Independent Transient Plane Design for Protection in OpenFlow-Based Networks

Nattapong Kitsuwan, Séamas McGettrick, Frank Slyne, David B. Payne, and Marco Ruffini

***Abstract*—Network protection against link failure is required to allow packets to reach the destination with minimal packet loss when a failure occurs. When a link fails, traffic that attempts to use the failed link is interrupted. Typically, routers in the network discover the failure and find a new route to bypass the failed link. Alternatively, well-known segment protection schemes can also be used to speed up the link recovery time by rerouting packets locally through precalculated protection paths. However, several backup paths have to be prepared for each primary path, making path configuration rather complex and poorly scalable. This paper proposes a design for fast rerouting in an OpenFlow-based network. This new design reduces the number of flow entries and the number of configuration messages needed for network rerouting, which in turn reduces the memory size needed in each switch and the CPU load at the controller. We show empirically and using simulations that our design can reduce the number of flow entries and configuration messages needed by about 60% and 75%, respectively, when compared with an existing OpenFlow-based segment protection design. Furthermore, we implement the proposed design on a pan-European network and show that our design can recover from a link failure in as little as 25 ms.**

***Index Terms*—Mininet; Network protection; OpenFlow; Routing; Software-defined network.**

I. Introduction

Failure protection is an important issue in network survivability. When connectivity is lost due to a failure, packets that attempt to pass the failed link cannot be delivered to their destination. An open shortest path first (OSPF) network, in which a link-state-based routing protocol is used to learn network topology, can detect the link failure and converge to a new topology. However, recovery time in the OSPF network takes more than a second [1]. Recovery times of this order are not acceptable in many networks, as target protection times of 50 ms or 100 ms are common, respectively, for leased line traffic and video/audio services [2]. Multiprotocol label switched (MPLS) networks provide fast rerouting using an alternative label switched path (LSP) to reroute packets from a node connected to the failed link to another node or to the destination. In a typical MPLS network, operating a distributed control plane [3,4], forwarding decisions are made locally by each switch, based on its predefined configurations. Although this allows for dynamic and automatic forwarding decisions to be made in the local switch, there is no guarantee that the chosen route is still optimal when the failure occurs.

Segment protection approaches rely on preplanned backup paths that deflect packets from a primary path to other paths [5–7], as shown in Fig. 1. At each switch, primary and backup ports are defined. Packets travel through the primary path via the primary port of each switch to the destination during normal operations. If the switch detects that the primary port is not available, the packets are deflected from the primary path to a neighboring switch via the backup port of that switch. The neighboring switch then forwards the packets to the destination via its primary port.

Software-defined networking (SDN) [8,9] is an attractive solution that enables a network administrator to control the network requirements, including how to deal with failure protection [10,11]. SDN separates the network control and data forwarding planes. This allows network engineers and administrators to respond quickly to changing requirements of the network from a centralized controller. The network administrator can avail of improved network programmability to flexibly control the physical switches from the controller, which runs all the intelligent control and management software, regardless of the vendor or the model of the switches used.

OpenFlow [12,13] is a widely known protocol (southbound interface) for SDN networks that enables the controller to interact with the forwarding plane of the switches and thus make adjustments to the network. The hardware switches can also use OpenFlow messages to inform the controller when links go down or when a packet arrives with no specified forwarding instructions. The forwarding instructions are based on a flow entry, which is defined by a set of specific parameters, such as the source and destination Ethernet/IP addresses, the switch input port and VLAN tag, etc. [13]. The controller specifies the set of parameters and how packets that match the flow entry should be processed. Packets are matched against flow entries based on prioritization. Higher priority flow entries are used, when available, over lower priority ones.

Manuscript received October 9, 2014; revised February 4, 2015; accepted February 4, 2015; published March 23, 2015 (Doc. ID 224583).

The authors are with CTVR, Trinity College Dublin, Dublin 2, Ireland (e-mail: kitsuwan@tcd.ie).

http://dx.doi.org/10.1364/JOCN.7.000264

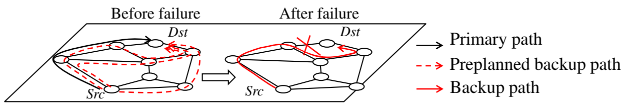


Fig. 1. Segment protection design.

In [14] Sgambelluri *et al.* implemented a segment protection scheme in an OpenFlow-based network. In order to implement the segment protection scheme in OpenFlow, a low priority value of flow entry is set for the backup path, and a high priority value is set for the primary path in each switch. A fast switchover time is guaranteed by the fact that path recovery is performed locally by the switch connected to the failed link. An additional mechanism is required in every switch to remove flow entries from the primary path that relate to the failed link. The switches then send their port status to the controller to update the topology.

This design has some disadvantages. First, the primary and backup paths are correlated, as the latter is reassigned every time the primary path is changed. In addition, several backup paths have to be prepared for the primary path. The number of backup paths ($N_{\rm BP}$) for each primary path depends on the number of intermediate switches (N_S), where $N_{\rm BP} = N_S + 1$. This makes the provisioning of backup paths more complex if there are several intermediate nodes between the source and destination pair. This will be explained in more detail using an example in Section II. In a large network, a large number of flow entries for source–destination pairs have to be stored in each switch, which can lead to larger loads on the forwarding tables of the switch [thus requiring larger amounts of ternary content-addressable memories (TCAMs)]. TCAM is a type of high-speed memory capable of searching its entire content in one clock cycle. This makes them very fast and highly appropriate for use in switching tables. However, their complexity also makes them very expensive to produce. Second, this segment protection design requires the controller to issue a large number of flow commands when the data path is changed, since backup ports on each switch have to be recalculated. This leads to an increased load on the CPU and in turn to increased power consumption in the controller.

In this paper we propose a failure protection design, originally introduced in [15,16], which addresses many of the disadvantages of segment protection. The proposed design reduces the forwarding table size by reducing the number of flow entries, compared to the segment protection design reported in [14], while maintaining fast protection times. In the proposed design, two uncorrelated, working, and transient planes are adopted for the data forwarding plane. A working path on the working plane controls routing between source and destination when no failure occurs. The transient plane, which is permanently stored in each switch, provides routing for any case of failure to take care of on-the-fly packets that are already on route to the destination at the time of failure. Packets are then routed through a backup path on the working plane after the configuration is done. Results from the numerical analysis and simulations show a reduction in the number of flow entries of about 60% and in the number of configuration messages of about 75%. Furthermore, we have implemented this design on a pan-European OpenFlow network achieving protection times of less than 25 ms.

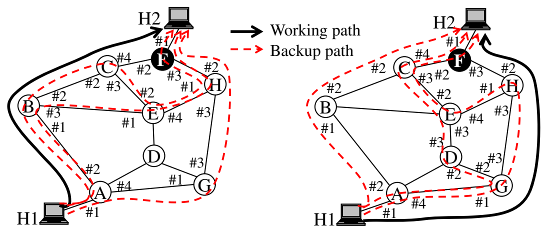
II. OpenFlow-Based Segment Protection

Segment protection schemes are designed to provide bypass backup paths for any possible link failure between adjacent nodes in advance of any failure occurring. This has the advantage that segment protection designs have greatly reduced switchover times compared to other protection schemes since the predefined backup path is already available when a failure occurs.

In a segment protection design all switches on the network should contain a working and backup data port for all links. The working port is used during normal operation, and the backup port is only used in the event of a link failure. An interesting OpenFlow-based segment protection scheme was developed in [14], which maintains two flows at different priority levels for each working and backup path needed in the network. The priority levels ensure that the working path is used by default and that the backup path is only used in the event of a link failure on the primary port. When a link failure occurs, an *auto-reject* mechanism on switches connected to the failed link removes the corresponding flow entries. This leaves the lower priority backup port flow, provided that it too has not been affected by the failure, to divert the traffic to the backup path. The switches then send their port status to the controller to update the network topology. Thus, the backup flow entries guide packets from the switch connected to the failed link to a destination via a predefined backup path that corresponds to the failure.

It should be noted that an OpenFlow switch forwards a packet based on a match field that can contain a large number of parameters, including fields in headers at layers 2, 3, and 4 of the OSI stack, in addition to physical port matching. While a detailed match field (e.g., considering IP addresses and TCP ports) is useful for certain applications, segment protection makes use of port matching, as this allows aggregating a large number of flows into one table entry, while using only port-matching information to bypass the failed link. Thus OpenFlow allows applying a "wildcard" to files in the header of layer 2, 3, and 4 protocols.

