Designing a Disaster-resilient Network with Software Defined Networking

An Xie[*], Xiaoliang Wang[*], Guido Maier[†] and Sanglu Lu[*]

[*] National Key Laboratory for Novel Software Technology

Nanjing University, Nanjing, P.R. China

Email: waxili@nju.edu.cn

[†]Dipartimento di Elettronica, Informazione e Bioingegneria

Politecnico di Milano, Milano, Italy

Email: guido.maier@polimi.it

Abstract—**With the wide deployment of network facilities and the increasing requirement of network reliability, the disruptive event like natural disaster, power outage or malicious attack has become a non-negligible threat to the current communication network. Such disruptive event can simultaneously destroy all devices in a specific geographical area and affect many network based applications for a long time. Hence, it is essential to build disaster-resilient network for future highly survivable communication services. In this paper, we consider the problem of designing a highly resilient network through the technique of SDN (Software Defined Networking). In contrast to the conventional idea of handling all the failures on the control plane (the controller), we focus on an integrated design to mitigate disaster risks by adding some redundant functions on the data plane. Our design consists of a sub-graph based proactive protection approach on the data plane and a splicing approach at the controller for effective restoration on the control plane. Such a systematic design is implemented in the OpenFlow framework through the Mininet emulator and Nox controller. Numerical results show that our approach can achieve high robustness with low control overhead.**

I. INTRODUCTION

Networks having very high degree of interconnection are vulnerable to the disruptive events such as floods, earthquakes, power outages, electronic attacks, etc. Such regional damages are usually unpredictable and may simultaneously destroy multiple network facilities in a specific geographical area, which result in a long period of network outages. For example, the east Japan earthquake on March 2011 caused 385 telephone offices stopping operation immediately, cut off millions of users from the telephone service and even the emergency restoration took more than one month [1].

The conventional techniques to maintain network continuity can not work well in case of disasters. Network protection, which relies on the expensive pre-allocated backup resources, may fail to deal with regional damage when the backup resources corrupt simultaneously with the primary ones. The restoration mechanism, which computes new routes based on the actual status of network, may introduce too long convergence time to meet the requirements of mission-critical and real-time applications, and leads to serious consequences like transient loops and blackholes [2].

To build disaster-resilient networks, this paper focuses on leveraging the technique of SDN (Software Defined Network-

ing) which provides more intelligent and flexible network management. SDN networks, such as OpenFlow - enabled [3] networks, decouple the network control plane from the data plane, and have been successfully deployed in the operator's WAN and corporation's LAN to provide robust network services, e.g., the global carrier NTT communications networks, Google and Microsoft inter-datacenter WANs, etc [4], [5]. Due to its intrinsic great flexibility and global management of the network, SDN is potentially suitable to execute an efficient recovery during a major disruption.

Although SDN has a good potential for handling failures, the current architecture may be not sufficient to recover from large-scale failures such as disaster failures. *Control plane scalability* and the *recovery time requirement* are the two major challenges. Generally, the SDN controller computes routes whenever a failure occurs. And the controller is responsible for updating all the forwarding elements' status. However, the multiple failures caused by a catastrophic event will simultaneously disrupt a lot of end nodes. This will lead to a huge amount of reconnection requests, making it impractical to offload the task of all the routing computation and to update the forwarding elements' status to the controller. This is because the dynamic route re-computation can lead to huge overhead, and inserting all new routes into SDN forwarding elements alongside is time-consuming [6] and error-prone due to the consistent packet processing problem [7], [8]. Moreover, the stringent recovery time requirements of mission-critical and real-time applications [9] make the enhancement design of the control plane (e.g., the distributed control plane design like Onix [10]) incompetent. This is because the status synchronization among physically distributed controllers requires additional time.

In this paper, we propose a new framework to deal with the considered problems in face of disaster failures by SDN. Several design challenges are addressed in this paper,

(1) *Low controller overhead*. The controller overhead should be low in order to reduce the likelihood of controller being the bottleneck.

(2) *Fast recovery*. The recovery should be quick in order to meet the requirement of some mission-critical and real-time applications.

(3) *Strong connectivity*. The connectivity ratio should be

high even after a major disaster event destroying a lot of network components.

To address above challenges, our main idea is to pre-install redundant flow entries (backup entries) into the data plane. Different from the previous enhancement design of the control plane, our enhancement design of the data plane guarantees that a large proportion of the reconnection requests can be handled on the data plane. Since only a small fraction of the requests are handled by the controller, the control overhead is low. Besides, the data plane handled requests will not be sent to the control plane, thus saving the round-trip recovery delay between the data plane and the control plane. To address the third challenge, we consider the disaster failure's geographical layout and its failure size distribution. In order to do this, we adopt the novel metric of the vulnerable zone of a path during generating the backup entries. So the pre-installed backup routes are less likely to simultaneously get destroyed by a disaster failure. By combing with the recovery on the control plane, our design guarantees strong connectivity after a disaster failure.

More concretely, our proposed design consists of two modules: the proactive local failure recovery module running on the switches (data plane) and the reactive global restoration module running on the controller (control plane). In the protection module, we adopt the multi-topology routing to do local fast rerouting, and consider the geography properties (shape and size) of the disaster failure to generate robust backup routes. In the restoration module, we give an effective algorithm to reconnect failed nodes by rescheduling the pre-installed routes. We further consider the load balance performance during recovery by formulating an ILP, after which an heuristic algorithm is proposed. We implement the prototype by utilizing multiple tables pipeline processing and fast failover group tables of OpenFlow (Section III). Simulations (Section IV) on both random generated and realistic topologies show that, the protection module is able to handle approximately 70% of the reconnection requests. The rests are processed by the restoration module. Only by rescheduling the pre-installed redundancies, more than 90% of the disconnected end nodes can be reconnected even when the failure diameter is $1/6$ of the network deployment region's length.

II. PRELIMINARY

In this section, we first introduce the network model and failure mode adopted in this paper. Then we introduce the vulnerable zone of a routing path.

A. Network Model

We consider a physical network $G(V, E)$ as a planar graph inside the deployment area $D \in \mathbb{R}^2$, which is represented by the network components: V is the set of forwarding elements (routers or SDN switches) and E is the set of links connecting them. In SDN context, all forwarding elements have a channel connected to the central *controller* C (in-band or out-of-band)[1]. By e_{ij} we denote the link between adjacent nodes

---

[1]The logical controller *C* can be implemented distributedly [11], [12], this refers to the controller placement problem and is out of the scope of our works.

