PRoPHYS: Providing Resilient Path in Hybrid Software Defined Networks

Myriana Rifai, Dino Pacheco, Quentin Jacquemart, Guillaume Urvoy-Keller

PRoPHYS: Providing Resilient Path in Hybrid Software Defined Networks

Myriana Rifai, Dino Lopez Pacheco, Quentin Jacquemart and Guillaume Urvoy-Keller

I3S/CNRS/UCA

April 25, 2017

Abstract

The growing reliance on high-bandwidth, time-critical, and delay-sensitive technologies (such as video streaming, voice over IP, and virtual reality) leads to a dominance of delay-sensitive IP flows. Nowadays, a 50ms protection interval is considered tolerable by service providers [3] since most applications support the resulting loss with limited impact on the end-user's quality of experience (QoE). However, following the *increase* of required *broadband* speed per application [5], the amount of packets lost during those 50ms will also increase, leading to disruptions in the buffer playout process, and negatively impacting the QoE. It is therefore essential to enhance network resiliency techniques in order to *combat network failures*.

To tackle this problem, we devised PRoPHYS – Providing Resilient Path in Hybrid SDN – which leverages the centralized SDN controller *in hybrid networks* (SDN alongside legacy L2/L3 equipment) to monitor traffic or network segments, and quickly detect failures or network disruptions, to reroute traffic. PRoPHYS features two strategies (one passive and one active) to detect network segment failures; and is designed for smooth integration with legacy routing protocols, e.g., OSPF. Our evaluation of PRoPHYS illustrates that it can decrease downtime by 50% during segment link failures, thus reducing drastically the number of packets lost, while minimizing the number of false positives.

1 Introduction

The ever growing popularity of high-bandwidth and/or time-critical applications (such as video streaming, voice over IP, and virtual reality) leads to a global increase in IP traffic [5], which, consequently, raises the average monetary cost of network and data center outages [26]. In order to guarantee a good quality of experience for the end-user (QoE), it is essential to *improve fault tolerance* mechanisms so as to protect these network flows against large-scale packet loss.

Existing fault detection techniques rely on *link failure detection*, which is achieved by detecting missing session packets. In OSPF, these are so-called `HELLO` packets [23], and the process takes a few seconds. The equivalent for Software-Defined Networking (SDN) networks are LLDP packets [6], and the

process takes, by default, 100ms. The main difference between legacy and SDN networks is that in SDN networks, the controller has a global view of the network topology, and communicates with every SDN nodes. Thus the controller can change the routes of all nodes directly, without any delay needed for routing information propagation between network nodes and convergence. With the help of additional services, such as Multi-protocol Label Switching (MPLS) [3], Bidirectional Forwarding Detection (BFD) [16], or Fast Rerouting [29], the link failure detection time can be decreased to around 50ms for both kinds of networks.

Coincidently, 50ms is considered to be tolerable [3] by service providers, since most applications support the resulting loss with little impact on the QoE. Nevertheless, as the available bandwidth, and the applications' bandwidth requirements increase [5], the number of packets (and, thereby, the amount of data) lost during these 50ms will also increase. For this reason, *network resiliency must be improved in order to decrease the total downtime (detection and rerouting time) below the currently acceptable 50ms.*

Since SDN networks do not need to reconverge upon changes in the control plane, the programmability of the SDN control plane is of massive help to methodologies to decrease the failure detection time. Unfortunately, the migration from an existing legacy network infrastructure to a full SDN infrastructure is known to be expensive and cumbersome [25]. In the meantime, so-called *hybrid* networks, where SDN nodes are incrementally introduced in legacy networks, are used.

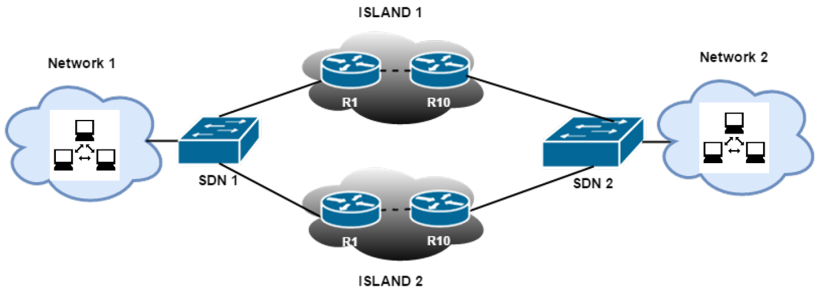


Figure 1: Example topology for PRoPHYS.