Figure 2 shows an example of the segment protection design described in [14]. The working path between $H1$ and $H2$ in Fig. 2(a) is A, B, C, F. Three backup paths are provided for each link failure along the working path. A backup path BP1 is provided for link failure between A and B. A backup path BP2 is provided for link failure between B and C. A backup path BP3 is provided for link failure between C and F. If the working path is changed to A, G, H, F, as shown in Fig. 2(b), the backup paths have to be recalculated. A backup path BP4 is provided for link failure between A and G. A backup path BP5 is provided for link failure between G and H. A backup path BP6 is provided for link failure between H and F. The controller reconfigures flows by removing 13 flows and adding 14 flows overall



Backup paths
BP1:A,G,H,F for link (A,B) failure
BP2:A,B,E,H,F for link (B,C) failure
BP3:A,B,C,E,H,F for link (C,F) failure

A		
Inport	action	priority
#1	#2	High
#1	#4	Low

B		
Inport	action	priority
#1	#2	High
#1	#3	Low

C		
Inport	action	priority
#2	#4	High
#2	#3	Low

D		
Inport	action	priority

E		
Inport	action	priority
#1	#4	Low
#2	#4	Low

F		
Inport	action	priority
#2	#1	High
#3	#1	Low

G		
Inport	action	priority
#1	#3	Low

H		
Inport	action	priority
#1	#2	Low
#3	#2	Low

(a)

Backup paths
BP4:A,B,C,F for link (A,G) failure
BP5:A,G,D,E,C,F for link (G,H) failure
BP6:A,G,H,E,F for link (H,F) failure

A		
Inport	action	priority
~~#1~~	~~#2~~	~~High~~
~~#1~~	~~#4~~	~~Low~~
#1	**#4**	**High**
#1	**#2**	**Low**

B		
Inport	action	priority
~~#1~~	~~#2~~	~~High~~
#1	**#2**	**Low**
~~#1~~	~~#3~~	~~Low~~

C		
Inport	action	Priority
~~#2~~	~~#4~~	~~High~~
#2	**#4**	**Low**
#3	**#4**	**Low**
~~#2~~	~~#3~~	~~Low~~

D		
Inport	action	priority
#2	**#3**	**Low**

E		
Inport	action	priority
#3	**#2**	**Low**
#4	**#2**	**Low**
~~#1~~	~~#4~~	~~Low~~
~~#2~~	~~#4~~	~~Low~~

F		
Inport	action	priority
~~#2~~	~~#1~~	~~High~~
~~#3~~	~~#1~~	~~Low~~
#3	**#1**	**High**
#2	**#1**	**Low**

G		
Inport	action	priority
~~#1~~	~~#3~~	~~Low~~
#1	**#3**	**High**
#1	**#2**	**Low**

H		
Inport	action	priority
~~#1~~	~~#2~~	~~Low~~
~~#3~~	~~#2~~	~~Low~~
#3	**#2**	**High**
#3	**#1**	**Low**

(b)

Fig. 2. Example of the configuration for each switch in the segment protection design. (a) Working path is A,B,C,F; (b) working path is A,G,H,F.

in the switches, bringing the total number of required flow commands to 27.

III. Proposed Independent Transient Plane Design

The proposed independent transient plane (ITP) design provides segment protection together with ease of path configuration. It consists of working and transient planes, which are used at different stages of the failure recovery action, as shown in Fig. 3.

The working plane is used as a data path to route packets from source to destination and contains information about both the primary paths and disjointed backup paths, one backup path for each primary path. The primary path is used for normal network operations, while the backup path replaces the corresponding primary path when it is brought down by a failure.

The transient plane, where the routing is statically predefined to support all failure cases, is only temporarily used while the backup path is being configured. A shortest path is considered as a routing policy. Without the transient plane, on-the-fly packets have to wait for forwarding instruction at the switch connected to the failed link until exceeding a *hard timeout* timer. The transient plane is only temporarily used to route packets that are already en route to their destination. As soon as the controller is informed of the failure the backup path is installed in the switches so that all new packets entering the network will follow the new backup route.

The operation of the ITP design is as follows. During normal operation, packets are forwarded to the destination using configurations of the primary path on the working plane, shown in Fig. 3(a). When a link failure occurs, switches connected to the failed link detect the failure and send a port status to the controller. The controller adds a flow entry to the switch as a means to move packets to the transient plane, as shown in Fig. 3(b). Such a connection deflects incoming packets that attempt to pass the failed link to the transient plane via a backup port of the local switches. The flow entry of that connection replaces a tag (either a VLAN or a MPLS can be used) in the packet header with one representing a new link ID. The flow entry of the connection to the transient plane also identifies the backup output port of the switch. Figure 4 shows an example: suppose the link between switch-B (SW_B) and SW_C fails. The tag of packets with the destination SW_C is replaced with an ID (say 101), and the packets are sent to backup port #2 at SW_B. After the packet arrives at SW_A, it is forwarded to port #2 via the transient plane. It should be noted that the link ID is used by the transient plane to avoid sending packets to the failed link.

As described above the ITP uses OpenFlow as is, without any additional extensions. Therefore, the flow message from the controller to move packets from the working plane to the transient plane is required. However, an extension mechanism such as *auto-rejection* in [14] can be used to add

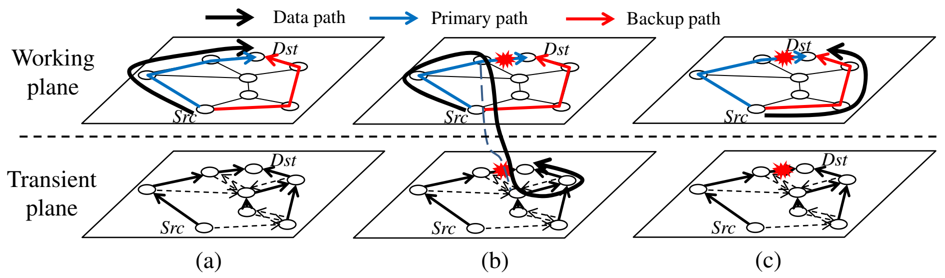
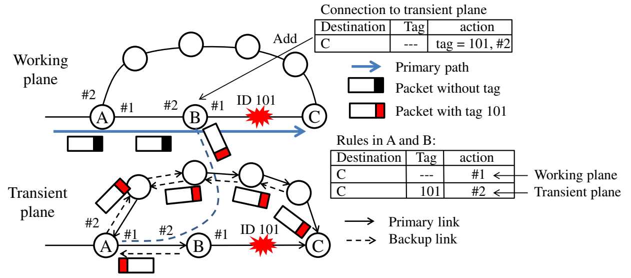


Fig. 3. Actions when link failure occurs in ITP design. (a) Packets travel on the primary path before failure; (b) controller adds a connection to the transient plane to switch, and packets travel via the transient plane before configuration of the backup path is done; (c) packets travel on the backup path after the configuration is done.



Connection to transient plane

Destination	Tag	action
C	---	tag = 101, #2

Rules in A and B:

Destination	Tag	action
C	---	#1
C	101	#2

Fig. 4. Example of using a tag.

the flow, instead of the controller, before reporting the port status to the controller.

The transient plane guides on-the-fly packets to their destination. The switches then generate an OFPPR_DELETE port status message and send it to the controller to update the topology. After that, the controller changes the data path from the primary path to a predefined backup path, by sending OFPFC_ADD messages to set up the backup path and OFPFC_DELETE_STRICT messages to remove the primary path.

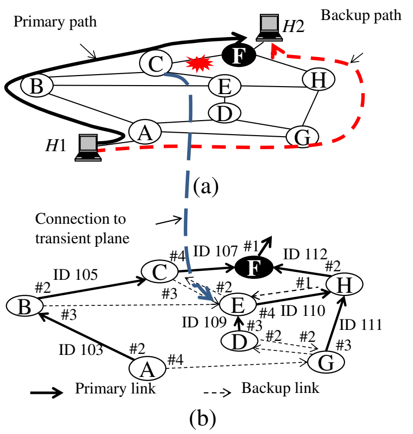
The ITP addresses the three main disadvantages of segment protection mentioned above. First, The ITP backup path is completely independent of the working path. A single backup path exists for every working path. In segment protection, multiple backup paths were needed depending on where the failure occurred. The transient plane replaces the need for multiple backup paths. This means that changes can be made to the working path without making changes to the backup path. Second, the transient plane can be designed to avoid congestion on the network. Since a single transient plane routing is used for any failure it is easier to avoid congestion than it is with single flow backup paths used in segment protection. Finally, thanks to the fact that a single transient plane is used for all failures there is a decrease in the number of flows in each switch in a large network with multiple source and destination pairs. In addition, the number of controller messages needed to recover the network from failure also decreases. We will quantify this in the results section.