(a) vulnerable zone of link e_{ij} (b) vulnerable zone of path x_{st}
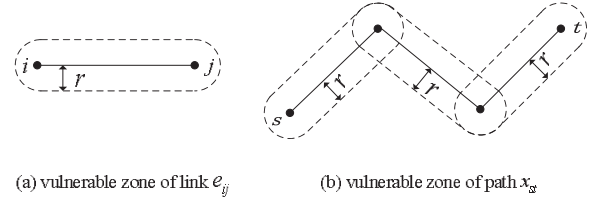
Fig. 1.   Vulnerable zone: the union of points that are located no more than r distance from the network components. Any region failure occurs in the vulnerable zone will break the network component.

i and j, $i, j \in V, e_{ij} \in E$. By x_{st} we denote the path between nodes s and t, $s, t \in V$.

B. Failure Model

During the extreme events such as disasters or malicious attacks, multiple network components located closely to each other may fail together. We summarize the behaviors of such large scale attacks to model the "the geographically correlated failure".

Definition 1: (**Geographically Correlated Failure**) is defined as follows:

1) Network components intersecting the region of failures will be removed from the network. The size of a geographically correlated failure is determined by the radius r.
2) The radius r follows the distribution functions $f(r)$, $r_a \leq r \leq r_b$, where r_a (resp. r_b) is the minimum (resp. maximum) considered region size[2].

It's notable that our model does not make any assumptions about the failure locations and radiuses, which are usually difficult to obtain due to the uncertainty of the disaster failures. Our model is more general than the previous deterministic failure model [14], [15] (which requires the knowledge of the failure radiuses) and SRLG related model [16]–[21] (which requires the knowledge of the failure locations).

C. Vulnerable Zone of a Path

According to the definition of regional failures, a disaster region can be of any shape with arbitrary size and located anywhere in the plane. Therefore, there are infinite number of region failures to be considered. Our first problem is to find a proper statistical metric to evaluate the impact of region failures.

Given a regional failure with radius r, a link e_{ij} may fail if it intersects with the failure region. In other words, if a disaster happens and its epicenter is less than r distance from e_{ij}, e_{ij} will be broken. We call the set of those points the *vulnerable zone* of link e_{ij}, denoted by $\boldsymbol{Z}_{e_{ij}}^r$, defined as follows:

Definition 2: (**Vulnerable zone of a link**) is the region sub-area such that any region failure with radius r whose epicenter falls within it will always cause the corruption of the given link.

---

[2]The distribution function $f(r)$ of the destructive natural regional failures, such as earthquakes, usually follows the power-law distribution [13].

As illustrated in Fig. 1 (a), the "hippodrome" in dash line represents the vulnerable zone of link e_{ij}, which consists of all points whose shortest distance to link e_{ij} is no more than r. Similarly, we can further define the vulnerable zone of a path x_{st}, denoted as $\boldsymbol{Z_{x_{st}}^r}$, as shown in Fig. 1 (b), which is the union of all circle centers that are located no more than r distance from the path.

Definition 3: (**Vulnerable zone of a path**) is the region sub-area such that any region failure with radius r having epicenter falling within it will always cause the corruption of the given path.

The vulnerable zone of a path x_{st} is the union of the vulnerable zones of all the links of the path, i.e., $Z_{x_{st}}^r = \cup_{e_{ij} \in x_{st}} Z_{e_{ij}}^r$.

III. SYSTEM DESIGN

In this section, we first define the problem of SDN network reliability against regional damage. Then we introduce our system, which consists of two modules: the proactive Backup Topologies Generation Module for local recovery and the reactive Splicing Module for global restoration.

A. Overview of System Design

The problem to be solved can be defined as follows: *Given a network $G(V, E)$ and a central controller C, 1) how do we pre-install some redundancies into the network so that the controller is able to reschedule these reduncancies and 2) how does the controller reschedule the protection resources with low controller overhead to survive from the large-scale multiple failures caused by regional damage.*

To solve the above problem, we apply the SDN framework for failure recovery. Our design consists of two modules, a proactive local failure recovery module working in the forwarding plane (Backup Topologies Generation Module) and a reactive global restoration module running in the control plane (Splicing Module). The failure reconnection requests first get handled by the local failure recovery module. The reconnection requests that the local recovery module is not able to handle are led to the global restoration module as illustrated in Fig. 2.

How to install the redundancies for the proactive local failure recovery module needs to be carefully addressed. To reduce the controller overhead, we want to handle failures locally as much as possible. However, the limited number of redundancies on the data plane can not handle all of the failures. We thus have to distinguish the types of failures, so as to decide which of them to handle on the data plane and which on the control plane. This "distinction" function can only be implemented on the data plane. Otherwise it would require the interference of the controller which may lead to some additional controller overhead (to decide the types) and additional recovery delay (the round trip time between the control plane and the data plane). To make the distinction on the data plane possible, we refer to the approach of Multi-Topology (MT) Routing (RFC 4915 and RFC 5120 [22], [23]) to add redundancies. And by the joint design of multi-topology redundancies and the multi-table pipeline

processing of OpenFlow, this distinction function is made possible without any interference of the controller.

The basic idea of MT routing is to take the original graph G as input, and generate k backup topologies $\{G_1 \ldots G_k\}$. Routing tables $\{T_1 \ldots T_k\}$ are computed and installed based on $\{G_1 \ldots G_k\}$. Moreover, we notice that the recent research on MT Routing, Multiple Routing Configurations (MRC) [24], [25], is a good technique. The design goal of MRC is to prepare different configurations for different single node or link failures to achieve fast rerouting. With the adoption of MRC, the goal of the distinction function is clear, i.e., to distinguish the single link or node failure and the multiple failure. Furthermore, during the route planning in each backup topology, we consider the geographical distribution of network components to reduce the likelihood of route corruptions by regional failures.