In this context, we propose PRoPHYS: Providing Resilient Path in Hybrid Software-Defined Networks. PRoPHYS leverages the capabilities of the SDN controller in a hybrid network to detect network failure and reroute traffic. It features two modes of action: (*i*) the passive monitoring of the transmitted and received traffic of both SDN nodes (Section 2), or (*ii*) the active probing of the paths connecting SDN nodes (Section 3). These two modes of actions can be deployed independently or together to provide the lowest detection interval of both methods. Ideally, SDN nodes should be placed such that they can communicate using two disjoint paths. In this article, we focus on the necessary criteria and steps to ensure efficient flow protection between two SDN nodes. The placement of SDN nodes in a hybrid network to create disjoint paths is out of the scope of this document *e.g.* it has been studied in other articles [13]. In its simplest form, using Figure 1 for illustration, upon detection of a

segment failure within ISLAND 1, or attached to it, PRoPHYS reroutes traffic through the second-best shortest path (ISLAND 2), effectively decreasing the loss rate (Section 4). Our results (Section 5), show that the two varieties of failure detection module of PRoPHYS provide a minimal decrease of 50% in total network traffic loss compared to the standard 50ms protection interval while respecting the maximum number of events supported by SDN nodes.

2 Passive Probing Failure Detection Methodology

The first methodology that PRoPHYS employs to detect the failure of a *segment* – either a link failure within a legacy island, or a link failure between an SDN router and the legacy island – is a passive monitoring of the network traffic at the set of transmitting and receiving SDN nodes. After receiving SDN ports statitics, it searches for discrepancies between the inbound and outbound traffic. Hence, the Passive Probing Failure Detection Methodology follows the following steps: (*i*) create the matrix of the communicating SDN ports; (*ii*) monitor the ports; (*iii*) upon reception of ports statistics, trigger the failure detection module. We now detail each step successively in the remainder of this Section.

2.1 Matrix of Communicating SDN Ports

In SDN, when a new flow arrives in the network, the controller sets up the ad-hoc rules on every SDN switch used by the flow. Thus, the controller has a global view of all the segments used by the flows, and of the series pairs of communicating SDN ports. In other words, the controller is capable of building a communication matrix $\mathcal{M}_{\text{ports}}$, indicating whether an SDN switch port is sending traffic to another SDN port, or not. For example, for the two flows depicted in Figure 2, the controller builds the matrix illustrated in Table 1. A value of 1 indicates a transmission between two ports, a value of 0 indicates no transmission.

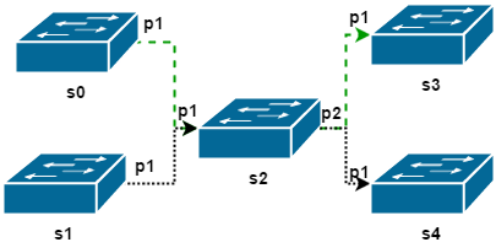


Figure 2: Flows passing through the network.

2.2 SDN Ports Monitoring

To detect traffic loss, ports statistics are fetched from the SDN nodes every few milliseconds. A reasonable value for the polling interval is the estimated delay between the two SDN nodes. Another one would be a value defined by the

		RECEIVERS		
		s2 p1	s3 p1	s4 p1
SENDERS	s0 p1	1	0	0
	s1 p1	1	0	0
	s2 p2	0	1	1

Table 1: SDN ports communication matrix $\mathcal{M}_{\text{ports}}$ as built within the SDN controller for the flows depicted in Figure 2.

network administrator. We believe the optimal value for the polling interval to be max(10ms, estimated delay). The 10ms minimum threshold reflects the fact SDN nodes support 200 (control) events/s [18]. Hence with this threshold, fetching port statistics requires a maximum of 100 events/s, which is feasible [18]. It should be noted that it only requires one single request from the SDN controller to get the statistics of all the ports from one SDN node. In other words, the number of required events does not increase with the number of ports, ensuring scalability.

Even though the granularity of port statistics is coarser than the granularity of flow statistics, we elected to consider ports statistics because (*i*) monitoring the ports instead of flows enables to scale, and (*ii*) SDN switches update the flow counters of their OpenFlow database only once every second [11]. Going below this 1s flow counter update threshold increases the CPU consumption of the switch, and the results returned to the controller will be delayed and inaccurate.

2.3 Failure Detection Module

After receiving the ports statistics from all the switches, the failure detection module checks if the outbound traffic from transmitting ports was received by their corresponding receiving ports. For example, if we take the matrix defined in Table 1 for the topology of Figure 2, the failure detection module will compare $T^t_{\text{s0p1}} + T^t_{\text{s1p1}}$ to $R^{t+\delta t}_{\text{s2p1}}$, where T^t_i is the traffic sent by port i at a given time t, and $R^{t+\delta t}_j$ is the traffic received by port j after δt. δ_t is the maximum observed delay for packets to transit from the upstreamer SDN ports to the downstreamer SDN ports i.e. $\delta t = \max\left(\delta \text{t}_{(\text{s0p1}-\text{s2p1})},\ \delta \text{t}_{(\text{s1p1}-\text{s2p1})}\right)$.

