Towards Filling The Gap of Routing Changes in Software-Defined Networks

Ali Malik (✉), Benjamin Aziz, and Mo Adda

School of Computing, University of Portsmouth, Portsmouth, UK
{ali.al-bdairi, benjamin.aziz, mo.adda}@port.ac.uk

Abstract. The inflexibility and rigidity of the current networking systems in light of the growing number of computing devices that connected to the Internet has led to the need of a new generation of computer networks. In this context, software-defined network has recently emerged as one of the most recognizable solutions to facilitate the network management and decrease the complexity through detaching the control layer from network appliances. Since, each new technology accompanied with new issues and concerns, one of the main challenges that facing the SDNs is the fault tolerance and resilience. Although there are a numerous of studies that have discussed this issue, the after failure repair scenario is remained unclear and have not addressed well yet. For this reason, we specify this paper as a step forward to address the issue of routing changes, which is important for utilising the optimal paths after failure repair. We produced a new network model as well as a compatible framework with SDN architecture to fulfill the routing changes activity. The consequence is estimated based on an analytical model and finally, a case study example is provided in order to conceptualise the concept.

Keywords: Routing changes; traffic engineering; fault tolerance; software defined networks.

1 Introduction

The management of traditional networks is complex and difficult due to the nature of the networking equipment, from switches to middleboxes such as firewalls, intrusion detectors and load balancers, that work based on the distributed protocols and require a long process of configuration across every single device. Software-Defined networking (SDN) is the next generation of networking architecture that aims to solving legacy networking challenges through simplifying network management and facilitating network evolution and innovation. In SDN, the control plane has been decoupled from the data plane, which has resulted in a new programmable network architecture that can be classified into three layers: *Control, Data,* and *Application* [1]. The control layer, which also called the *controller*, represents the network brain that includes the entire logic unit and intelligence and therefore it is responsible for managing the whole network events and activities. The application layer can be considered as a complimentary part of the control layer as it represents the innovative solutions for different

problems such as load balancing, access control, fault tolerance, etc. Sometimes, the control and application layers counted as one layer. The data layer represents a set of interconnected forwarding elements (e.g. switches and routers) that in charge of forwarding the data packets among the nodes. Due to the decoupling operation, the data plane nodes become a dummy and hence it should be dictated by the control plane so that it is imperatively necessary for each node to expose its state periodically to the network controller as well as requesting the possible action regarding the newly arrived packets. The controller in turn pushes the proper action to the relevant nodes, which will be installed as a forwarding rule (i.e. flow entry). Hence, SDN is a centralised networking system in which the controller has a global view on the network topology and this is one of the main advantages of SDNs. OpenFlow (OF) [2] is the first standard protocol that used to govern the communication between the data plane and controller. Despite of the SDN paradigm has brought a bunch of benefits to the networking system, new concerns have risen regarding the new architecture of SDN and one of those challenges is the fault tolerance [3]. So far, SDN pursuing the same mechanism of traditional networks in terms of recovery from failure –in other words, SDN has the ability to mask the failure events either *proactively* (also known as protection) or *reactively* (also known as restoration). There are a couple of pros and cons that associated with each strategy and perhaps the most prominent ones are those related to the overhead of control plane from one side, and the memory cost of the forwarding elements from the other side.

In protection, the *alternative* (i.e backup) paths are preplanned in advance along with the *primary* paths, hence, no controller intervention is required at the moment of failure. However, this mechanism is memory exhaustion since massive number of rules need to be installed. In restoration, the alternatives will be computed when failure occurs, thus, controller intervention is required in order to divert the affected traffic away from the inoperative area. In fact and according to [4], most of the failure management proposed methods were limited to the stage of failure recovery. Therefore, this paper focuses on the scenario of maintaining optimal paths after failed links have been repaired. The rest of the paper is organised as follows. In Section 2 we highlight the main issue with the current fault management techniques and presents the contribution of this paper. In Section 3, we define the network model as well as an analytical model that dedicated to measure the effect of our method. In Section 4, the proposed framework is demonstrated with a new novel extended-routing algorithm. A case study and discussion are presented in Section 5. Finally, the conclusion and future work are provided in Section 6.