For the reactive global recovery module running on the control plane to handle the remaining failures, a straight-forward idea is to compute new routes on the controller for each failed flow and install all the new rules into the corresponding switches ([12], [26], [27]). However, such an operation is time consuming and error-prone due to the consistent packet processing problem [7], [8]. Instead, we exploit the usage of pre-computed backup topologies to rebuild the failed connections, and a splicing algorithm is proposed in the Splicing Module at the controller to find new paths.
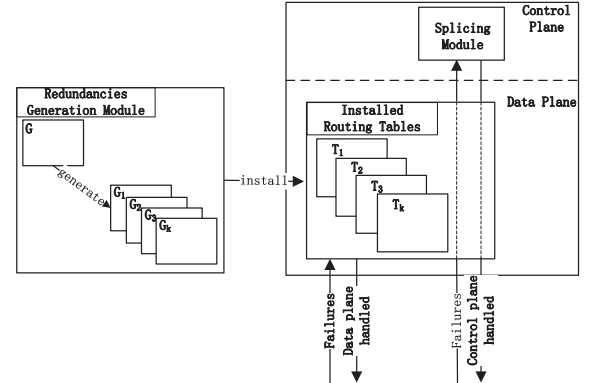

Fig. 2. System overview

B. Review of MRC (Multiple Routing Configurations)

For the completeness of our work, we first give a brief review of the MRC algorithm. The key idea of MRC routing algorithm is to prepare multiple backup topologies $\{G_1 \ldots G_k\}$, and select a proper backup topology in accordance with the current network failure state [25]. In each backup topology G_i, some links $e_{u_i v_i}$ are defined as *isolated links* and *restricted links* while some nodes u_i are defined as *isolated nodes*. The isolated links are set infinite weight and can be excluded from G_i. The restricted links are set very high weight so that they will not be chosen by some routing algorithms (i.e., shortest path routing mechanisms) unless have to. A node u_i in G_i is isolated if and only if its adjacent links are all either restricted or isolated. Whenever the isolated nodes fail, it will not affect the connection of other paths.

If node u detects a failure of adjacent link e_{uv}, u will select a backup topology G_i in which the failed next hop link e_{uv} is isolated. Then it will tag packets with the selected backup topology id i to notify the subsequent node to forward packets based on this backup topology. Since in G_i, $e_{u_i v_i}$ is assigned a very high weight and it does not undertake any transit traffic, the packets are guaranteed to reach their destination.

Generally, to restore an arbitrary single link or node failure, the following constraints must be satisfied:

(1) Each node u in the original graph must be isolated in at least one backup topology G_i. Each link e_{uv} in the original graph must be isolated in at least one backup topology G_i.
(2) Each link must be isolated with one of its adjacent isolated nodes in one backup topology.
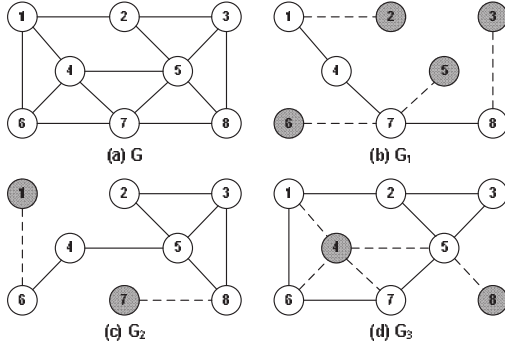(3) All node pairs must be mutually reachable in G_i.

Fig. 3. (a):Original network G.(b)-(c):backup topologies G_1-G_3. Dark node refers to the isolated node and dashed line refers to the restricted link. For clarity, we did not draw the isolated links in G_i.

Fig. 3 shows the generated backup topologies. Every node is isolated in exactly one backup topology. Consider a flow with (src=1,dst=3): normally its path is $1 \rightarrow 2 \rightarrow 3$ based on shortest path in G. Assume node 2 failed and node 1 detected the failure. Since node 2 is isolated in G_1, node 1 would tag the packet with tag 1 which refers to backup topology G_1. The alongside nodes will also forward the packet belonging to the failed flow based on G_1. Thus, the routing path from 1 to 3 becomes $1 \rightarrow 4 \rightarrow 7 \rightarrow 8 \rightarrow 3$.

The MRC is originally designed for locally handling single link or node failure. It is not adequate to handle multiple failures caused by large-scale regional failure. To leverage its redundancy, we first modify it based on the geographical distribution of failures to better accommodate the Splicing Module as described in the next subsection.

C. Backup Topologies Generation Module

This section first introduce how to generate routes on these backup graphs $\{G_1 \ldots G_k\}$ obtained by the MRC algorithm. Even with a sophisticated path finding algorithm, it's impossible for the limited number of redundancies on the data plane to handle all of the failures. So this module is also responsible for distinguishing the failure types so as to deliver some failures to the control plane to get handled.

1) Geography based Backup Route Generation: After running the MRC backup topology generation algorithm, we obtain multiple backup topologies $\{G_1 \ldots G_k\}$. If we run the same path finding algorithm (i.e., Dijkstra algorithm) on all these backup topologies, for specific nodes s and t, the path between them, x_{st}^i (on G_i) and y_{st}^j (on G_j) , will be possibly coincided. This should be avoided because paths on these backup graphs are too close to each other and are vulnerable to a common risk. To improve the reliability of these paths, we adopt the vulnerable area of a path during the route generation in each G_i.

For each $s, t \in G$, we may have different paths on each G_i. Compared to the original path in G, these paths are redundant. They can be used to rebuild the failed connection between s and t. However, if the vulnerable areas of backup paths intersect, they can be possibly destroyed by a regional failure simultaneously. Consider a flow with (src=6, dst=3): normally its primary path in G is $6 \rightarrow 7 \rightarrow 5 \rightarrow 3$ based on the shortest path. The backup path from 6 to 3 in G_1 is $6 \rightarrow 7 \rightarrow 8 \rightarrow 3$. The backup path in G_2 is $6 \rightarrow 4 \rightarrow 5 \rightarrow 3$. The backup path in G_3 is $6 \rightarrow 7 \rightarrow 5 \rightarrow 3$. Assume that a regional failure destroys node 5 and node 8 simultaneously. All the primary and backup paths are destroyed. This is because in this case, the primary path and the three backup paths are not region-disjoint. If, however, in G_3, we select the region-disjoint path $6 \rightarrow 1 \rightarrow 2 \rightarrow 3$ from $6 \rightarrow 7 \rightarrow 5 \rightarrow 3$ in G_1, backup path in G_3 would not get destroyed by the regional failure.

To avoid the situation in which all the primary path and backup paths are destroyed, it is required that these paths are region-disjoint [28]. However, finding region-disjoint paths is NP-hard even with a fixed failure radius r [28] and is difficult to solve in general. Therefore, we refer to heuristic algorithms. Our algorithm is shown in Algorithm 1. It first finds the shortest path x_{st}^0 from s to t on G (G_0) as the primary path between s and t. Then it iterates on all the backup topologies. $[r_a, r_b]$ is evenly divided into k intervals. Each backup topology G_i is resilient to failures with radius up to $r_i = r_a + (i - 1) \cdot \frac{r_b - r_a}{k-1}$. This is achieved by reducing the likelihood of the vulnerable zone of backup path, $Z_{e_{uv}}^{r_i}$ intersecting with the vulnerable zone of the primary path, $Z_{x_{st}^0}^{r_i}$. If the two vulnerable zones intersect, it means that they can be both destroyed by a failure with radius r_i. We assign very high weight to those links whose vulnerable zones intersect with $Z_{x_{st}^0}^{r_i}$ to reduce the likelihood of choosing those links in y_{st}^i.