However, directly comparing the amount of transmitted traffic with the amount of received traffic δt later (*e.g.* $T^t_{\text{s0p1}} + T^t_{\text{s1p1}} = R^{t+\delta t}_{\text{s2p1}}$) might lead to false positives. Indeed, the maximum observed delay at a given time might be lower than the actual maximum delay that a packet can observe in a non-SDN island (e.g. due to high processing or queueing delays). To cope with this inaccuracy, in PRoPHYS, we compare only a fraction of sent traffic to the total volume of received traffic. Hence, our proposition detects a possible network segment failure in a non-SDN island if the following inequality holds:

$$(T^t_{\text{s0p1}} + T^t_{\text{s1p1}}) - \theta * (T^t_{\text{s0p1}} + T^t_{\text{s1p1}}) \leq R^{t+\delta t}_{\text{s2p1}} \tag{1}$$

Where θ represents the fraction of transmitted traffic that can still be missing at the receiving side due to the different network delays and $0 \leq \theta < 1$.

To confirm the segment failure and avoid false positive results, PRoPHYS checks the validity of Equation 1 up to *thresh_count* consecutively. If during

these *thresh_count* segment failure test, PRoPHYS always resolves to a failed network segment, then, PRoPHYS confirm the network failure and rerouting actions can take place now. Note that PRoPHYS checks the validity of Equation 1, each time new statistics are received from the concerned ports.

The introduction of the *thresh_count* variable should be set in a conservative, and, yet, non-intrusive way such that it allows to decrease the number of false positives while trying not to lose a huge amount of data traffic when the network failure is actually detected. It should be noted here that this methodology would not be able to distinct between high congestion periods that cause huge amount of losses and link failure. However, we believe that rerouting the traffic when high congestions rates cause huge packet losses in the segment is good as the traffic would be routed to different segments, preventing OSPF nodes in the legacy island to detect a false positive link failure and reroute the traffic which could cause higher delays or even higher packet loss due to the convergence period.

Moreover, since PRoPHYS is developed to be deployed in Internet Service Provider Autonomous System (ISP AS) domains, multicast traffic would not cause a problem as ISP AS domains have only few multicast control packets. These packets would not lead to false positives due to the utilization of θ and *thresh_count* variables which prohibit false positives due to network dynamics. And since PRoPHYS operates in ISP AS network domain, end user multicast traffic is usually used on private Intranets but it is not a service provided across the public Internet [20]. Add to that, we suppose in PRoPHYS that traffic drop policies (e.g. firewalls) are placed at the entry/exit points of the AS, hence the traffic would be filtered before or after being detected by PRoPHYS.

3 Active Probing Failure Detection Methodology

The second methodology that PRoPHYS suggests to detect the failure of a segment is a variation to the classical active probing methodology which inject messages in the network. However, the main difference between the classical and our active probing methodology lies in the fact that *PRoPHYS detects link failure when the probing is lost after a timeout that depends on the link or segment delay.*

Basically, the SDN controller transmits a probing packet to the SDN switches with the highest IP Precedence to provide highest priority, the switches then flood the probe to their SDN neighbors across the connected OSPF island. To achieve this, the controller generates a $SDDP$ Segment Discrepancy Detection packet for each SDN node. This packet will be transmitted from the controller, flooded through the SDN network segments, and back to the controller (see Figure 3) similar to LLDP packets [30]. However, SDDP can be transmitted over network segments instead of being limited to the directly connected link. And SDDP is also an enhancement compared to LLDP- which features a slow failure detection time of over 100ms- as it uses a dynamic timeout which depends on link/segment delay. Once the SDDP message is sent on the network interface, the controller starts a timeout timer that is set dynamically based on the estimated delay of the link/segment.

To calculate the *SDDP timeout* threshold for every link/segment, we use an exponential moving average approach. Before the reception of the first SDDP packet, the delay is set to 1s. Then we use equations similar to TCP's Round-Trip Time's [27] to calculate *SDDP timeout*. The estimated delay of the link/segment, d_l, is updated as depicted in Equation 2 where α is set to 0.125. The variation of the link/segment, var_l, is computed following Equation 3, and we set the timeout threshold to Equation 4 with K=4 (α and K are set to the same values used in TCP timeout).

$$d_l[i] = (1 - \alpha)\ d_l[i-1] + \alpha\ \text{new\_}d_l \tag{2}$$

$$\text{var}_l[i] = (1 - \beta)\ \text{var}_l[i-1] + \beta\ |d_l[i-1] - \text{new\_}d_l| \tag{3}$$

$$SDDP\_timeout = d_l + K\ \text{var}_l \tag{4}$$

Then similarly to other link failure detection active probing methods, if the message is not received back (by the controller) before the timer expires, the controller declares segment discrepancies. If the message is received *after* the timer has expired, the controller learns that the particular network segment is congested and modifies the timeout value.

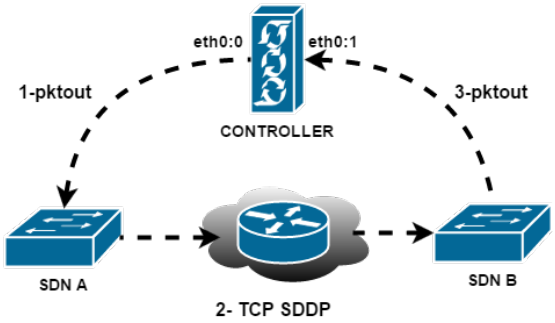


Figure 3: *Packet_out* transmission over the network.