2 Problem statement

The Traffic Engineering (TE) and routing strategies are concerned with keep the network operating efficiently by optimising the network performance through the dynamically analysing and regulating the transferred data over the network. Currently, one of the most SDN-TE challenges is the perspective of fault tolerance

(as stated earlier). Changing paths according to failure incidents is not enough as this will leave the network in a not fully-optimised state. Since the alternative path, which will be adopted to override a failure scenario, is usually treated as a sub-optimal solution[1] comparing with the influenced primary one, therefore; it should resides as an interim solution till the affected primary path get back to the operational state again. Although, some of the legacy networking protocols (e.g. the Interior Gateway Protocol IGP) have considered this issue through appending two routing changes: one when failure occurs and the other one when failure get reformed [5], none of the literature of reactive SDN fault tolerance has addressed the network situation when link failures get repaired, however, some of the SDN-TE solutions such as the Adaptive Multipath Computation Framework (ADMPCF) [6] that assisting to quickly overcome the failure incidents through striving to find a multiple good paths that can be employed in a different context (i.e. failures, load balance and resource utilisation), along with some other contributions in [7] that targeted the same problem. Unfortunately, and unlike the traditional network, all the previous works have not explicitly discussed the repair events and its instantaneous impact on improving the network performance and therefore we put a step closer to complete the picture of reactive fault tolerance as it is not an issue for the proactive technique, since the primary and backup are both installed and the switching between them is based on their availability. In spite of the fundamental requirements of the failure recovery mechanisms of SDNs, the action of when failure get repair is no less important than the action of when failure is detected as it affects the Quality of Service (QoS) by assigning a sub-optimal path instead of the optimal one. To this end, the main contribution of this paper can be summarised in: (1) define a new network model to tackle the issue of when the failed link is repaired, (2) design a new framework that incorporates a novel algorithm to show how the issue can be remedied.

3 Modelling

3.1 Network model

The network model describes the major concepts that need to be integrated with the reactive fault tolerance mechanisms of SDNs. We firstly model the network topology as an *undirected graph* $G = (V, E)$, where V represents the set of vertices (i.e. routers) that ranges over by $v_i, v_j, \ldots, v_z$ where $i, j, \ldots, z \in \{1, \ldots, n\}$ for $n \in \mathbb{N}$, and E represents the set of bidirectional edges (i.e. links) that denoted as $\{e_{ij}\}$ where each $e_{ij} \in E$ is an edge that enables v_i and v_j to connect each other. Now, we define the following test operational function (OP) over a link, which reflects the link's state whether it is working or not:

[1] Routing algorithms aim to designate the optimal route, also known as the primary path, to the traffic on demand.

$$OP(e_{ij}) = \begin{cases} 1 & \text{the link is operational} \\ 0 & \text{otherwise} \end{cases}$$

We then define the failed link set (F) as following:

$$F = \{e_{ij} \mid e_{ij} \in E \wedge OP(e_{ij}) = 0\}$$

Based on G, we define a path P as a *sequence* of consecutive vertices that represent the actual routers in the underlying network. Each path starts at a source router, s, and ends with a destination router, d:

$$P = (s, \ldots, d)$$

We then define the set $Flow$ of all demand traffic flows to be carried via the network topology G. Each $flow \in Flow$ is an instance of a path P, which is associated with a particular traffic that defined by a unique s and d pairs. We utilised the well known Dijkstra [8] as a function $\mathcal{D}$ to form the shortest path of each $flow$, which is typically capture the path within the minimal number of hops. We consider $flow_{set}$ as the set of all possible paths between s and d, which can be defined as follows:

$$flow_{set} = \{P \mid (\mathit{first}(P) = s) \wedge (\mathit{last}(P) = d)\}$$

and the definition of *first* and *last* is given as functions on any general sequence $(a_1, \ldots, a_n)$:

$$\mathit{first}((a_1, \ldots, a_n)) = a_1$$
$$\mathit{last}((a_1, \ldots, a_n)) = a_n$$

We also consider P_{set} as a set that contains all the admissible paths that can be constructed from G, so, this means that $P \in P_{set}$ and therefore, $Flow \subset P_{set}$. When a link failure is reported in G, as a consequence, the affected routes from that incident should be captured and added to the failed route set (F_r) as follows:

$$F_r = \{flow \mid flow \in Flow \wedge \exists_{v_i, v_j}.v_i, v_j \in flow \wedge OP(v_i, v_j) = 0\}$$