2) Implementation: If the packet can not be handled on the data plane, they are sent to the controller to get handled on the control plane. The data plane should be able to deliver the failures to the control plane immediately after finding itself unable to handle them. Unlike the traditional router, the data plane and the control plane in SDN are usually physically separated. Also, the controller should not interfere with this logic. Otherwise, it would prolong the recovery time (due to the round trip time between switches and the controller) and increase the controller overhead. We achieve this by carefully arranging the routes in the pipeline and leveraging the fast failover group table provided in the OpenFlow.

Algorithm 1: Backup Routes Generation

Input: network topology $G = (V, E)$, backup topologies
$\{G_1 \ldots G_k\}$, source address s, destination address
t

Output: k backup routes in $\{G_1 \ldots G_k\}$

1 begin

2   Find x_{st}^0 in the original topology G

3   **for** $i \leftarrow 1$ **to** k **do**

4 $r_i := r_a + (i-1) \cdot \frac{r_b - r_a}{k-1}$

5   **forall the** *edge* $e_{uv} \in E_i$ **do**

6   **if** $Z_{e_{uv}}^{r_i} \cap Z_{x_{st}^0}^{r_i} \neq \emptyset$ **then**
          $w_{e_{uv}}^i := very\ high\ weight$

7   Find y_{st}^i in backup topology G_i using the new
        weight

8   **return** $\{y_{st}^1 \ldots y_{st}^k\}$

Generally, the OpenFlow pipeline processing consists of multiple routing tables $\{T_0 \ldots T_{max}\}$ and a group table T_g. Flow entries both in T_i ($0 \leq i \leq max$) and T_g consist of a lot of terms. In T_i, entries consist of match fields, instructions and priority. The failover group entries in T_g, consist of group id and action buckets. Each action bucket is associated with a specific port (watch port) that controls the bucket's liveness. The action buckets within an entry are evaluated sequentially. The first bucket which is associated with a live port is selected.

The conventional routing procedure is to directly forward a packet p to a specific port. In contrast, to leverage the fast failover group, we first forward p to a specific group in the group table. Then it's up to the group to decide which port to forward to based on the port's liveness. The packets are basically divided into two types, i.e., clean packets and dirty packets. The clean packets are those packets that have not encounter any failure yet via routing. The dirty packets have encountered failures before. The clean packets and the dirty packets are processed by different processing flows in the multiple table pipeline. The two types of packets are explicitly distinguished by the MPLS tag in the packet header.

The detailed procedure is shown in Fig. 5. Concretely, T_0 is the starting table for all packets, which works as a diverting table to divert packets to different processing flows.

(1) A clean packet p is diverted by T_0 to one of the groups in T_g. The group which T_0 forwards p to, is responsible for checking the liveness of the output port which is based on routing table T_0. If the output port is alive, p will be sent out via that output port. Otherwise, it will be tagged a MPLS label ø (ø is a backup topology's number) to indicate that it is a dirty packet. Then it will be sent out via another port which is decided by the routing in $T_ø$.

(2) A dirty packet p with a MPLS label i is diverted by T_0 to T_i. Then T_i will forward p to one of the groups in T_g. The group will check the liveness of the output port based on routing table T_i. If the port is alive, p will be sent out, otherwise it will be sent to the controller to get further processed.
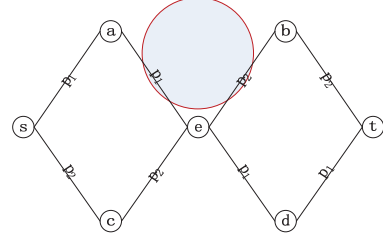

Fig. 4. A regional failure destroys both $p1$ and $p2$. We rebuild the path by installing one route on node e to divert traffic from $p1$ to $p2$.

D. Splicing Module

The splicing module refers to the *reactive* recovery at the controller. It's responsible for rebuilding the failed connections that can't not get handled by the pre-installed redundancies. As described in Section III-A, unlike conventional approaches that install all routes into forwarding elements, we rebuild the failed connections by utilizing the pre-installed redundancies. By doing so, the number of installed routes is reduced, thus reducing the likelihood of the consistent packet processing problem. A motivation example is shown in Fig. 4. The are two paths from s to t, $p1$ and $p2$. A regional failure destroy e_{ae} and e_{eb} simultaneously. As a result, both $p1$ and $p2$ are destroyed. To rebuild the connection, the traditional approach is to install new routes along $s \rightarrow c \rightarrow e \rightarrow d \rightarrow t$, which requires installing five new rules. In contrast, we only install two rules, one on node s to divert traffic to $p2$, the other one on node e to divert traffic from $p2$ to $p1$.

Another issue we consider is the load balancing during recovery. Unlike the single link or node failure, the regional failures usually destroy a huge amount of network components simultaneously and lead to a huge amount of disconnected end nodes. Such a huge number of disconnected end nodes requires lots of reconnections. The splicing module should handle the reconnections in a proper way to avoid that some nodes bear exceedingly more rerouting paths than others. The requests that sent to the controller give us the opportunity to redistribute some of the reconnecting traffic. Aside from the connectivity, we also consider the problem of how to reconnect the failed paths. We first define a metric, then we give an ILP to formulate the problem, after which we propose an efficient heuristic algorithm to reduce the complexity.

We define the metric, maximal load to quantify the routing load balance degree after a regional failure.

Definition 4: (**Maximal Load**): given a network graph $G'(V', E')$ after a regional failure (with failure radius r), and a reconnection request matrix $\mathcal{RM}$ on G', the maximal load gap is the load of the most loaded node in the network, where the load of a node is the number of primary and rerouted paths passing it, i.e.,

$$ML = |(P_u + R_u')|_{max}, \forall u \in V'$$

By P_u we denote the number of primary paths that pass u. By R_u' we denote the number of the rerouted paths that pass u based on $\mathcal{RM}$ after a regional failure. Here, we consider the primary resource and the rerouted resource separately, and we assume that if a primary path for a particular node pair is not

failed, the primary path can not be altered to avoid network wide reconfiguration. Our goal is to minimize the maximal load.