The closest work to our SDDP packet is BFD Multipath [17]. Nonetheless SDDP differs from BFD Multipath because it does not need a session for every link and does not depend on the architecture of the SDN switch. *The performance of BFD depends on the manufacturer.* For example, the HP5412 switch provides a minimum of 1 second detection interval [12]. Moreover, the use of *BFD negatively impact the performance of routers, because the creation of the transmitted packets and their processing must be done at the general CPU of SDN switches*, which has very limited capacities in comparison to the CPU capacities of servers [28]. Because our methodology solely relies on the controller to take decisions, it is manufacturer independent, and it allows to decrease the CPU overhead on the switch.

4 Rerouting

After detecting a failure in the network, PRoPHYS reroutes the traffic away from the lossy link, to other paths of the network. To select the best path to use,

	Bandwidth(Mbps)	Time(ms)
ovs_bridge (SDN link)	943	8.9
ovs_stt (SDN tunnel)	934	9.1
ovs_gre (SDN tunnel)	919	9.1

Table 2: Bandwidth and time of transmission of 1000 MByte of data from a client to a server.

PRoPHYS computes a virtual topology graph by removing all lossy segments $G_{\text{virtual}} = G_{\text{real}} \setminus \{\text{lossy\_segment}\}$ from the initial topology G_{real}. Then the controller selects the shortest path in the virtual topology, and sends the new rules to be installed on the SDN switches.

However, since we consider hybrid networks, we should also account for the fact that legacy routers, in-between SDN nodes, will keep their outdated view of the routing topology, since their failure detection time is longer. Hence, the conflicting views, between the SDN controller and the OSPF nodes, could cause routing loops and denial of service. For example, if we take a topology similar to Figure 1, and assume that the number of hops on segment 1 is 3, while the number of hops on segment 2 is 6, the first OSPF node on segment 2 (R1) will send traffic destined to Network 2 through SDN1 (shortest path). Once SDN1 detects failures on segment 1, and PRoPHYS decides to reroute traffic through segment 2, R1 still has its (old) routing table since it did not detect any network updates. It will thus retransmit the traffic back to SDN1, or drop it altogether. Consequently, PRoPHYS will not be able to route the traffic from SDN1 to SDN2 via segment 2. To force OSPF routers to route traffic via the (new) path selected by PRoPHYS instead of the old, lossy path, we use tunnels to route traffic between SDN nodes and the first OSPF/SDN node that has the appropriate routing that allows it to reach the destination (in our example we use the tunnel between SDN1 and R2 in Island2). To know which nodes can route the traffic properly to the destination without any tunnels, we listen to OSPF update packets at the SDN nodes to rebuild the topology structure.

Using tunnels to reroute traffic has been suggested in [4] to handle single link failures, and avoid the failure detection period. In PRoPHYS, we leverage this solution and reroute traffic through tunnels, until reaching the closest node with the correct routing to the destination, where the traffic is decapsulated and transmitted to reach the end host.

However, it should be noted that routing data through tunnels can decrease the end-to-end bandwidth attainable by the client application. For instance, Table 2 reports the maximum attained bandwidth, and the delay required to route 1GB of traffic between two machines, through a tunnel active on these machines using either GRE (Generic Routing Encapsulation) [7], or STT (Stateless Transport Tunneling) [2] deployed over OpenVSwitches-based routers.

The end hosts are connected using 1Gbps Ethernet links. We observe that the bandwidth decreases, and that the delay of transmission increases, once we use tunnels (*ovs_gre* and *ovs_stt*) compared to no tunnels (*ovs_bridge*). Moreover, the choice of the tunneling protocol impacts the results: GRE obtains a lower bandwidth than STT, 919Mbps instead of 934Mbps. Hence, in PRoPHYS we use STT tunnels to reroute the traffic through OSPF islands.

5 Performance Evaluation

To test the performance and the impact of PRoPHYS on network traffic, we conducted a series of tests using a Floodlight v2 controller [14] connected remotely to a Mininet [22] network. The test network, depicted in Figure 4, consists of 5 SDN switches connected through OSPF islands. We set the delay between the network nodes and between SDN nodes and the controller to 2ms. The 2ms delay was motivated by the average link delay between nodes in germany50 ISP topology that was calculated based on the geographical coordinates of the links provided in SNDlib[32] In this network, h0 communicates with h1 and with h3; and h2 communicates with h3. The traffic is generated by iperf3 and is either bidirectional CBR (Constant Bit Rate) UDP traffic or TCP traffic. Five seconds after starting iperf3 on the hosts, we simulate a link failure in the OSPF island connecting the nodes s0 and s1 and observe the impact of PRoPHYS. We repeat each experience 100 times to obtain statistical significance.