3.2 Analytical model

In order to incorporate the two networking events (i.e. failure and repair) in routing strategy, which dominates the routing changes, we have modelled the path transition state (i.e. from optimal to sub-optimal and vice versa) as depicted in Figure 1 in which f represents the negative trigger point (i.e. *failure*) that always leads to a sub-optimal state, while, $\bar{f}$ represents the positive trigger point of *failure* that drives into an optimal state, which might be the case of when there are a couple of equal-cost paths so that if one path fails then, the alternative will have an equivalent cost and therefore counted as an optimal. Unlike the failure triggering points, the *repair* r will always play a positive role and lead

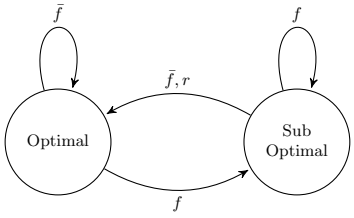

Fig. 1. Path state transition.

to an optimal state. Accordingly, the positive trigger point reflects a positive routing change and vice versa. Currently, we build this model on the basis on two aspects as follows:

1) *Frequency-based measurements*

Considering the networking events in Figure 1, We adopt a formal frequency-based definition of event probability [9]. that is, given n_{total} number of events (i.e. $f, \bar{f}$ and r), then the probability of getting a positive routing change, $\mathbb{P}_{change^+}$, that will lead to move across the optimal state, is approximated by the relative frequency:

$$\mathbb{P}_{change^+} \approx \frac{n_{change^+}}{n_{total}} \tag{1}$$

with the usual assumption that:

$$\mathbb{P}_{change^+} = \lim_{n_{total} \to \infty} \frac{n_{change^+}}{n_{total}} \tag{2}$$

Where n_{change^+} in our case represents the number of times that the positive triggering points (i.e. $\bar{f}, r$) have been occurred during the network operational time.

2) *Time-based measurements* Another aspect of why routing changes could be of benefit after every repair state is introduced in this section, which is discussing the optimal (i.e. primary) path utilisation. Figure 2 shows the time relation between failure and repair events as well as its impact on a particular $flow$. The red line shows the moments of when the link failure event occurs (i.e. F_1, F_2 and F_n) on a particular path, which also means that a new alternative (probably a sub-optimal) path should be selected when such a kind of events occur. Due to the fact that failure could be repaired before the next failure moment or spans after some successive failures, we have implemented the probability of repair through the concept of circle. We assume that b is the beginning time of when the *repair process* (ΔT_F) undertakes, which is typically conducted directly after every failure event.

On one hand, if the repair accomplished before the next failure event where e represents the maximum time that ΔT_F could be ended up before the next failure moment, therefore, (b, e) counts as the diameter of the largest circle (i.e. c3) on which the repair could be achieved. However, the repair might also finished at

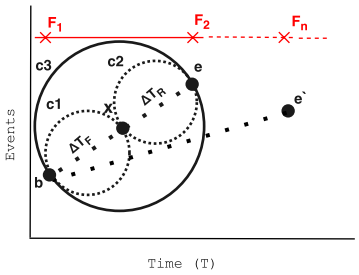

Fig. 2. Time relation between failure and repair events.

any point between b and e, for instance, consider the scenario of when the repair is done at x then, the circle (c1) with diameter (b, x) represents the amount of time that is required to restore the failure (ΔT_F); while the remaining area that composing the other circle (i.e. c2) with diameter (x, e) represents the *time window* that called ΔT_R, in which the probability of finding a better solution will be increased (as mentioned earlier in 3.2-1).

On the other hand, if the process of repair exceeds the next failure moment then, the same aforementioned scenario will be repeated but this time the largest constructed circle is between b and e', where e' refers to the maximum time that requires to fulfill the ΔT_F in a time span outside of the time between two failures. Since we are interested in maximising the chances of locating better paths with each repair trigger point, which in turn will maximise the utilisation of the better solutions, then we have the following possible outcomes:

- $\Delta T_R = null$: This is the case of c3, in which the repair process is accomplished at the same time of the next failure moment and therefore the area of c3 is consumed for merely ΔT_F.
- $\Delta T_R \neq null$: In this case, two circles (i.e. c1 and c2) are expected and the ΔT_R is usually associated with area of c2. In other words, the large c2 area the small c1 area (i.e. the fast repair process) and therefore the more opportunity to locate a better solution in an ideal time, which definitely leads to higher consumption of the obtained optimal solutions.