We formulate the problem by the following ILP,

$$\min_{i \in V'} \max \left(\sum_{\forall s,t \in V} \sum_{e_{ij} \in E} x_{ij}^{st} + \sum_{\forall s,t \in \mathcal{RM}_1} \sum_{e_{ij} \in E'} y_{ij}^{st} \right.$$
$$\left. + \sum_{\forall s,t \in \mathcal{RM}_2} \sum_{e_{ij} \in E'} z_{ij}^{st} \right) \quad \text{(ILP1)}$$

$$\text{s.t.} \quad z_{st} \text{ is a routing path} \quad (1a)$$
$$y_{ij}^{st} \in \{0,1\}, \forall e_{ij} \in E, \forall s,t \in V' \quad (1b)$$

The first part in the object function corresponds to P_i, in which x_{ij}^{st} equals to 1 if path x_{st} passes e_{ij} and 0 otherwise. x_{ij}^{st} is computed based on the routing in G_0. The second part in the object function corresponds to the additional rerouting paths that i have to undertake due to the reconnection requests $\mathcal{RM}_1$. $\mathcal{RM}_1$ is the reconnection requests that can be handled on the data plane. The second part can also be computed. The third part in the object function computes the number of rerouting paths that i have to undertake due to the reconnection requests $\mathcal{RM}_2$. $\mathcal{RM}_2$ is the reconnection requests that can not be handled on the data plane. It is notable that $\mathcal{RM}$ equals to the sum of $\mathcal{RM}_1$ and $\mathcal{RM}_2$.

ILP1 distribute the rerouting traffic in a min-max fashion. However, the above optimization may not scale well to large networks. In addition to the optimization, we also give a heuristic algorithm. To evenly distribute the reconnections request using the installed redundancies, for all node $u \in E$, we record R'_u on the control plane. We first construct a temporary graph in which, we fill all the available segments (not broken by the regional failure) from $\{G_1 \ldots G_k\}$ into the temporary graph. Then weight assignment is performed based on R'_u for all node $u \in E$. The detailed algorithm is in Algorithm 2.

In Algorithm 2, we first test if s and t are physically disconnected. If they are, it's impossible to find a path between them. Then we construct a multigraph G_{temp} by adding edges from k paths from s to t in $\{G_1 \ldots G_k\}$ excluding the failed edges. To evenly distribute the rerouting paths, we set link weight of e_{uv} to the mean of R'_u and R'_v. After the weight is set, we try to find a path on this temporary graph. If a path is found, we update $R'_u, u \in V$ and return the splicing actions. Otherwise, it means that the failed path can not be rebuild by splicing the existing redundancies. In this case, one may have to install a whole new path.

IV. PERFORMANCE EVALUATION

A. Simulation Setting

We use both random and realistic topologies for our simulation. The random50 topology contains 50 nodes and 120 edges, the random100 topology contains 100 nodes and 211 edges. The realistic Germany backbone consists of 50 nodes and 88 edges. The deployment area is 1200 x 1200 (arbitrary units) for all the cases. One connection is requested by each node pair of the network. We implement our prototype using

---

Algorithm 2: Splicing Action Generation

Input: network topology $G = (V, E)$, backup topologies $\{G_1 \ldots G_k\}$, source address s, destination address t

Output: a set of splicing actions.

1 **begin**
2   **if** *s and t are physically disconnected* **then**
3   **return** failed and abort
4   **else**
5   build a temporal topology $G_{temp}(V_{temp}, E_{temp})$
6     $V_{temp} := V, E_{temp} := \emptyset$
7   **for** $i \leftarrow 1$ **to** k **do**
8   **forall the** *edge* $e_{uv} \in E_i$ **do**
9   **if** e_{uv} *is alive and on the path from s to t in* G_i **then**
10           $E_{temp} := E_{temp} \cup e_{uv}$
11           $\ell_{e_{uv}} := i$
12   **else**
13   continue
14   **forall the** *edge* $e_{uv} \in E_{temp}$ **do**
15       $w_{e_{uv}} := (R'_u + R'_v)/2$
16   find shortest path p_{temp} on G_{temp} from s to t
17   **if** *found* **then**
18   **forall the** *edge* $e_{uv} \in E_{temp}$ **do**
19   **if** $e_{uv} \in p_{temp}$ **then**
20           $R'_u := R'_u + 1$
21           $R'_v := R'_v + 1$
22   **else**
23   **return** failed and abort
24   **return** splicing actions based on p_{temp}

OpenFlow 1.3.3 [29] and NOX controller [30] and examine the prototype's performance on Mininet testbed [31]. For the throughput test, we use two PCs, one running mininet and the other running NOX controller. Iperf [32] is adopted as our test tool.

Comparsion Metrics. We use the following metrics to quantify the results [33], [34].

- *Recovery Ratio*. Recovery ratio is introduced to evaluate the capacity of network recovery to reset the connection between pairs of disconnected nodes, which can be defined as follows,
  Definition 5: **(Recovery Ratio)**

  $$\text{Recovery Ratio} = \frac{\text{number of recovered paths}}{\text{number of recoverable paths}}$$

  As a result of multiple failures, the underlying topology may be divided into disconnected components, or the source (and/or destination) of a certain flow becomes failed. Hence, we call a disconnected routing path as "recoverable" if both the end nodes are alive and they are not physically separated.

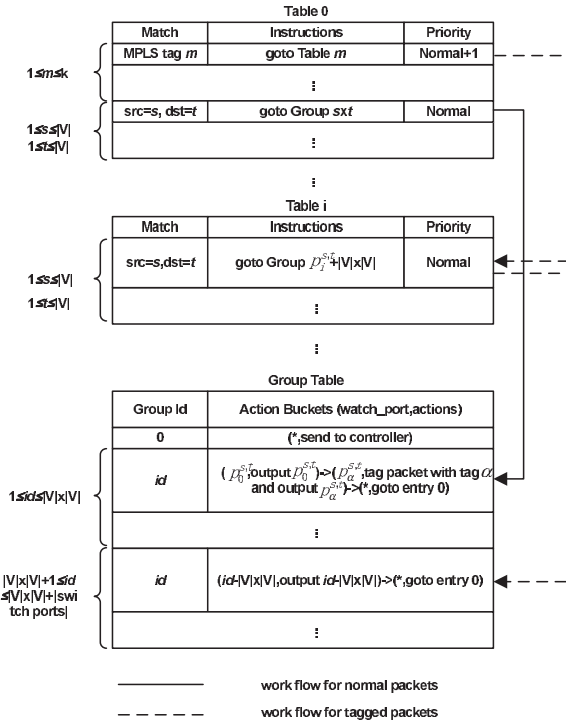- *Path stretch*. The detail definition can be found in [35].