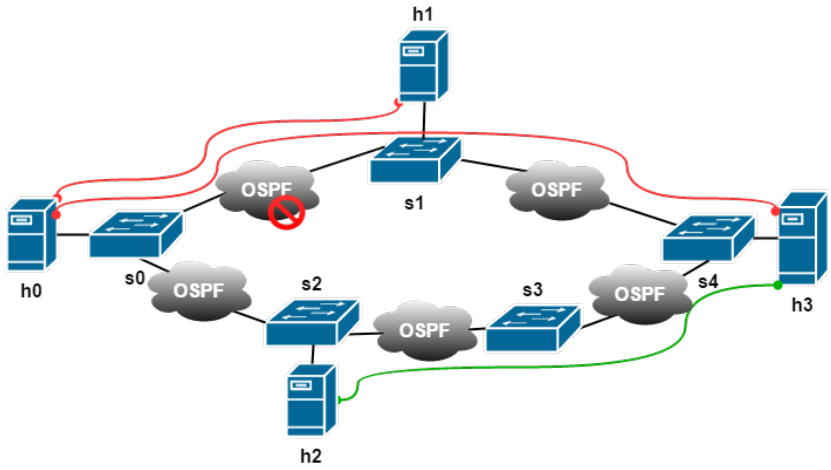


Figure 4: The SDN testing network topology in Mininet.

In all of these experiments, we set a segment monitoring interval of 10ms which is equal to the minimal monitoring interval, as discussed in Section 2. We run two versions of PRoPHYS that differ based on the failure detection technique they use, either Passive Probing (Section 2) or Active Probing (Section 3). For the Passive Probing, we declare segment failure when the following two conditions apply: (i) the number of packets lost or not yet arrived (θ) is higher than X% of the traffic transmitted in the last iteration, where $X \in \{25\%, 50\%, 75\%, 90\%\}$, and (ii) for $thresh\_count = 2$ consecutive iterations. For the Active Probing methodology, we declare a failure when the delay between transmitting and receiving the packet on the other interface of the controller is higher than the SDDP timeout delay introduced in Section 3. For the sake of briefness, in the remainder of this Section, we refer to the Passive Probing methodology as *PortStats*, and to the Active Probing methodology as *PktOut*.

METHODOLOGY	BANDWIDTH			
	1Mbps	10Mbps	20Mbps	40Mbps
50ms threshold	5	42	84	168
PktOut	2	16	30	63
PortStats 25%	3	20	35	71
PortStats 50%	5	23	35	83
PortStats 75%	3	32	35	79
PortStats 90%	2	22	43	84

Table 3: Maximum number of packets lost.

5.1 Impact on Network Traffic

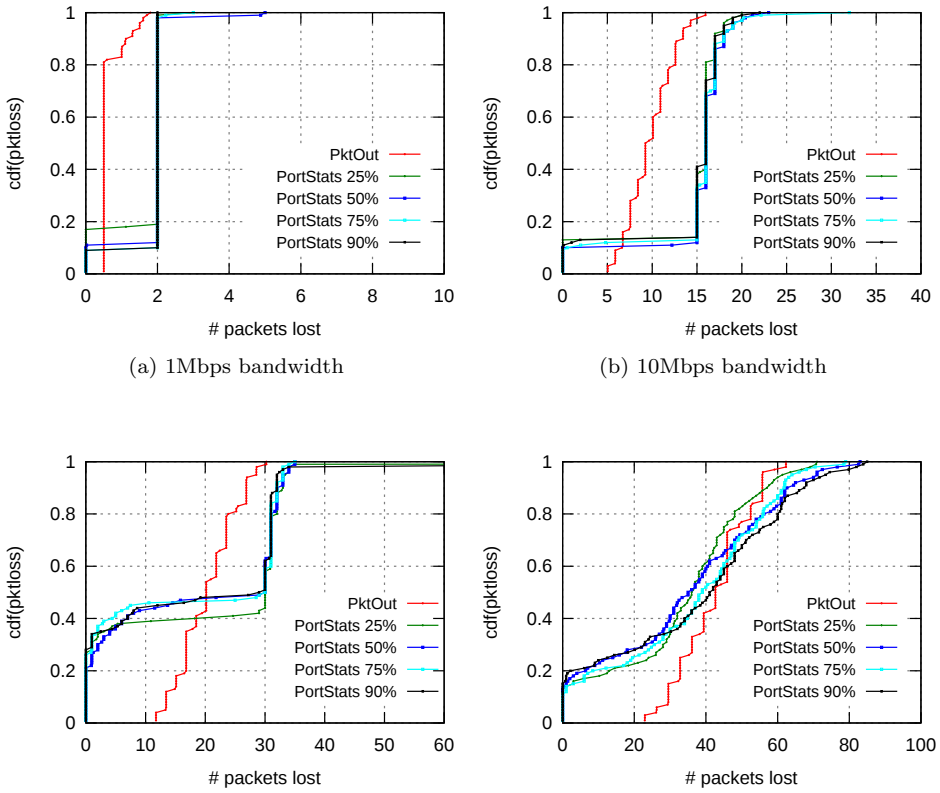
5.1.1 UDP experiments

We assess the performance of the PRoPHYS variants first by comparing them with the *reference case* of 50ms. Specifically, we look at the maximum number of packets lost per connection (over the 100 experiments) for different connections bandwidths. The results are shown in Table 3, where we notice that the maximum number of packets lost using PRoPHYS, for all speeds, and for both detection methodologies, is 50% less than the reference case, which can be translated by a maximum downtime of 25ms.