According to the literature, routing changes of the reactive fault tolerance techniques are appended only when there is a failure event, in other words, it ignores the changes after the repair and hence, in all the previous solutions the ΔT_R is not counted (i.e $\Delta T_R = null$). In consequence the per event utilisation percentage of better solution is given by:

$$\mathcal{U} = \left(\frac{\Delta T_R}{\Delta T_F + \Delta T_R}\right).100\% \tag{3}$$

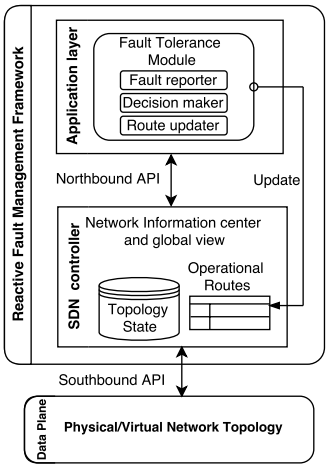

Fig. 3. The proposed framework.

4 Proposed Framework

From a high level point of view, Figure 3 illustrates the SDN proposed framework with an extended feature to the module of fault tolerance as follows:

- *Fault reporter* : When failure occurs and detected by the network controller, then, the fault tolerance module will be notified and this entity will report the information about the incidents. In further, this entity is not only reporting the failure scenarios, but also report the failed link when it comes back to service again.
- *Decision maker* : This entity relays on the information of the *Fault reporter*, which is necessary to maintain the affected route(s) through calculating either a new alternative routes when a failure state is reported or re-calculating the $Flow$ to make sure each $flow$ is in an optimal situation, otherwise, switch to the optimal path.
- *Route updater* : Finally, this entity is responsible for sending the required amendment of the *Decision maker* to the network controller, which in turn will push the updates as rules to all the relevant nodes in the data plane.

Because of the fault management mechanisms were merely invoke at the moment of failures, which will definitely affect the QoS as the action that taken by the Fault Tolerance Module does not meant to be as a permanent solution. Hence, we believe that this issue is not spotted yet and therefore we propose to append an additional route update after reporting a link repair state. In this context, Algorithm 1 shows how the proposed framework works under the conditions of both link activities: failure and repair.

According to Algorithm 1, most of the work of SDN fault management have considered the part of when failure is reported and most of the literature con-

Algorithm 1: Extended Routing Algorithm

On Normal: $\forall$ $flow \in Flow$: *Set Primary Path as* $flow \in P_{set}$
On Event : *Do the following procedure*

if *Link failure reported* **then**
  foreach $e_{ij} \in F$ **do**
    Compute: F_r
  end
  do
    $LF \leftarrow flow$
    $OF_{Remove}\ (flow)$
    $flow_{set} := flow_{set} - \{flow\}$
    $flow := \mathcal{D}(flow_{set})$
    $OF_{Install}\ (flow)$
    $F_r := F_r - \{flow\}$
  while $F_r \neq \emptyset$;
end
c := 0
if *Link repair reported* **then**
  do
    if $flow_c$ *is currently optimal* **then**
      Do nothing
      c := c + 1
    end
    if $flow_c$ *is currently sub-optimal* **then**
      $OF_{Remove}\ (flow_c)$
      $flow_c := \mathcal{D}(flow_{c_{set}})$
      $OF_{Install}\ (flow_c)$
      $LF := LF - \{flow_c\}$
      c := c + 1
    end
    if *number of links = len(G(E))* **then**
      $LF :=$ *empty*
    end
  while $c \leq len(LF)$;
end

tributions were revolve upon solving the failure scenarios efficiently, which in somehow dealing with a similar issue that illustrated by lines (1-13). However, lines (14-32) show the missing aspect that has to be taken into account in order to meet one of the main goals of the QoS, which is the optimal routing. In such a problem, distinguishing between the optimal and sub-optimal paths after every state of repair is required, to do so; we assume that each alternative (i.e. sub-optimal $flow$) is additionally stored in a special set called the *Labeled Flow (LF)*, where $LF \subset P_{set}$. Eventually, each repair report is associated with a specific one-out-of-three possible case that lead the update/change to be satisfied as follows:

case 1: When a current flow ($flow_c$) in LF is still considered as an optimal solution, in such a case the repaired link is not involved in the primary failed path of $flow_c$ and therefore the $flow_c$ should remain in LF. (Line 17-20)

case 2: When the current flow ($flow_c$) in LF is counted as a sub-optimal, which is the case when the repaired link gives a chance to construct a best path than the current one and hence, a change to the discovered path is required. (Line 21-27)

case 3: When the network is not experiencing any failure by the means of that all the previously occurred failures get repaired, while, there will be still some $flow$ reside in LF. In such a case, the LF must be cleared as the holding labeled paths have the same cost as the optimal ones and therefore, the routing change in this situation is useless and typically will lead to an unnecessary flaps. (Line 28-30)

5 Case study and discussion

5.1 Network topology

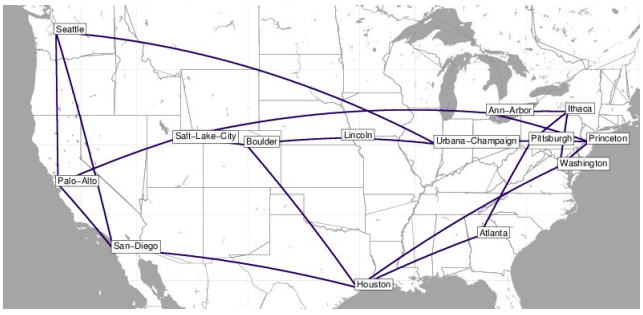

Fig. 4. Nobel-US network topology.

To evaluate the aforementioned model and framework, the Nobel-Us [10] with 14 nodes and 21 edges have been chosen to represent the realistic core network topology of the system as depicted in Figure 4.

5.2 Theoretical comparison

Table 1. Time analysis for link events and route changes

Network Activities			$Change^{-r}$		$Change^{+r}$	
T	Event	Operationalability	*Flow*	Action	*Flow*	Action
1	No event	{✔ , ✔}	$\{P_1, P_2\}$	-	$\{P_1, P_2\}$	-
2	L_1 Down	{✖ , ✔}	≡	Activate $P_1{}'$	≡	Activate $P_1{}'$
3	No event	{✔ , ✔}	$\{P_1', P_2\}$	-	$\{P_1', P_2\}$	-
4	L_1 Up	{✔ , ✔}	≡	-	≡	Activate P_1
5	No event	{✔ , ✔}	≡	-	$\{P_1, P_2\}$	-
6	L_2 Down	{✔ , ✖}	≡	Activate $P_1, P_2{}'$	≡	Activate $P_2{}'$
7	No event	{✔ , ✔}	$\{P_1, P_2{}'\}$	-	$\{P_1, P_2{}'\}$	-
8	L_2 Up	{✔ , ✔}	≡	-	≡	-

The purpose of this section is to compare the performance of the current reactive fault tolerance mechanisms that ignore the routing change after r, which we call ($Change^{-r}$), against the proposed strategy that considers r, which we call ($Change^{+r}$). To do so, we consider that only two $flow$ are currently belong to $Flow$. The first $flow$ is defined by $s =$ `Seattle` and $d =$ `Princeton`, whereas, for the second $flow$ $s =$ `Washington` and $d =$ `Salt-Lake-City`. By applying Dijkstra function ($\mathcal{D}$) on each $flow \in Flow$, we then get all the primary paths. For simplicity, we use the terms P_1 and P_2 for the primary paths of the first and second $flow$ respectively, while, we use the prime character (i.e. $\prime$) over a path to indicate that the $flow$ is currently operating under a sub-optimal state. At the beginning, all the network links/nodes should be in an operational state and therefore the optimal paths that will be discovered by Dijkstra are :