Table 0

Match	Instructions	Priority
MPLS tag m	goto Table m	Normal+1
$\vdots$		
src=s, dst=t	goto Group sxt	Normal
$\vdots$		

$1 \leq m \leq k$

$1 \leq s \leq |V|$
$1 \leq t \leq |V|$

Table i

Match	Instructions	Priority
src=s,dst=t	goto Group $p_i^{s,t}$+|V|x|V|	Normal
$\vdots$		
$\vdots$		

$1 \leq s \leq |V|$
$1 \leq t \leq |V|$

Group Table

Group Id	Action Buckets (watch_port,actions)
0	(*,send to controller)
id	$(p_0^{s,t}$,output $p_0^{s,t}) \rightarrow (p_\alpha^{s,t}$,tag packet with tag α and output $p_\alpha^{s,t}) \rightarrow$(*,goto entry 0)
$\vdots$	
id	(id-|V|x|V|,output id-|V|x|V|)$\rightarrow$(*,goto entry 0)
$\vdots$	

$1 \leq id \leq |V|x|V|$

|V|x|V|+1$\leq id$
$\leq$|V|x|V|+|switch ports|

—————— work flow for normal packets

- - - - - work flow for tagged packets

Fig. 5. Prototype architecture

Generally, a path with longer stretch requires more network resources. We adopt the notation of *stretch* to measure the ratio of alternate path length over the expected shortest path length.

- *Controller overhead.* As the aforementioned idea of backup topology generation, we try to reduce the load of the controller by locally restoring the failed connection. The effectiveness of this approach is measured through the metric below.
  Definition 6: (**Controller Overhead**) Controller overhead is defined as the proportion of reconnection requests that need to be processed by the controller.
- *Maximal Load.* As defined in Section III-D, it quantifies the maximal load among nodes after a regional failure.

We compare our SDN-based Fast and Resilient Routing against Disaster (SDN-FRRD) approach with the following approaches proposed in the literature,

- MRC [25]. The MRC is designed for local fast recovery. We evaluate it to see if it's sufficient for regional failures.
- SDN-MRC. We apply the MRC to our novel framework, by directly using the MRC in the Backup Topologies Generation module.
- Path Splicing [33]. The advanced multipath routing algorithm, path splicing, is to random splicing routes in the data plane. The setting of Path Splicing is: using the same number of k backup topologies as MRC and SDN-MRC, the link weight perturbation function is: $weight(i,j) = (degree(i) + degree(j))/degree_{max}$ where $degree_{max}$ is the maximal node degree and $weight(i,j)$ ranges from 0 to 2.

B. Evaluation Results

1) Recovery Ratio: Fig. 6 shows the recovery ratio in term of the number of backup topologies k in the three topologies when the radius of the regional damage is 50. From the graph, we can see that both SDN-FRRD and SDN-MRC, that apply the SDN framework can steadily achieve more than 90% recovery ratio. Comparing to the MRC and the Path Splicing curves, clearly shows the effectiveness of our SDN framework.

Fig. 7 shows the recovery ratio when the radius of the regional failure is 100. The trend of the curves is similar to the ones inFig. 6. When the failure radius is 100, the failure breaks more links than when the failure radius is 50. Thus the recovery ratio of the MRC, SDN-MRC, Path Splicing gets decreased. For example, the recovery ratio decrease by about 5% in Fig. 7(a) compared to Fig. 6(a), and decreases by about 10% in Fig. 7(c) compared to Fig. 6(c). However, we observe no significant decrease of the curve SDN-FRRD. This is because in the Backup Topologies Generation module (see Algorithm 1), we adopt the vulnerable area of a path and consider the distribution of the failure radius to generate backup routes, such that the recovery ratio is not significantly influence by the size of the regional failure. This can also be validated in Fig. 9, where the recovery ratio remains above 95% even when the failure radius is 150 in all the three topologies.

Since the recovery ratio of the SDN-FRRD is almost about 100% and is steady when the number of the backup topologies k is from 6 to 15. Since small k already has satisfying performance of the recovery ratio, small values of k is sufficient. Because larger k means the backup tables would consume more switch resources, network operators who have a strict limitation of switch resources can consider choosing the smallest k.

2) Stretch: Fig. 8 shows the stretch in term of $k = 6, 7, 8, 9$ in the three topologies. As we can see, in all the three topologies, about 90% of the stretch is below 1.5. Normally, larger k means more redundancies, which can lead to smaller stretches. The four values of k achieve approximately equal recovery ratio, larger k tends to have smaller stretch. This can be seen, for example, in Fig. 8(c), the curve of $k = 9$ is on the left side of the curve of $k = 6$, which means a smaller stretch. This leads to a trade off between cost and performance at the initialization of network, i.e., operators who want to get a lower stretch can choose larger k, at the cost of more switch/router routing tables consumptions.

3) Controller Overhead: Fig. 10 shows that when $k = 6$, the controller overhead in terms of the failure radius. In all the radiuses, the controller only needs to handle about 40% of failures, which means the data plane has already handle more than 60% of the failures. As the failure radius grows bigger, the controller overhead has the trend to get heavier too. This is because when more links are destroyed, it is more difficult for the data plane to recover from the failure. Even when the failure radius is 150, about 60% are handled locally.

4) Maximal Load: Fig. 11 shows the maximal load of the splicing actions generation algorithm (Algorithm 2), compared to the shortest path splicing actions generation. The shortest path splicing actions generation choose the path with the
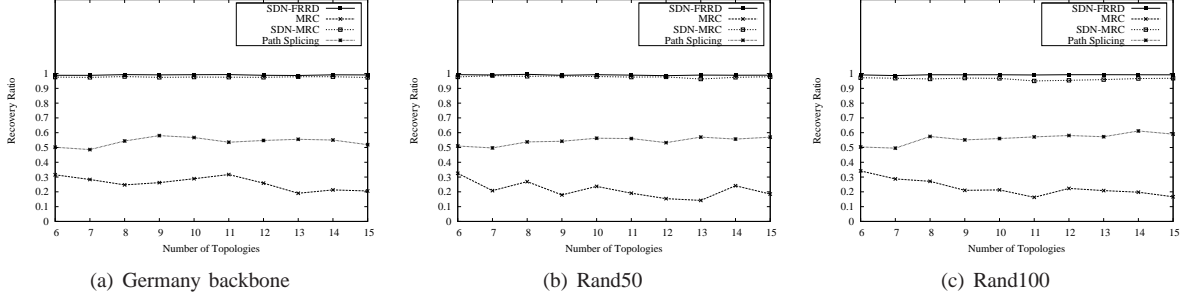
Fig. 6. Recovery Ratio vs. Number of backup topologies k when the failure radius=50
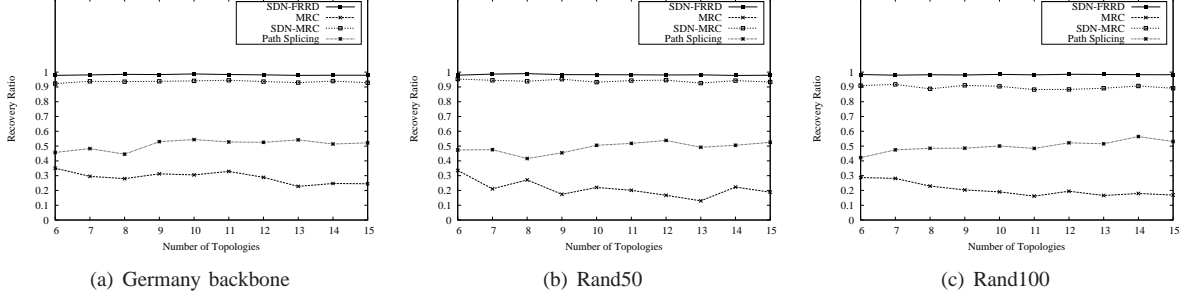