IV. Configuration for the Transient Plane

There are three configurations needed in each switch: the working plane, the transient plane, and the connection between those two planes. The configuration for the working plane is needed for the switches along the primary path when no failure has occurred and along the backup path when a failure has occurred. The configuration for the transient plane is needed for every switch in the network. The configuration for the connection between those two planes is needed only in the switches connected to the failed link. All configurations are categorized based on OpenFlow prioritization. The configurations are as follows:

1. *Configuration on the working plane*: Source and destination IP addresses and priority value W are considered as matching parameters for a flow entry. Figure 5(c) shows flow entries in each switch of the primary path (highlighted in black), which are A, B, C, and F, as preassigned in Fig. 5(a). After a link failure occurs between C and F, the primary path is removed and the preassigned backup path, as shown in Fig. 5(a), is configured by the controller.
2. *Configuration on the transient plane*: The transient plane is based on a tree topology, which we refer to as the transient tree. The transient tree is used to guide packets from any switch in the network to a specific destination switch, called the root, when a failure occurs. The transient tree is actually created by overlapping a number of simpler trees, which we refer to as the common tree and multiple link-failure trees.

   The common tree represents the network with no failures and uses the shortest path from each switch in the network to the root destination node. As shown in Fig. 6, switch F is assumed to be the root. A common tree for this topology is shown in Fig. 6(b). Each link on the common tree is called a primary link.



H1	H2		Mod tag =107, #3	B

Connection to transient plane

▪ Working plane
□ Transient plane

A				
src	dst	Tag	action	priority
	H2		#2	Tl
	H2	103	#4	Th
H1	**H2**		**#2**	**W**

B				
src	dst	Tag	action	priority
	H2		#2	Tl
	H2	105	#3	Th
H1	**H2**		**#2**	**W**

C				
src	dst	Tag	action	priority
	H2		#4	Tl
	H2	107	#3	Th
H1	**H2**		**#4**	**W**

D				
src	dst	Tag	action	priority
	H2		#3	Tl
	H2	109	#2	Th

E				
src	dst	Tag	action	priority
	H2		#4	Tl
	H2	110	#2	Th
	H2	112	#2	Th

F				
src	dst	Tag	action	priority
H1	**H2**		**Strip tag, #1**	**W**

G				
src	dst	Tag	action	priority
	H2		#3	Tl
	H2	111	#2	Th

H				
src	dst	Tag	action	priority
	H2		#2	Tl
	H2	112	#1	Th

(c)

Fig. 5. Example of configuration for each switch in the ITP design. (a) Routing on working planes; (b) routing on transient planes; (c) flow tables in each switch.

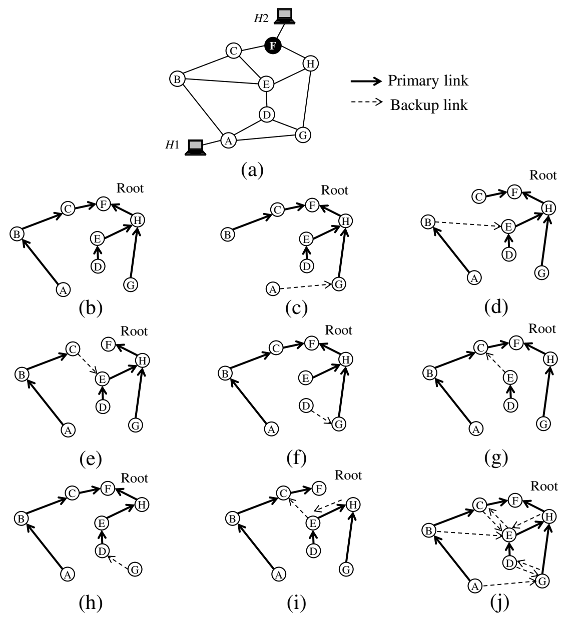


Fig. 6. Creating the transient tree for root F. (a) Network topology; (b) common tree; (c) link-failure tree for (A,B); (d) link-failure tree for (B,C); (e) link-failure tree for (C,F); (f) link-failure tree for (D,E); (g) link-failure tree for (E,H); (h) link-failure tree for (G,H); (i) link-failure tree for (H,F); (j) transient tree for root F.

A link-failure tree takes possible link failures into account and must be calculated for every primary link. The link-failure tree is calculated from the common tree by removing a primary link and finding a shortest path from a switch connected to the failed link to the root. The calculation is repeated until every primary link is considered as the failure. For example, let (A, B) be a link between switches A and B. It should be noted that if the link (A, B) fails, packets at the switch A cannot be received from or sent to switch B via (A, B). As shown in Fig. 6(c), when link (A, B) fails, switch A deflects packets to switch G, and primary links from switch G to the root are used as routing. In Fig. 6(d), when link (B, C) fails, switch B deflects packets to switch E, and primary links from switch E to the root are used as routing. The transient tree, Fig. 6(j), merges the common tree and all the link-failure trees so that routing is available for every possible failure in the network. The process of creating transient trees is repeated until every switch is considered as the root.

In the transient plane, destination IP addresses, a tag, and priority are used as matching parameters for each flow entry. Two priorities, T_h and T_l, are used for the transient tree, where $T_h > W > T_l$. T_h reroutes the packet to avoid the failed link via the backup link. Otherwise, T_l guides packets to the root via primary links. Figure 5(c) shows the configuration of the transient plane (highlighted with white) in each switch.

3. *Configuration of the connection to the transient plane*: This connection is inserted by the controller to a local switch when the controller receives port status after the failure from the switch to modify a tag and to deflect the packets to the transient plane via a backup output port. Source and destination IP addresses and a priority B, where $B > T_h > W > T_l$, are considered as patching parameters for the flow entry of the connection. Figure 5(c) shows the flow entry of the connection when link (C, F) fails.

V. Number of Flow Entries Analysis

Since the objective of this work is to reduce the number of flow entries required for protection and the number of configuration messages when link failure occurs, we report a comparison of those numbers in both the segment protection and the ITP designs. We based our analysis on a $n \times n$ grid topology due to its suitability to carry out analytical studies.

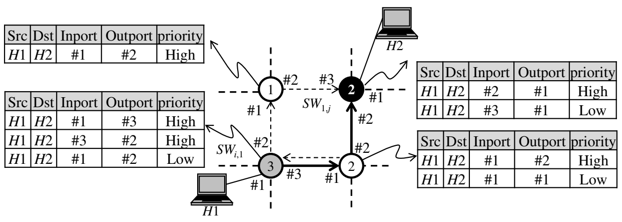
A. Analysis of Segment Protection Design

The total number of flow entries needed in the network is the sum of the flow entries needed by every source–destination pair. Some source–destination pairs will only affect the flow tables of a subsection of the grid topology, and so it can be useful to break the grid into submatrices for the calculation.

For a generalized case, let us assume an $n \times n$ matrix defining the network topology, and a subset of this topology identified by the submatrix $i \times j$. Each cell of the matrix identifies a network node where a switch SW is located. Let $\mathrm{SW}_{i,j}$ be an index for a switch in the submatrix, where $i, j \leq n$ and $j \leq i$. Figure 7 shows an example of the configuration of each switch for sending packets from $H1$ to $H2$. $H1$ connects to a source switch $\mathrm{SW}_{i,1}$, which is shown in gray. $H2$ connects to a destination switch $\mathrm{SW}_{1,j}$, which is shown in black. The number in each switch represents the number of flow entries needed for source $H1$ and destination $H2$.

