., 1998. OSPF: Anatomy of an Internet Routing Protocol. 1st Edn., Addison Wesley Professional, ISBN-10: 0201634724, pp: 339.

Muruganathan, S.D., D.C.F., Ma, R.I. Bhasin and A.O. Fapojuwo, 2005. A centralized energy-efficient routing protocol for wireless sensor networks. IEEE Commun. Magazine, 43: S8-S13. DOI: 10.1109/mcom.2005.1404592

Myoupo, J.F., Y.F. Yankam and V.K. Tchendji, 2018. A centralized and conflict-free routing table update method through triplets' lists vector in SDN architectures. Proceedings of the IEEE SmartWorld, Ubiquitous Intelligence and Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People and Smart City Innovations, Oct. 8-12, IEEE Xplore Press, Guangzhou, China, pp: 1509-1515. DOI: 10.1109/SmartWorld.2018.00261

Nashid, S., R. Ahmed, A. Khan, S.R. Chowdhury, R. Boutaba and J. Mitra, 2016. ReNoVatE: Recovery from node failure in virtual network embedding. Proceedings of the 12th International Conference on Network and Service Management, IEEE Xplore Press, Montreal, QC, Canada, pp: 19-27. DOI: 10.1109/cnsm.2016.7818396

Papan, T., P. Segec, M. Moravcik, J. Hrabovsky and L. Mikus *et al*., 2017. Existing mechanisms of IP fast reroute. Proceedings of the 15th International Conference on Emerging eLearning Technologies and Applications, Oct. 26-27, IEEE Xplore Press, Stary Smokovec, Slovakia, pp: 1-7. DOI: 10.1109/iceta.2017.8102516.

Pham, T.S., 2014. Autonomous management of quality of service in virtual networks. Ph.D Thesis, University of Technology of Compiègne (UTC), France.

Perkins, C., E. Belding-Royer and S. Das, 2003. Ad hoc on demand distance vector (aodv) routing. RFC Editor.

Rothenberg, C.E., M.R. Nascimento, M.R. Salvador, C.N. Araujo Corrêa and S. Cunha de Lucena *et al*., 2012. Revisiting routing control platforms with the eyes and muscles of software-defined networking. Proceedings of the 1st Workshop on Hot Topics in Software Defined Networks, Aug. 13-13, ACM Press, Helsinki, Finland, pp: 13-18. DOI: 10.1145/2342441.2342445