In Figure 5, we go beyond the maximum value and plot the CDFs of the number of packets lost with PRoPHYS, for both detection methods, and each bandwidth. We notice first that as the connection bandwidth increases, PortStats provides better results than PktOut. For example, for 1Mbps (Figure 5a), only 10% of the connections obtained less packet lost using PortStats than PktOut. However, as the bandwidth speed increases from 1Mbps to 40Mbps, (see Figures 5b-5d), PortStats tends to have better performance than PktOut. Specifically, for 40Mbps, 60% of the connections using the PortStats 50%, PortStats 75%, and PortStats 90%), as well as up to 90% of the connections using PortStats 25% feature less lost packets than PktOut. These results are in line with intuition. Indeed, the higher the rate, the higher is the number of packets that should be lost or delayed before declaring a failed link which decreases the false positive probability of PortStats. In contrast, the higher the traffic rate, the larger the SDDP *timeout* delay in Equation 4, which increases its detection time.

We also notice in Figure 5, that the exact value of θ (i.e., the tolerable percentage of packets lost or not received that ranges between 25 and 90%) for PortStats has little impact on the number of packets lost. This is because even if a lower θ might lead to detect a failure quicker (potentially doing a false positive), it is counterbalanced by the fact that two such consecutive observations must be made (`thresh_count` $= 2$).

In Figure 5, we also notice that, using PortStats, we can obtain 0% packet loss. This is due to false positives detections. Indeed, as shown in Figure 6, PortStats tends to have between 1 and 5 false positives in our tests. Some of these false positives can occur before the actual link failure, hence all the traffic would have been rerouted from the original path before the segment actually fails, leading to 0% loss rates. In contrast, in all of our tests PktOut had zero false positives, which proves its stability. From these results we conclude that



(a) 1Mbps bandwidth

(b) 10Mbps bandwidth

(c) 20Mbps bandwidth

(d) 40Mbps bandwidth

Figure 5: Packets loss of connections using the failing link.

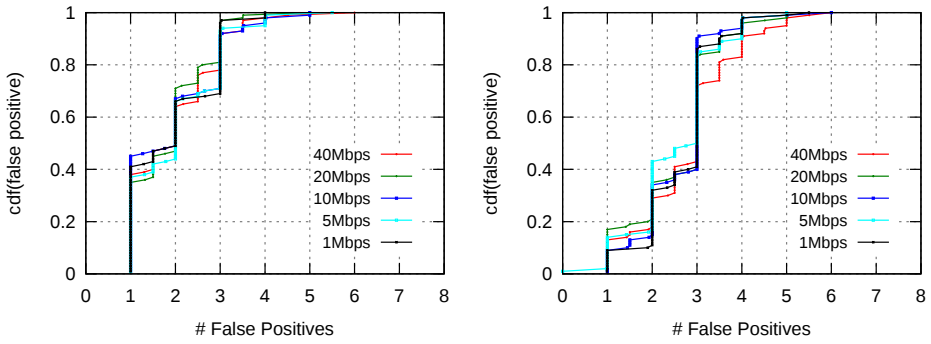
PktOut is more stable than PortStats, however PortStats shows better results as the bandwidth increases.

As a final note on our UDP experiments, it is important to point out that the variability observed in all figures is due to the OvS switches and the PRoPHYS implementation (within Floodlight), rather than in iperf3 or Mininet, since the UDP sources emit CBR traffic and the network is well provisioned.

5.1.2 TCP Experiments

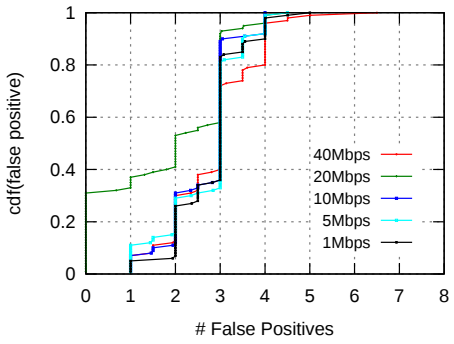
After studying the impact of PRoPHYS using CBR traffic and having proven its effectiveness in that context, we study the impact of PRoPHYS on TCP traffic. We thus repeat the experiments with the PktOut method, and with the PortStats 50% method for TCP connections. One UDP flow is replaced by one TCP flow, with an average speed of 10Mbps.

Without PRoPHYS, TCP connections lose connectivity, as they time out before OSPF re-converges. With PRoPHYS, all connections were able to survive the link failure, and the number of retransmissions was low, as seen in Figure 7. In this figure, we see that the number of retransmissions reached a maximum of 25 packets for PortStats50%, and of 20 packets for PktOut. These values

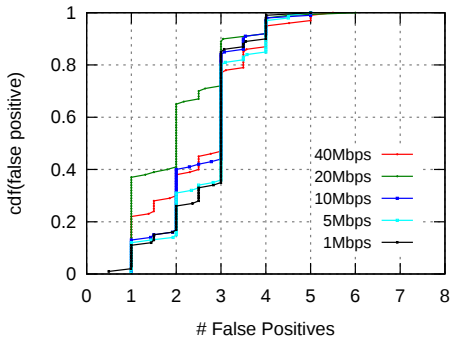


(a) Threshold=25%

(b) Threshold=50%



(c) Threshold=75%



(d) Threshold=90%

Figure 6: Number of false positive detections of segment failures with PortStats.

are far below the 42 packets lost that a 50ms recovery time would generate, see Table 3.