Let $F_{i,j}$ be the number of flow entries, including primary and backup paths, needed in the network to route packets from source $\mathrm{SW}_{i,1}$ to destination $\mathrm{SW}_{1,j}$. It is assumed that the primary path travels along the border of the submatrix, and the backup path is designed as in Fig. 8. Figure 8 shows examples of $F_{i,j}$ when i is 2, 3, and 4 and j is 2, 3, and 4. Let $P_{i,j}$ be the number of source–destination pairs on submatrix $i \times j$. $F_{i,j}$ and $P_{i,j}$ are expressed by

$$F_{i,j} = \begin{cases} 6 + 4(i-2); & j = 1 \\ 8 + 4(i-2); & j = 2 \\ 16 + 4(i-3) + 4(j-3); & j \geq 3 \end{cases}, \qquad (1)$$



Src	Dst	Inport	Outport	priority
$H1$	$H2$	#1	#2	High

Src	Dst	Inport	Outport	priority
$H1$	$H2$	#1	#3	High
$H1$	$H2$	#3	#2	High
$H1$	$H2$	#1	#2	Low

Src	Dst	Inport	Outport	priority
$H1$	$H2$	#2	#1	High
$H1$	$H2$	#3	#1	Low

Src	Dst	Inport	Outport	priority
$H1$	$H2$	#1	#2	High
$H1$	$H2$	#1	#1	Low

Fig. 7. Flow entries in each switch for segment protection.

$$P(i,j) = \begin{cases} (n-i+1) \times (n-j+1) \times 4; & j = 1 \text{ or } i = j \\ (n-i+1) \times (n-j+1) \times 8; & \text{otherwise} \end{cases}. \quad (2)$$

The constants 6 and 8 in Eq. (1) represent the number of flow entries in the switches at the edge when $j = 1$ and $j = 2$, respectively. For example, in the 2×2 submatrix in Fig. 8 the sum of the flows in the switches is 6 where $j = 1$ and 8 where $j = 2$. The variable term given by $4(i-2)$ in Eq. (1) represents the number of extra flow entries needed for each new row you add to the submatrix; i.e., for each new row added to the submatrix, four new flow entries must be written to switches in the network. When $j \geq 3$, the number of flows at edge and corner switches is 16. This time two variable terms are added to the equation $4(i-3)$ and $4(j-3)$. These represent the number of flow entries at intermediate switches on horizontal and vertical planes, respectively.

In Eq. (2), $(n-i+1)$ and $(n-j+1)$ represent the total number of pairs from (1,1) to (i,j) in submatrix $n \times j$ and $i \times n$, respectively. The constant multiplier is caused by the fact that each submatrix can be mirrored and transposed. In addition, bidirectional communication is required so each destination is also a source and *vice versa*, which gives $2 \times 2 \times 2 = 8$; see Fig. 9, where $P_{2,1}$. In the case of $i = j$, we only consider mirroring and transposing of submatrices, since link bidirectionality can be represented as a matrix transposition. Therefore, $\times 4$ is used in cases of $j = 1$ or $i = j$; see Fig. 9, where $P_{2,2}$.

The total number of flow entries in the network with matrix $n \times n$, $T_{\text{SEG}}(n)$, is obtained by

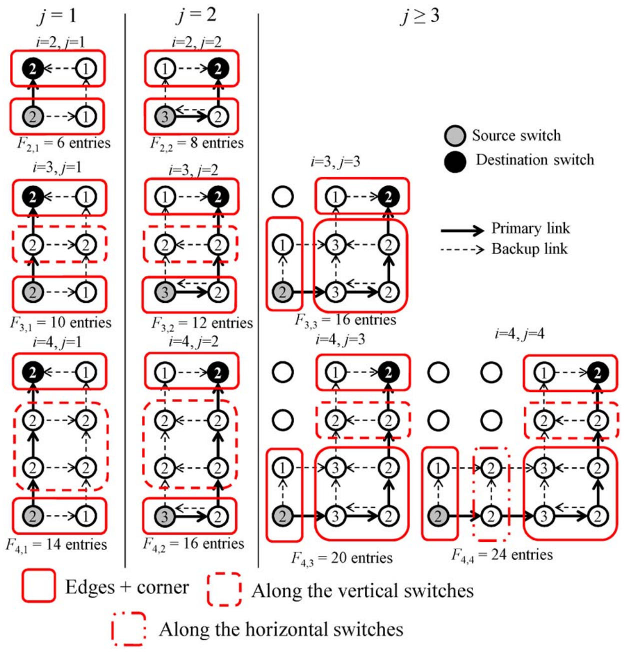


Fig. 8. Examples of $F_{i,j}$ in segment protection. Numbers in the switches represent the number of flow entries stored at that location.

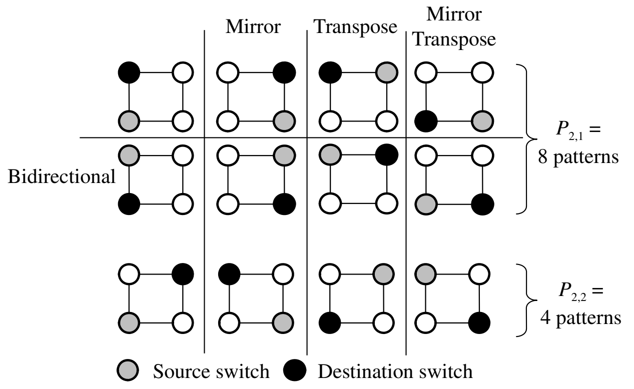


Fig. 9. Examples of $P_{i,j}$, when $i = 2$.

$$T_{\text{SEG}}(n) = \sum_{i=2}^{n} \sum_{j=1}^{i} (F_{i,j} \times P_{i,j}). \quad (3)$$

To clarify the analysis, the matrix $n = 2$ is used as an example. From Fig. 9 we know the submatrix 2×1 has eight patterns. Each pattern has six flow entries, as calculated in Fig. 8, where $j = 1$. Therefore the 2×1 submatrix has 48 flow entries. In the submatrix 2×2, there are four patterns each with eight flow entries giving a total of 32 flow entries. Thus the total number of required flow entries in the matrix where $n = 2$ is 80.

B. Analysis of ITP Design

The common and link-failure trees for the transient tree in the transient plane are shown in Fig. 10. The total number of flow entries in the ITP design in the network with matrix $n \times n$, $T_{\text{ITP}}(n)$, is obtained by

$$T_{\text{ITP}}(n) = T_T(n) + T_W(n), \quad (4)$$

where $T_T(n)$ and $T_W(n)$ represent the total number of flow entries for the transient plane and the working plane in the matrix $n \times n$, respectively. $T_T(n)$ and $T_W(n)$ are defined as follows:

$$T_T(n) = n^2 \times [1 + 2n(n-1) + 3(n-1)]. \quad (5)$$

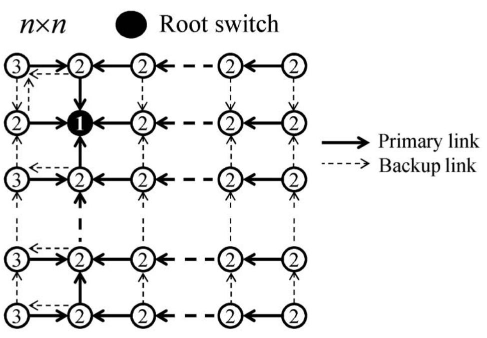


Fig. 10. Number of flow entries in each switch for transient plane in ITP.

For example, Fig. 10 shows the number of flow entries needed by each of the switches when switch (2,2) is considered as the root. From Fig. 10 we see that the root node itself needs only one flow entry, $(n-1)$ switches require three flow entries, and all other switches, i.e., $n(n-1)$, require two flow entries each. This process is then repeated, making each switch, in turn, the root node of the network: this is accounted for by the n^2 multiplier in Eq. (5).

$$T_W(n) = \sum_{i=2}^{n} \sum_{j=1}^{i} (E(i,j) \times P(i,j)), \tag{6}$$

where $P_{i,j}$ is obtained from Eq. (2), and $E_{i,j}$ represents the number of flow entries needed for the working plane to make a connection from $\mathrm{SW}_{i,1}$ to $\mathrm{SW}_{1,j}$ in submatrix $i \times j$, which is defined by

$$E_{i,j} = i + j - 1. \tag{7}$$

VI. Performance Analysis

In this section we compare the performance of our ITP design with that of the segment protection design presented in [14]. The comparison is carried out over a grid network topology with $2 \leq n \leq 8$.

First, the performance of both designs is evaluated in terms of total number of flow entries in the network, which is counted after the configurations of the primary path and the transient tree are completed. Figure 11 shows that for all grid topologies tested the ITP design requires fewer flow entries than the segment protection scheme. From Fig. 11 we can see that ITP requires 25% fewer flow entries than segment protection in a 2×2 grid. The benefits of ITP are even greater in larger networks with a 60% reduction in flow entries needed in the network when ITP is used instead of segment protection in an 8×8 grid. Figure 12 shows the percentage of flow entries reduction of the ITP design compared to that of segment protection, which is above 50% for topologies counting as few as 30 switches. This shows that our ITP mechanism can significantly reduce the occupation of flow table resources in the switch for most practical scenarios.