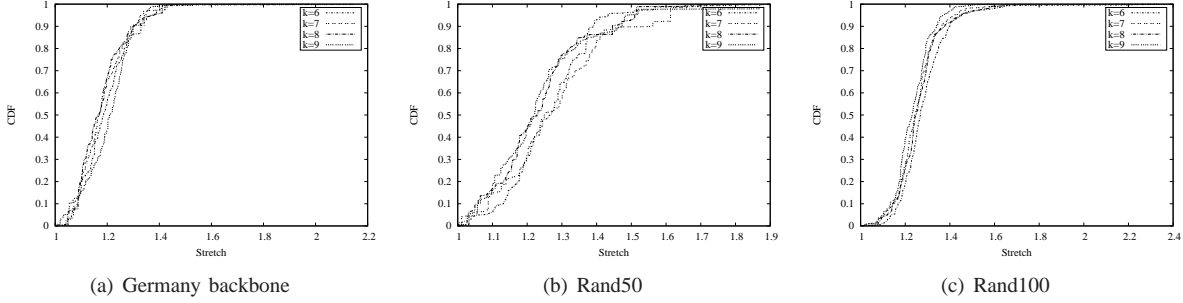Fig. 7. Recovery Ratio vs. Number of backup topologies k when the failure radius=100

Fig. 8. Path Stretch vs. Number of backup topologies k when the failure radius=50

minimal path length between a reconnection request (s, t) when multiple rerouting paths between them are available [36]. It however does not consider the load distribution among nodes. The results of the ML reduction are normalized based on the result of ILP1. From the graph, we can see that our algorithm can reduce the ML. As the failure radius becomes bigger, the ML also gets bigger, which indicates that without consider the load distribution, the load imbalance among node gets more severer.

5) Recovery Time: Fig. 12 shows the receive rate on the Iperf client. A region failure occurred between the Iperf server and client at 0.3s. Packets can not be handled locally by backup tables, thus are sent to the controller. The receive rate on the client did have a sharp reduction at 0.3s, but it recovered very fast after about 10ms. The recovery time in real scenarios differs, which depends largely on the round trip time between a switch and a controller.

V. RELATED WORK

There are limited number of recent papers focusing on leveraging SDN for large-scale regional failures. Nguyen *et*

al. [12] studied latency between a switch and a controller and confirmed the applicability of SDN on disaster-resilient WANs. Works in [26], [27] studied using SDN to meet carrier-grade requirements and pointed out that the reactive approach may not be able to achieve sub-50ms recovery. However, the above works did not consider the heavy controller overhead and the consistent packet processing problem [7], [8]. To reduce the recovery time, Sgambelluri *et al.* [37] proposed the proactive segment protection. Kamamura *et al.* [38] gave a prototype to achieve IP fast rerouting using backup tables via autonomous OpenFlow controllers. The proposed proactive recovery can significantly reduce the recovery time. But, in face of region failure scenarios, the performance may be significantly decreased since the flexibility of SDN's global view is not used.

VI. CONCLUSION

In this paper, we propose a SDN based architecture to enhance the reliability of network against disaster failures. We propose our algorithms for geographic-based backup topologies generation and splicing considering the laod distribution
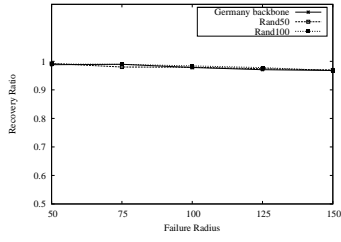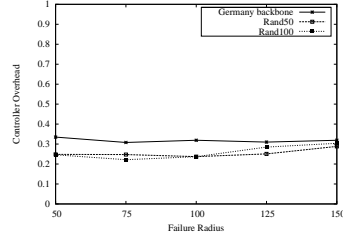
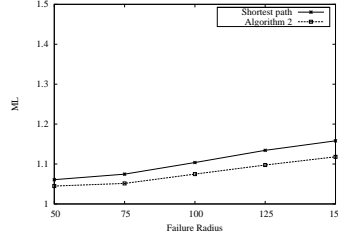Fig. 9. Recovery Ratio vs. Failure Radius

Fig. 10. Controller Overhead

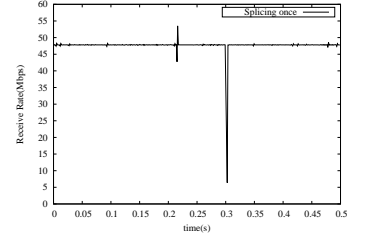Fig. 11. ML on the Germany backbone

Fig. 12. Receive Rate on the Iperf client

among nodes, and implement our approach by utilizing multiple tables pipeline processing and fast failover group tables of OpenFlow. Experiments show that, by well pre-designed backup topologies protection, our fast restoration approach can efficiently use the redundancy to achieve high reachability and low stretch with low controller overhead. The load distribution after a regional is more even, compared to the previous splicing algorithm in [36].

REFERENCES

[1] T. Sakano, Z. M. Fadlullah, T. Ngo, H. Nishiyama, M. Nakazawa, F. Adachi, N. Kato, A. Takahara, T. Kumagai, H. Kasahara *et al.*, "Disaster-resilient networking: a new vision based on movable and deployable resource units," *Network, IEEE*, vol. 27, no. 4, 2013.