$P_1 =$ `Seattle, Urbana-Champaign, Pittsburgh, Princeton`

$P_2 =$ `Washington, Houston, Boulder, Salt-Lake-City`

We assume that there will be four network events to occur during the first eight time units since the network starts working as shown in Table 1 that has three main columns, namely, *Network Activities*, $Change^{-r}$ and $Change^{+r}$. The Network Activities shows the *Operationalability*, which reflects the state of each $flow \in Flow$ of whether it is working (i.e. usable ✔) or not (i.e. unusable ✖), according to the network circumstances (i.e. *Event*) that occur over the time (i.e. `T`). The $Change^{-r}$ and $Change^{+r}$ show the taken action(s) as well as the current held paths by $Flow$ in terms of optimality. When L_1, which is the link that connects `Seattle` to `Urbana-Champaign`, gets down at `T=2` then, a same action

will be taken by both strategies, which is to employ an alternative path for the first $flow$ as it has affected by the failure. The alternative path in this scenario is considered as a sub-optimal, since the cost of the primary is less than the alternative in terms of number of hops.

$P_1{}'$ = `Seattle, San-Diego, Houston,Washington,Princeton`

As a consequence, the $Flow$ end up with a new sub-optimal path (i.e. $P_1{}'$). After a while and when the link L_1 is repaired at `T=4`, the $Change^{+r}$ triggers a routing change by activating the primary path of P_1 since it is the optimal one, however, the $Change^{-r}$ will not react to this kind of events and therefore, no modification on $Flow$ at this time. When L_2, which is the link that connects between `Washington` and `Houston`, fails when at `T=6` then, the $Change^{-r}$ reacts to the failure by changing the two paths of $Flow$ through to P_1 and $P_2\prime$ respectively. In contrast, $Change^{+r}$ responds to the failure by activating the alternative of the affected P_2 only.

$P_2{}'$ = `Washington, Ithaca, Ann-Arbor, Salt-Lake-City`

Finally, when the link L_2 is repaired at `T=8` then, the $Change^{+r}$ will not append a routing change as the sub-optimal ($P_2{}'$) and the optimal (P_2) have the same cost and therefore the change is needless in such a case. As a result, it can be clearly seen how the $Change^{+r}$ succeeded to leverage the $\mathcal{U}_{P_1}$, which is up to 50% in the given example.

6 Conclusion

In this paper, we presented a new model that considers the issue of link repair and the positive role it plays in maximising the utilisation of optimal paths, in addition, we demonstrated how the proposed model can be implemented along with a new algorithm that appends two routing changes based on failure and repair occasions. The advantage of the proposed method is twofold. First, it increases the chance of relocating the optimal paths, and Second, it allocates the optimal paths as soon as they become available and hence the maximisation of utilisation. Theoretically, the provided case study shown the advantages of our method and answered the question of why such a model could be of benefit. In future, the proposed framework will be tested in a pure SDN environment, also, we will consider a more rigorous model to act as a SDN failure protocol.

References

1. Kreutz, D., Ramos, F. M., Verissimo, P. E., Rothenberg, C. E., Azodolmolky, S., & Uhlig, S. (2015). Software-defined networking: A comprehensive survey. *Proceedings of the IEEE, 103*(1), 14-76.
2. McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., ... & Turner, J. (2008). OpenFlow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review, 38*(2), 69-74.

3. Akyildiz, I. F., Lee, A., Wang, P., Luo, M., & Chou, W. (2016). Research challenges for traffic engineering in software defined networks. *IEEE Network, 30*(3), 52-58.
4. Fonseca, P., & Mota, E. (2017). A Survey on Fault Management in Software-Defined Networks. *IEEE Communications Surveys & Tutorials*.
5. Poretsky, S., Imhoff, B., & Michielsen, K. (2011). *Terminology for Benchmarking Link-State IGP Data-Plane Route Convergence* (No. RFC 6412).
6. Luo, M., Zeng, Y., Li, J., & Chou, W. (2015). An adaptive multi-path computation framework for centrally controlled networks. *Computer Networks, 83,* 30-44.
7. Mendiola, A., Astorga, J., Jacob, E., & Higuero, M. (2016). A survey on the contributions of software-defined networking to traffic engineering. *IEEE Communications Surveys & Tutorials, 19*(2), 918-953.
8. E. W. Dijkstra, E., W. (1959, December). A note on two problems in connexion with graphs. *Numerische Mathematik, 1*(1), 269-271.
9. R. von Mises , Probability, Statistics and Truth, William Hodge & Co., Ltd., 1939 .
10. SNDlib library. [Online]. Available: http://sndlib.zib.de.