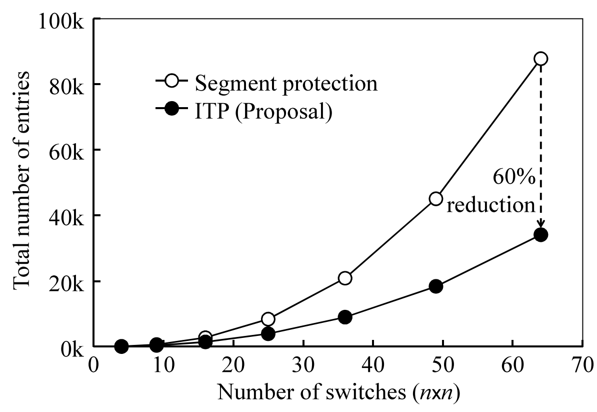


Fig. 11. Total number of flow entries in the network.

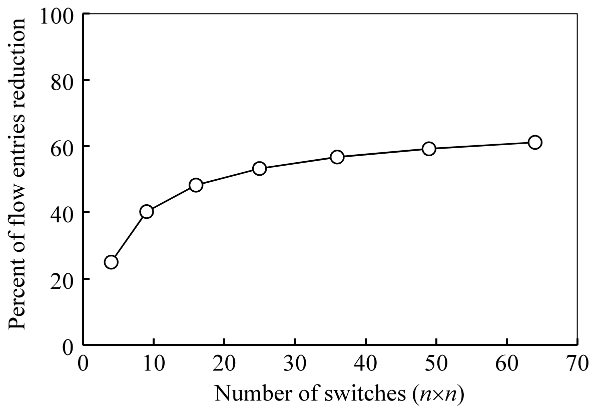


Fig. 12. Percentage of flow entries' reduction.

Finally, the number of configuration messages is evaluated when the primary path is changed to the backup path. Figure 13 shows that the number of configuration messages in the ITP design is 75% smaller than that of the segment protection design. For example, when $n \times n = 64$, only 28 configuration messages are needed in the ITP design, while the segment protection design requires 112 configuration messages. The reduction of configuration messages required directly translates into lower computational resources consumed.

The analysis of the number of flow entries considers the case in which each OpenFlow switch uses TCAMs to store all of the flow entries. However, some switch vendors might implement different memory architectures. For example, some strategies are implemented for which only active flow entries lie in TCAM and other inactive flow entries lie in a non-TCAM memory. In this case, the number of active flow entries in both segment protection and the ITP design are the same, since the working paths of both designs are the same, and only one flow entry is needed in each switch for each active flow. Therefore, the only improvement brought by ITP on switching table size would affect mainly non-TCAM memory. However, ITP still has an advantage in reducing the message processing overhead. The analysis of the number of flows for non-TCAM in this case is almost

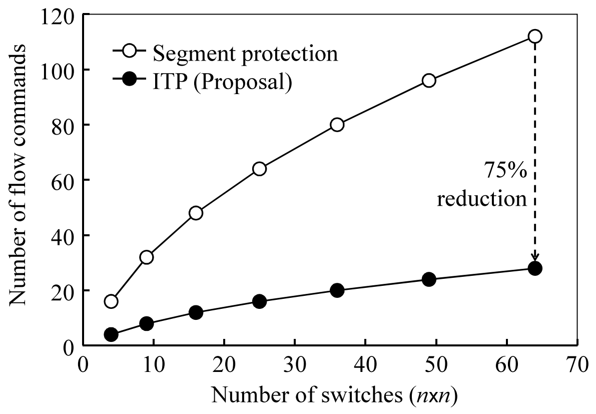


Fig. 13. Number of configuration messages when primary path is changed.

the same as the case in which TCAM is used to store all flows.

Moreover, we evaluate the performance of the ITP design with nonsymmetrical topologies in Fig. 14, which consists of 7 and 14 nodes, and considering every possible source–destination pair. We evaluate the number of flow entries in both the segment protection and the ITP design by simulation. A shortest path policy is used as routing for each source and destination pair. With the topology in Fig. 14(a), the total number of flow entries needed in the network is 447 flows in the segment protection. In the ITP design, the number of flow entries for the transient plane and the working plane is 124 and 149 flows, respectively. The total number of flow entries in the ITP design is 273 flows, which is 39% reduced compared to the segment protection. With the topology in Fig. 14(b), the total number of flow entries needed in the network is 2271 flows in the segment protection. In the ITP design, the number of flow entries for the transient plane and the working plane is 509 and 593 flows, respectively. The total number of flow entries in the ITP design is 1102 flows, which is 51.5% reduced compared to the segment protection. This shows that the advantages of ITP still hold for irregular, more realistic, topologies.

VII. Mininet Emulation Results

In the previous section we have shown that the ITP design requires fewer flow entries and configuration messages than segment protection designs. In this section we use a Mininet emulator [17] to confirm functionality of the ITP design. In this implementation, the working plane, ports, and transient plane information are stored in the controller database in three tables: *Paths*, *Port_map*, and *Transient*, respectively, as shown in Fig. 15. The *Paths table* stores the primary and backup paths for source and destination IP address pairs by identifying switch IDs along the paths from the source to the destination. The *Port_map table* keeps a record of backup ports and link ID. The *Transient table* stores routing information for all the switches to each destination including alternate routes to avoid failed links. In this example we implement the network shown in Fig. 5. The IP addresses of $H1$ and $H2$ are set to 10.0.0.1 and 10.0.0.2, respectively. Priorities used in the experiment are $B = 20{,}000$, $T_h = 15{,}000$, $W = 10{,}000$, and $T_l = 1000$. POX is used as a controller.

Our protection scenario operates as follows. UDP packets are sent from $H1$ to $H2$. A link failure occurs between switches C and F. When the switches detect the link

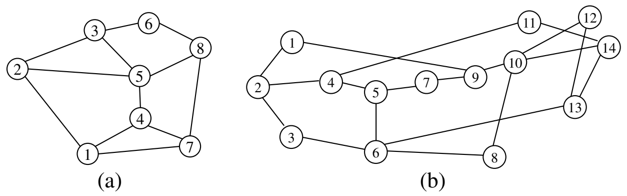


Fig. 14. Network topologies. (a) Sample topology; (b) NSF network topology.

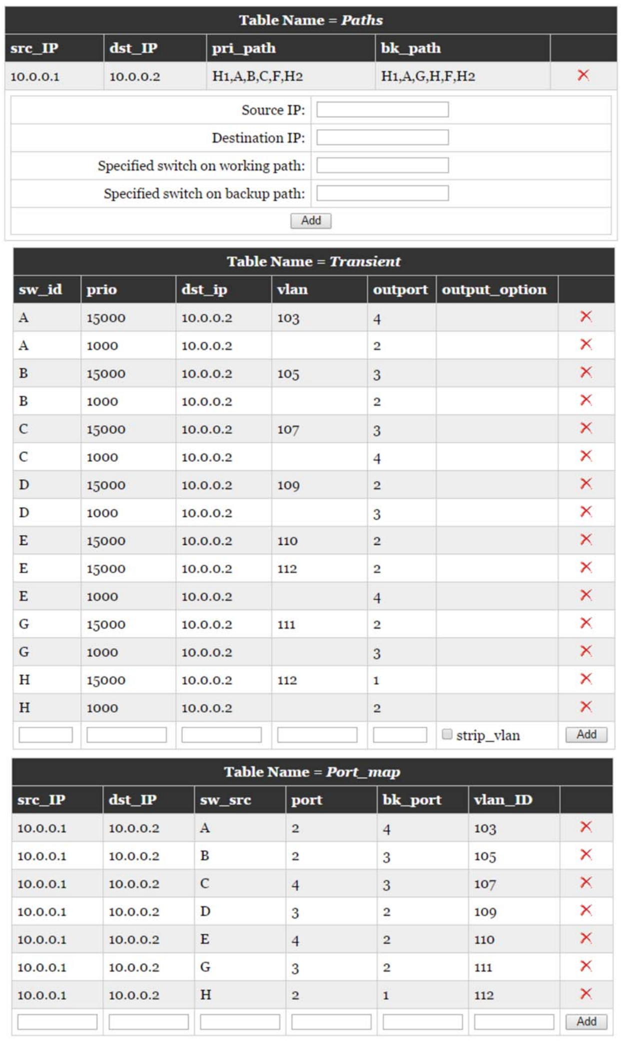


Table Name = *Paths*

src_IP	dst_IP	pri_path	bk_path	
10.0.0.1	10.0.0.2	H1,A,B,C,F,H2	H1,A,G,H,F,H2	✕

Source IP:	
Destination IP:	
Specified switch on working path:	
Specified switch on backup path:	
Add	