We observe that in a few % of cases, both methods achieve a 0% retransmission rate. This means that both methods do false positives, i.e., traffic was rerouted before the failure actually occurs. Still, the rate of false positives of PktOut remains below the one of PortStats, in line with the observations made for UDP traffic. We however observed that the false positives of PktOut are made in the warm up phase of the tests, when the TCP window opens widely and delay is the most difficult to track. We can expect that with real ISP traffic where the level of multiplexing is high and delay variations smoother, the false positive rate of PktOut remains small. This observation of false positives for both methods hints towards combining them to minimize the false positive rate.

5.2 Impact of the Segment Delay on PortStats

In order to study the impact of the segment delay on the 10ms monitoring interval that we have set for the previous experiments, we repeat the PortStats 50% UDP experiment with a 10Mbps bandwidth, while varying the segment delay. We set the delays to 5ms, 10ms, 15ms and 20ms. Figure 8a shows that

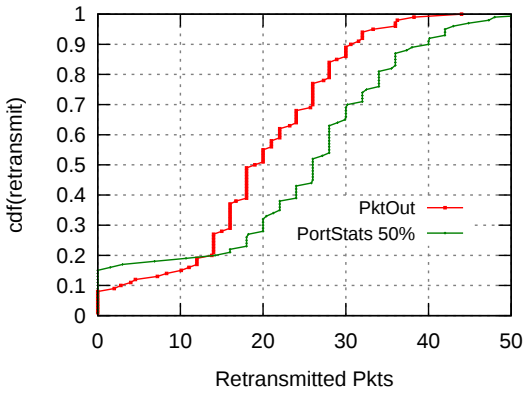
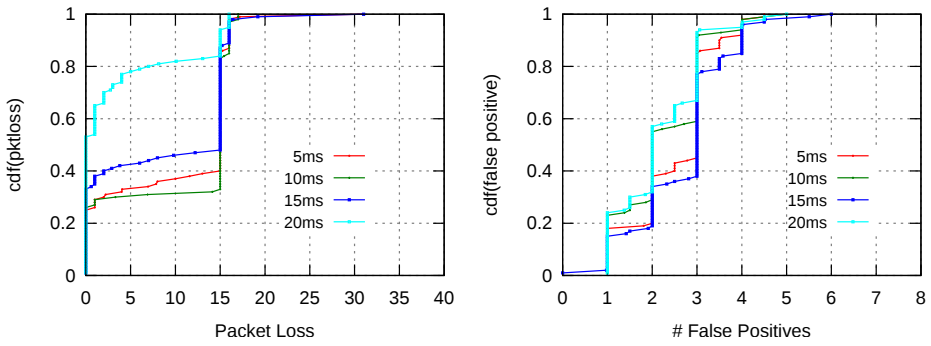


Figure 7: Number of packets retransmitted by TCP.

the number of packets lost is the lowest when the segment delay is 20ms and $\theta = 50\%$. This is due to the fact that, with a 20ms segment delay, we tend to receive, after 10ms, exactly 50% of the transmitted traffic. However, the number of packets lost increases slightly as the segment delay decreases. Hence we advise to set the monitoring interval, and the variable θ based on the delay obtained on the segments connecting the SDN nodes.



(a) CDF of packets lost

(b) CDF of false Positives

Figure 8: Packet loss and false positive variation with the variation of delay on the failed island using PRoPHYS PortStats 50% methodology

Since PRoPHYS was mainly created for mid size ISP topologies, the placement of the controller in the ISP network highly influences the detection delay results of PRoPHYS as the RTT delay between the controller and the switch makes part of the detection interval. Thus the controller placement should be optimized to minimize the delay between the switches and the controller in the ISP networks [10].

6 Related Work

Hybrid Networks To allow full network programmability, some propositions try to propagate SDN functionalities from SDN to legacy nodes in hybrid networks, such as [19, 15]. The authors of Panopticon [19], tried to propagate the enhanced functionality of the SDN nodes to the legacy nodes, however, they managed to decrease the failure detection interval to a minimum of 1s as they depend on the Spanning Tree Protocol (SPT) to detect a failure. In Telekinesis [15], the authors leveraged the SDN controller to force legacy L2 switches to use the same forwarding tree of SDN nodes. Although these works propagate SDN programmability to legacy devices, none of them tackles the problem of sub 50ms failure detection or rerouting in hybrid L3 networks.