[2] R. R. Kompella, J. Yates, A. Greenberg, and A. C. Snoeren, "Detection and localization of network black holes," in *Proceedings of INFOCOM*, 2007.

[3] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[4] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu *et al.*, "B4: Experience with a globally-deployed software defined wan," in *Proceedings of the ACM SIGCOMM*, 2013.

[5] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer, "Achieving high utilization with software-driven wan," in *Proceedings of ACM SIGCOMM*, 2013.

[6] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood, "On controller performance in software-defined networks," in *Proceedings of Hot-ICE*, 2012.

[7] N. P. Katta, J. Rexford, and D. Walker, "Incremental consistent updates," in *Proceedings of ACM SIGCOMM Hotnets*, 2013.

[8] P. Peresini, M. Kuzniar, N. Vasic, M. Canini, and D. Kostic, "Of. cpp: Consistent packet processing for openflow," Technical report, EPFL, Tech. Rep., 2013.

[9] M. Liotine, *Mission-critical network planning*. Artech House, 2003.

[10] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama *et al.*, "Onix: A distributed control platform for large-scale production networks." in *OSDI*, 2010.

[11] B. Heller, R. Sherwood, and N. McKeown, "The controller placement problem," in *Proceedings of HotSDN*, 2012.

[12] K. Nguyen, Q. T. Minh, and S. Yamada, "A software-defined networking approach for disaster-resilient wans," in *Proceedings of ICCCN*, 2013.

[13] Y. Y. Kagan, "Earthquake size distribution: Power-law with exponent 1 2 ?" *Tectonophysics*, vol. 490, pp. 103–114, 2010.

[14] A. Sen, S. Murthy, and S. Banerjee, "Region-based connectivity - a new paradigm for design of fault-tolerant networks," in *HPSR*, Paris, France, June 2009.

[15] S. Neumayer, G. Zussman, R. Cohen, and E. Modiano, "Assessing the impact of geographically correlated network failures," in *IEEE Military Communications Conference (MILCOM)*, Nov 2008.

[16] I. I. W. Group, "Inference of shared risk link groups," *Internet Draft*, Nov. 2001. [Online]. Available: http://tools.ietf.org/html/draft-many-inference-srlg-02

[17] J. Q. Hu, "Diverse routing in optical mesh networks," *IEEE Transactions on Communications*, vol. 51, no. 3, pp. 489–494, March 2003.

[18] P. Datta and A. K. Somani, "Diverse routing for shared risk resource groups (srrg) failures in wdm optical networks," in *BroadNet*. IEEE, 2004, pp. 120–129.

[19] L. Shen, S. Member, X. Yang, S. Member, and B. Ramamurthy, "Shared risk link group (SRLG)-diverse path provisioning under hybrid service level agreements in wavelength-routed optical mesh networks," in *IEEE/ACM Trans. Netw*, vol. 13, no. 4, 2005, pp. 918–931.

[20] B. Wu, P.-H. Ho, J. Tapolcai, and P. Babarczi, "Optimal allocation of monitoring trails for fast SRLG failure localization in all-optical networks," in *IEEE GLOBECOM 2010*, Dec. 2010.

[21] H. Lee, E. Modiano, and K. Lee, "Diverse routing in networks with probabilistic failures," *IEEE/ACM Transactions on Networking*, vol. 18, no. 6, pp. 1895–1907, Dec. 2010.

[22] P. Psenak, S. Mirtorabi, A. Roy, L. Nguyen, and P. Pillay-Esnault, "Multi-topology (MT) routing in ospf," RFC 4915, 2007.

[23] T. Przygienda, "M-ISIS: multi topology (MT) routing in intermediate system to intermediate systems (IS-ISs)," RFC 5120, 2008.

[24] A. Kvalbein, A. F. Hansen, T. Čičic, S. Gjessing, and O. Lysne, "Multiple routing configurations for fast IP network recovery," *IEEE/ACM Transactions on Networking (TON)*, vol. 17, no. 2, pp. 473–486, 2009.

[25] A. Kvalbein, A. F. Hansen, T. Cicic, S. Gjessing, and O. Lysne, "Fast IP network recovery using multiple routing configurations," in *Proceedings of INFOCOM*, 2006.

[26] D. Staessens, S. Sharma, D. Colle, M. Pickavet, and P. Demeester, "Software defined networking: Meeting carrier grade requirements," in *Proceedings of LANMAN*, 2011.

[27] S. Sharma, D. Staessens, D. Colle, M. Pickavet, and P. Demeester, "Openflow: meeting carrier-grade recovery requirements," *Computer Communications*, 2012.

[28] S. Trajanovski, F. A. Kuipers, P. V. Mieghem, A. Ilic, and J. Crowcroft, "Critical regions and region-disjoint paths in a network," in *IFIP Networking*, May, 22-24 2013.

[29] O. S. Specification, "Version 1.3.3," *Open Networking Foundation*, 2012.

[30] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, "Nox: towards an operating system for networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 3, pp. 105–110, 2008.

[31] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: rapid prototyping for software-defined networks," in *Hotnets*, 2010.

[32] A. Tirumala, F. Qin, J. Dugan, J. Ferguson, and K. Gibbs, "Iperf: The TCP/UDP bandwidth measurement tool," *http://dast. nlanr. net/Projects*, 2005.

[33] M. Motiwala, M. Elmore, N. Feamster, and S. Vempala, "Path splicing," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 4, pp. 27–38, 2008.

[34] X. Wang, X. Jiang, C.-T. Nguyen, X. Zhang, and S. Lu, "Fast connection recovery against region failures with landmark-based source routing," in *Proceedings of DRCN*, 2013.

[35] X. Wang, X. Jiang, and et.al., "Fast connection recovery from multiple failures with landmark-based source routing," in *DRCN*, 2013.

[36] A. Xie, X. Wang, W. Wang, and S. Lu, "Designing a disaster-resilient network with software defined networking," in *Quality of Service (IWQoS)*. IEEE, 2014.

[37] A. Sgambelluri, A. Giorgetti, F. Cugini, F. Paolucci, and P. Castoldi, "Openflow-based segment protection in ethernet networks," *Optical Communications and Networking, IEEE/OSA Journal of*, vol. 5, no. 9, pp. 1066–1075, 2013.

[38] S. Kamamura, D. Shimazaki, A. Hiramatsu, and H. Nakazato, "Autonomous ip fast rerouting with compressed backup flow entries using openflow," *IEICE TRANSACTIONS on Information and Systems*, vol. 96, no. 2, pp. 184–192, 2013.