Table Name = *Transient*

sw_id	prio	dst_ip	vlan	outport	output_option	
A	15000	10.0.0.2	103	4		✕
A	1000	10.0.0.2		2		✕
B	15000	10.0.0.2	105	3		✕
B	1000	10.0.0.2		2		✕
C	15000	10.0.0.2	107	3		✕
C	1000	10.0.0.2		4		✕
D	15000	10.0.0.2	109	2		✕
D	1000	10.0.0.2		3		✕
E	15000	10.0.0.2	110	2		✕
E	15000	10.0.0.2	112	2		✕
E	1000	10.0.0.2		4		✕
G	15000	10.0.0.2	111	2		✕
G	1000	10.0.0.2		3		✕
H	15000	10.0.0.2	112	1		✕
H	1000	10.0.0.2		2		✕
					☐ strip_vlan	Add

Table Name = *Port_map*

src_IP	dst_IP	sw_src	port	bk_port	vlan_ID	
10.0.0.1	10.0.0.2	A	2	4	103	✕
10.0.0.1	10.0.0.2	B	2	3	105	✕
10.0.0.1	10.0.0.2	C	4	3	107	✕
10.0.0.1	10.0.0.2	D	3	2	109	✕
10.0.0.1	10.0.0.2	E	4	2	110	✕
10.0.0.1	10.0.0.2	G	3	2	111	✕
10.0.0.1	10.0.0.2	H	2	1	112	✕
						Add

Fig. 15. Tables in the database.

failure, they generate an OFPPR_DELETE port status message and send it to the controller. The controller retrieves the failed link ID and backup port number from the *Port_map table*. The controller then reroutes packets that will be affected by the failure from the working plane to the transient plane by sending an OFPFC_ADD message (which adds a flow entry as a connection to the transient plane) to switches C and F. This flow entry deflects incoming packets directed toward the failed link to the transient plane via the backup ports on the switches. This message also adds a flow tag to the packet header with the failed link ID that is used to identify the failed link and route packets on the transient plane. The controller then sets up the permanent backup path by selecting a backup path from its *Paths table* and configuring the corresponding switches. It also removes the old primary path.

When the controller starts, the working path and the transient plane are read from the *Path table* and *Transient table*, respectively. The controller configures each switch using command "*ovs-ofctl dump-flows xx*," where *xx* is the switch_id. Flows for the transient plane are added into each switch except switch F, which is the root. The working plane is added to switches A, B, C, and F, which are the working path. During normal operations, 575 packets travel along the working path, as shown in Fig. 16.

The link between C and F is then broken by the command "link C F down" on Mininet. After the link failure, switch C adds the connection to the transient plane, which is the flow with priority 20,000. It should be noted that the connection to the transient plane should also be added at switch F. However, we omit this step since this simulation sends data in a single direction. The connection to the transient plane is added to switch C. Flows for the working path are removed from switches A, B, and C. Flows for the backup path are added to switches A, G, and H. At switch C, we observe 48 packets traveling to the transient plane via backup port #3. These packets travel to $H2$ via C, E, H, F, thus proving the functionality of the transient plane. After the backup path is activated, 2100 packets travel on the backup path via A, G, H, F to $H2$, as shown in Fig. 17, showing that the network has switched over to the backup path. We also confirm that there is no packet loss during the experiment by verifying the packet sequence number at $H2$. All packets are received at $H2$. However, some packets arrive at the destination out of sequence during the switchover. This is caused by the configuration delay and propagation delay that occur when the path is changed.

Moreover, the performance of topologies as in Fig. 14 with the number of source–destination pairs is investigated in both functionality and switchover time. Eight and 14 source–destination pairs are generated for the sample and NSF topologies, respectively. In the simulation, computers with Intel Core i7-3517U CPUs at 1.90 GHz with 8 GB of RAM are used to run Mininet. In the sample topology, link failure between nodes 3 and 6 is simulated. The ITP function performs correctly after the failure. The switchover time is measured, which is 47.2 ms as in Fig. 18(a), on the source–destination pair between 1 and 6. In the NSF network topology, link failure between nodes 9 and 10 is simulated. The ITP function also performs correctly after the failure. The switchover time is measured, which is 66.4 ms as in Fig. 18(b), on the source–destination pair between 1 and 14. Furthermore, we confirm that the switchover time does not vary appreciably when increasing the number of source–destination pairs. When testing with 1, 14, and 28 of the source–destination pairs in the NSF network topology we obtained switchover times of 66.11, 66.40, and 67.97 ms, respectively.

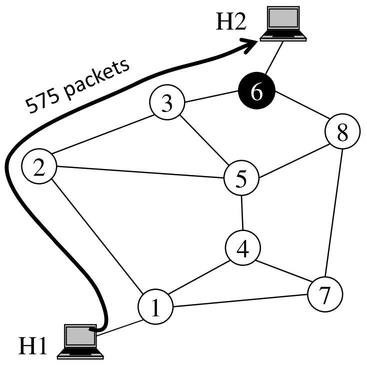


Fig. 16. Packet analysis before failure.

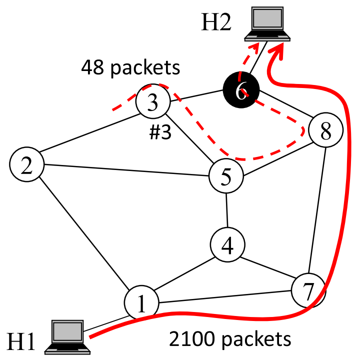


Fig. 17. Packet analysis after failure.

VIII. Testbed Experimental Results

Having proven the functionality of the ITP design with Mininet, we devised an experiment to measure the switchover time of the ITP on a real network. In fact, although Mininet can emulate link latency by configuring link propagation delay parameters, a realistic switchover time may not be obtained since there are other parameters that are not configurable, such as the delay between the

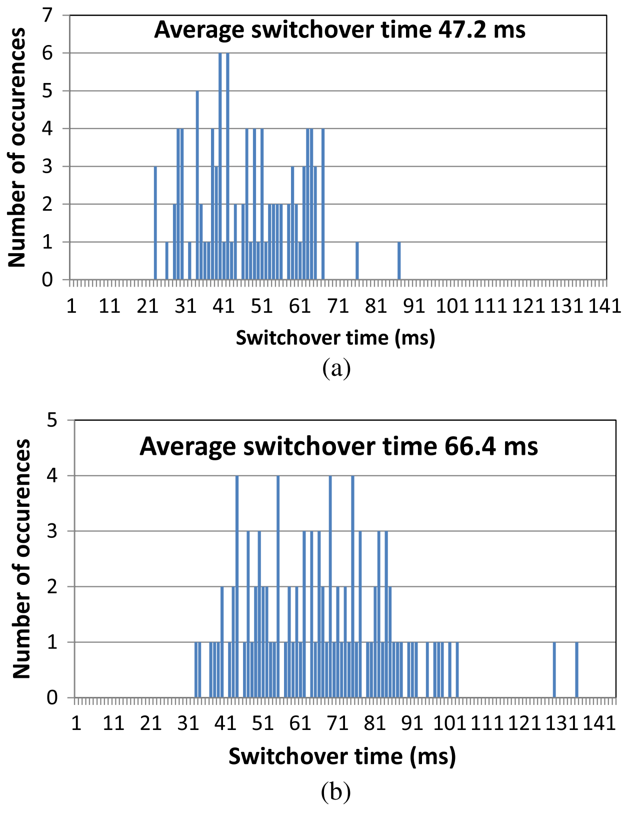


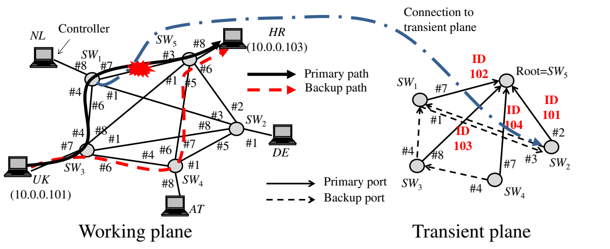
Fig. 18. Switchover time measured from Mininet. (a) Switchover time for a sample topology; (b) switchover time for NSF topology.

controller and each switch. Moreover, the computer running Mininet may also be running background processes, which could skew the results. We have thus run an experiment on the pan-European research and education (GÉANT) [18] OpenFlow facility network.

A. Experimental Setup

The GÉANT network nodes used in this experiment are deployed at five different locations around Europe, namely Amsterdam (NL–SW_1), Frankfurt (DE–SW_2), London (UK–SW_3), Vienna (AT–SW_4), and Zagreb (HR–SW_5). The topology we have created is a full mesh and is shown at the top of Fig. 19(a). The SDN controller, implemented in POX, is located at a server in the NL node.

In the scenario we have created for this experiment, as shown in Fig. 19(a), a working path UK → SW_3 → SW_1 → SW_5 → HR will be replaced by a backup path UK → SW_3 → SW_4 → SW_5 → HR when the former fails. Both paths are stored in the *Paths* table. The transient plane is designed as Fig. 19(b). We assume the link failure occurs between SW_1 and SW_5. Port #1 of SW_1 is set as a backup port. After the link failure, the controller configures the backup path. On-the-fly packets travel from the working plane to the transient plane via the connection to the transient plane during the configuration. We capture such packets at the DE node, while packets traveling on the backup route are captured at the AT node.



Connection to transient plane

(a)

*SW*1

Priority	nw_src	nw_dst	dl_vlan	Action
10000	10.0.0.101	10.0.0.103		output:7
1000		10.0.0.103		output:7
15000		10.0.0.103	102	output:1

*SW*2

Priority	nw_src	nw_dst	dl_vlan	Action
1000		10.0.0.103		output:2,1
15000		10.0.0.103	101	output:3

*SW*3

Priority	nw_src	nw_dst	dl_vlan	Action
10000	10.0.0.101	10.0.0.103		output:4
1000		10.0.0.103		output:8
15000		10.0.0.103	103	output:4

*SW*4

Priority	nw_src	nw_dst	dl_vlan	Action
1000		10.0.0.103		output:7
15000		10.0.0.103	104	output:4

*SW*5

Priority	nw_src	nw_dst	dl_vlan	Action
10000		10.0.0.103		output:8, strip_vlan

(b)

*SW*1

Priority	nw_src	nw_dst	dl_vlan	Action
20000	*10.0.0.101*	*10.0.0.103*		*output:1, vlan=102*
10000	10.0.0.101	10.0.0.103		output:7
1000		10.0.0.103		output:7
15000		10.0.0.103	102	output:1

*SW*2

Priority	nw_src	nw_dst	dl_vlan	Action
1000		10.0.0.103		output:2
15000		10.0.0.103	101	output:3

*SW*3

Priority	nw_src	nw_dst	dl_vlan	Action
10000	10.0.0.101	10.0.0.103		output:6
1000		10.0.0.103		output:8
15000		10.0.0.103	103	output:4

*SW*4

Priority	nw_src	nw_dst	dl_vlan	Action
10000	10.0.0.101	10.0.0.103		output:7,8
1000		10.0.0.103		output:7
15000		10.0.0.103	104	output:4

*SW*5

Priority	nw_src	nw_dst	dl_vlan	Action
10000		10.0.0.103		output:8, strip_vlan

▭ Working plane ▨ Transient plane

Fig. 19. GÉANT network setup showing the number of configuration message when the primary path is changed. (a) Flow tables before link failure; (b) flow tables after link failure.

In the figure we show for simplicity only a number of parameters, such as priority, IP address (nw_src), destination IP address (nw_dst), VLAN (dl_vlan), and actions, although in reality other parameters of the MAC, MPLS, IP, and TCP protocols could also be used for flow matching.

The initial flow table in every switch is set as in Fig. 19(a). Flow entries for the transient plane are set in every switch with priorities of 1000 and 15,000. A flow entry for the working plane is injected into the switches along the working path only, which in this example is SW_1, SW_3, and SW_5, with priority 10,000.

We simulate a failure by triggering a ping message from the AT node to the controller (we use this artifice because we are not allowed to configure any switch on the GÉANT network by command line). After the controller receives the trigger, a connection to the transient plane is injected into SW_1, as in Fig. 19(b). The backup path is then configured. At SW_3, the output on the flow entry with priority 10,000 is changed from #4 to #6. At SW_4, a new flow entry with priority 10,000 is added. It should be noted that output #1 at SW_2 and output #8 at SW_4 are used for the purpose of packet monitoring at the DE and AT nodes, respectively. The other flow entries remain unchanged.

B. Results and Evaluation

We initially measure the round trip time between the UK and HR nodes, which is approximately 45 and 50 ms, respectively, for the working and backup paths.

Next, we confirm that on-the-fly packets travel to the transient plane when a failure occurs. This is done using a client and server socket program that generates a packet every millisecond at UK with a destination of HR. During normal operation, no packets are captured on the transient plane at DE or the backup path at AT; thus packets are traveling on the working path. After the link failure, 95 packets appear on the transient plane at DE. This confirms that the packets are sent out at port #1 of SW_1 to the transient plane due to the connection to the transient plane. It should be noted that these 95 packets would be lost if the transient plane was disabled. After that, the transient plane at DE receives no more packets, and instead the packets begin to arrive on the backup path at AT. This implies that the packets travel on the transient plane, via SW_2, before the configuration of the backup path and via SW_4 when the configuration is completed. We also check the network for packet loss and find that no packet loss occurs during the protection process.

From this analysis, one and two new flow entries are required for switches on the working and transient planes, respectively. Two flow messages, which change the output from port #4 to port #6 on SW_3 and add an output to port #7 to SW_4, are required to change from the working to backup paths. One flow message is required for the connection to the transient plane on SW_1.

Finally, the switchover time is measured by capturing packets at the HR node. The switchover time is measured

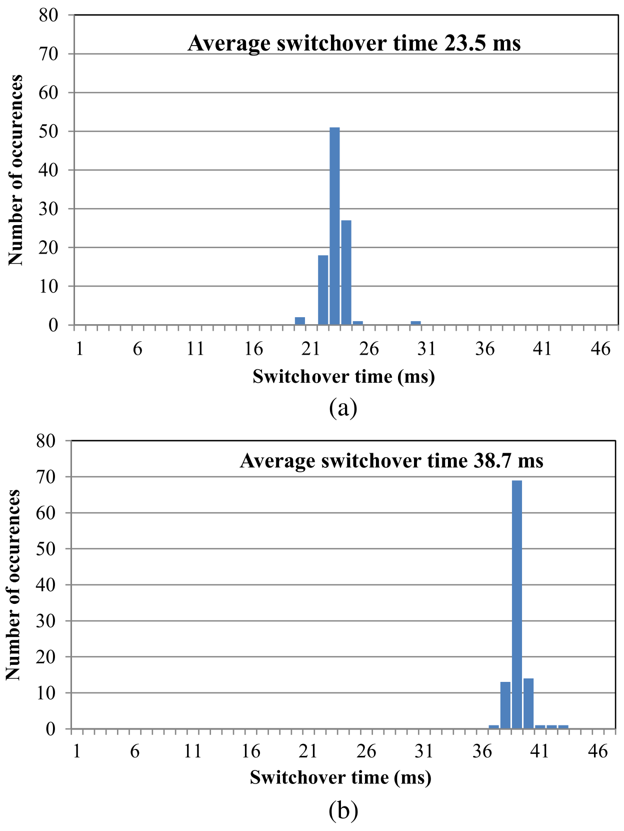


Fig. 20. Switchover time. (a) Switchover time with ITP; (b) switchover time without ITP.

by timing the difference of arrival time between the last received packet before the failure and the first received packet after the failure. The failure between SW_1 and SW_5 is repeated 100 times. Figure 20(a) shows the switchover time distribution obtained using ITP. The distribution is concentrated in the range of 22–24 ms, with an average of 23.5 ms. Figure 21 shows a breakdown of the switchover time. The switchover time includes the time during which packets are queued because of the failure and the difference in propagation delay between $SW_1 \rightarrow SW_5$ and $SW_1 \rightarrow SW_2 \rightarrow SW_5$. Packet queuing times include the controller processing time, the time required to send the configuration of the connection to the transient plane from the controller to SW_1, and the flow installation time at SW_1. This takes approximately 17 ms. The remaining time is the difference in propagation delay between the primary and backup routes, which in this instance is approximately

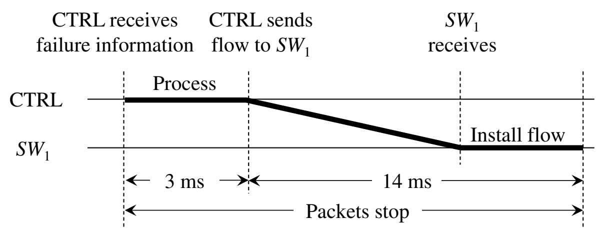


Fig. 21. Time chart for failure processing.

6.5 ms. Figure 20(b) shows the results of the same experiment carried out using a basic version of segment protection without ITP for comparison. From these results we can see that ITP improves the switchover time by approximately 40% for this network. Furthermore, we have tested the same scenario with a larger number of flows (using five source–destination pairs), obtaining the same results.

IX. Conclusion

This work presented a novel network protection mechanism, called ITP, implemented through OpenFlow. The advantages of the design are multifold: first it reduces the number of flow entries required in the switches' forwarding tables and the number of configuration messages from the controller, which impacts the size of the required TCAM memory in each switch and CPU process in the controller. Second, it reduces packet loss, as a temporary fast reroute path is made available while the network controller informs all switches about the new backup route to be used. Our analysis shows that our ITP design can reduce the number of flow entries by 60% and the number of configuration messages by 75% when compared to the existing segment protection designs. Moreover, the design can achieve a switchover time of approximately 25 ms.

Acknowledgments

This work is supported in part by the Science Foundation of Ireland through the CTVR CSET grant 10/CE/I1853 and in part by the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 318137 (Collaborative project DISCUS) [19,20].

References

[1] M. Goyal, K. K. Ramakrishnan, and W.-c. Feng, "Achieving faster failure detection in OSPF networks," in *Proc. IEEE Int. Conf. on Communications (IEEE ICC)*, 2003, pp. 296–300.

[2] Metro Ethernet Forum (MEF), "Requirements and framework for Ethernet service protection in metro Ethernet networks," Tech. Spec. MEF 2, 2004.

[3] J. Zhang, J. Zhou, J. Ren, and B. Wang, "A LDP fast protection switching scheme for concurrent multiple failures in MPLS network," in *Proc. Int. Conf. on Multimedia Information Networking and Security (MINES)*, 2009, pp. 259–262.

[4] L. Hundessa and J. Domingo-Pascual, "Reliable and fast rerouting mechanism for a protected label switched path," in *Proc. IEEE Global Telecommunications Conf. (IEEE GLOBECOM)*, 2002, pp. 1608–1612.

[5] B. Jaumard, N. Nahar Bhuiyan, S. Sebbah, F. Huc, and D. Coudert, "A new framework for efficient shared segment protection scheme for WDM networks," in *Proc. Int. Conf. on High Performance Switching and Routing (HPSR)*, 2010, pp. 189–196.

[6] B. Kantarci, H. T. Mouftah, and S. Oktug, "Availability analysis and connection provisioning in overlapping shared segment protection for optical networks," in *Proc. 23rd Int. Symp. on Computer and Information Sciences (ISCIS)*, Istanbul, Turkey, 2008.

[7] J. Tapolcai, P.-H. Ho, D. Verchere, T. Cinkler, and A. Haque, “A new shared segment protection method for survivable networks with guaranteed recovery time,” *IEEE Trans. Reliab.*, vol. 57, no. 2, pp. 272–282, 2008.

[8] D. McDysan, “Software defined networking opportunities for transport,” *IEEE Commun. Mag.*, vol. 51, no. 3, pp. 28–31, 2013.

[9] “Software-defined networking (SDN): The new norm for networks,” ONF White Paper, Apr. 2012 [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf.

[10] R. Kanagavelu, L. Bu Sung, R. Felipe Miguel, L. N. T. Dat, and L. N. Mingjie, “Software defined network based adaptive routing for data replication in data centers,” in *Proc. 19th IEEE Int. Conf. on Networks (IEEE ICON)*, 2013.

[11] S. Sharma, D. Staessens, D. Colle, M. Pickavet, and P. Demeester, “Fast failure recovery for in-band OpenFlow networks,” in *Proc. 9th Int. Conf. on the Design of Reliable Communication Networks (DRCN)*, 2013, pp. 52–59.

[12] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, “OpenFlow: Enabling innovation in campus networks,” Mar. 2008 [Online]. Available: http://archive.openflow.org/documents/openflow-wp-latest.pdf.

[13] A. Lara, A. Kolasani, and B. Ramamurthy, “Network innovation using OpenFlow: A survey,” *IEEE Commun. Surv. Tutorials.*, vol. 16, no. 1, pp. 493–512, 2014.

[14] A. Sgambelluri, A. Giorgetti, F. Cugini, F. Paolucci, and P. Castoldi, “OpenFlow-based segment protection in Ethernet networks,” *J. Opt. Commun. Netw.*, vol. 5, no. 9, pp. 1066–1075, 2013.

[15] N. Kitsuwan, F. Slyne, S. McGettrick, D. B. Payne, and M. Ruffini, “A Europe-wide demonstration of fast network restoration with OpenFlow,” *IEICE Commun. Express*, vol. 3, no. 9, pp. 275–280, 2014.

[16] N. Kitsuwan, D. B. Payne, and M. Ruffini, “A novel protection design for OpenFlow-based networks,” in *Proc. 16th Int. Conf. on Transparent Optical Networks (ICTON)*, Graz, Austria, 2014, paper We.A4.3.

[17] Mininet, http://mininet.org/.

[18] GÉANT network, http://geant3.archive.geant.net/.

[19] M. Ruffini, N. Doran, M. Achouche, N. Parsons, T. Pfeiffer, X. Yin, H. Rohde, M. Schiano, P. Ossieur, B. O’Sullivan, R. Wessaly, L. Wosinska, J. Montalvo, and D. B. Payne, “DISCUS: End-to-end network design for ubiquitous high speed broadband services (Invited),” in *Proc. 16th Int. Conf. on Transparent Optical Networks (ICTON)*, Cartagena, Spain, 2013, paper We.B3.1.

[20] M. Ruffini, L. Wosinska, M. Achouche, J. Chen, N. J. Doran, F. Farjady, J. Montalvo, P. Ossieur, B. O’Sullivan, N. Parsons, T. Pfeiffer, X.-Z. Qiu, C. Raack, H. Rohde, M. Schiano, P. Townsend, R. Wessaly, X. Yin, and D. B. Payne, “DISCUS: An end-to-end solution for ubiquitous broadband optical access,” *IEEE Commun. Mag.*, vol. 52, no. 2, pp. S24–S32, Feb. 2014.

Nattapong Kitsuwan received B.E. and M.E. degrees in electrical engineering (telecommunication) from Mahanakorn University of Technology, King Mongkut’s Institute of Technology, Ladkrabang, Thailand, and a Ph.D. in information and communication engineering from the University of Electro-Communications, Japan, in 2000, 2004, and 2011, respectively. From 2002 to 2003, he was an exchange student at the University of Electro-Communications, Tokyo, Japan, where he performed research regarding optical packet switching. From 2003 to 2005, he was working for ROHM Integrated Semiconductor, Thailand, as an Information System Expert. He was a post-doctoral researcher at the University of Electro-Communications. He currently works for the Telecommunications Research Centre (CTVR), Trinity College Dublin, Ireland. His research focuses on optical networks, optical burst switching, optical packet switching, scheduling algorithms, and software-defined networks.

Séamas McGettrick (mcgettrs@tcd.ie) is a research fellow at CTVR, the Telecommunications Research Centre at Trinty College Dublin. He is interested in reconfigurable hardware and how it can be used to solve complex real-world problems. He is currently researching and prototyping protocols for next-generation passive optical networks.

Frank Slyne is completing his Ph.D. at Trinity College Dublin in the research area of metro nodes as implemented in a flat-core and long-reach passive optical network (LR-PON) architecture. His interests are in the application of software-defined networks for the optimization of energy, capacity, and performance. He has worked for a number of years at Eircom, where he was responsible for ISP platforms and systems. He has a B.E. (Elect) from UCC (Cork) and an M.Eng. from DCU (Dublin).

David B. Payne spent most of his career with BT, where in his latter years he was responsible for broadband and optical networks research. After leaving BT in 2007, he did consultancy work before joining TCD as a professor in the Department of Optical Networks, and more recently he also joined Aston University. He now coordinates the DISCUS EU project.

Marco Ruffini (marco.ruffini@tcd.ie) received his M.Eng. in telecommunications in 2002, and after working as a research scientist for Philips in Germany he joined Trinity College Dublin (TCD), where he received his Ph.D. in 2007. He is now an assistant professor at TCD and co-coordinator of the DISCUS project. He is a co-author of more than 50 publications and 8 patents. His research focuses on flexible high-capacity fiber broadband architectures.