-**Failure Detection time** Multiple works were conducted to decrease the failure detection time such as [8, 16, 9, 17]. To detect a network failure most of them require on active probing i.e. a legacy node/SDN switch transmits a state packet to its neighbors to check the viability of the link connecting them [23, 16, 17] or the viability of the path between the nodes [17]. When this state packet is not received from the neighbor for more than a predefined fixed *timeout* duration the link or path connecting the node to its neighbor is declared down. By default, legacy routing protocols such as OSPF [23] detect failures after seconds, SDN devices however, detect link failures after around 110 ms. To decrease the default link or path failure detection, Bidirectional Forwarding Detection (BFD) [16] and BFD-multihop [17] were proposed. Both of these methodologies also depend on the active link/path probing. However, a session is established between the neighboring nodes across a physical link, tunnel or path. BFD, allows the detection of link failure in around 40ms [30]. However, as stated in [12], BFD packets are generated by the forwarding devices themselves *i.e.* the legacy node and SDN switch. Add to that the minimal interval between two consecutive BFD transmission on SDN hardware are proprietary dependent where some switches can support a minimal of 1 second minimum interval [12]. Nonetheless, very low BFD transmission interval increases the CPU overhead of the hardware SDN switch [12]. In PRoPHYS, we provide a maximum of 25ms of downtime using active or passive probing without increasing the CPU of the forwarding devices or being limited to the features provided by the SDN switch vendor.

A different failure detection methodology [9] predicts link failures based on opportunistic scheduling and power signal measurements. In [9], the authors mainly measure the electric signal on every port and then based on the disruptions of this electrical signal, they estimate that a network failure occurred. This method, however, is limited only to direct link failure.

-**Rerouting Time** To decrease the rerouting time, two main methodologies exist passive rerouting and active rerouting. Passive rerouting pre-installs backup routes or rules in the network such as [31, 4] and provides the least rerouting time. In this case, once the link failure has been detected, legacy node can use the backup route for example MPLS-FRR [1] extension of RSVP-TE for LSP tunnels, labels the traffic, and reroutes it using pre-established MPLS tunnels. OpenFlow nodes can use the backup rules through the notion of FAST-FAILOVER group rules. The authors of [31] showed that they can obtain a minimum of 99.7% reachability with k-link failures. This methodology

however, can not scale for large networks as it would require the installation of hundreds or even thousands of backup routes or rules in the limited (and expensive) TCAM memory. Following the same concept, to avoid losses in the case of link failures in hybrid networks, [4] introduced pre-setup backup tunnels from legacy routers towards SDN routers, and SDN nodes reroute traffic through non damaged paths.

As for active rerouting, the router/controller recalculates the route needed to reach the destination based on the information received from the update messages such as [21]-PRoPHYS relies on active rerouting with tunnels. RSDN [21] leverages the recursiveness of ISP networks, to create a set of aggregate routers called Logical Crossbars (LXBs). Route computation are done on LXBs for each level, and a summary of the results are sent to the parent LXB. When a node fails (e.g., node b) on a path (e.g., the path a-b-c-d), the LXB controller hierarchy will be able to virtualize b's routing table in a. RSDN will then compute recursively the paths to destinations.

In PRoPHYS we use active rerouting were we install previously the tunnels that could be used, but we notify the switch of using them when necessary. We chose this technique as passive rerouting cannot scale in the ISP due to the enormous number of routes that should be installed to cover all combinations of link failure which prohibits scalability.

Discussion

In this section, we discuss two issues: (i) the combination of the PortStat and PktOut methods, and (ii) an alternative approach to tunnelling and force OSPF nodes to re-converge faster

7 Combining PortStat and Pktut

From the results obtained in Section 5, we notice that the passive PortStats method outperforms the PktOut method when the connection speed increases, but at the cost of exhibiting false positives. On the other hand, PktOut outperforms PortStats for low connection speeds, and maintains a stable performance. Another difference that we see between the two methods is that the SDDP packet sent by the controller across a network segment connecting two SDN nodes only enables the measurement of the performance of one shortest path between the two SDN nodes. If multiple shortest paths exist between the two nodes (e.g., in the case of load balancing), PktOut will not be able to test the viability of all existing paths simultaneously, which could increase the detection delay. However, using PortStats enables to test all paths used by real traffic simultaneously, i.e., the passive approach measures a wider variety of paths across the legacy islands.

It thus sounds natural to use a combination of both methods. In order to decrease the number of false positives, we can use 3 possible combinations of PortStats and PktOut:

(*i*) PortStats and PktOut in parallel. In this scenario both methods are running in parallel and whenever a method detects failures, it validates/rejects its results by comparing them with the results of the other method.

(*ii*) PortStats then PktOut. PortStats is used, and when it detects a segment failure, it launches PktOut to validate the result.

(*iii*) **PktOut then PortStats.** PktOut is used, and when it detects a segment failure, it launches PortStats to validate the result.

The parallel approach (case (*i*)) would allow to validate the detection of segments failure with the minimum detection interval. Moreover, it allows to adapt `thresh`, `thresh_count`, and the monitoring interval variables of PortStats based on the estimated delay information that are obtained by PktOut. The main advantage of one method followed by the other (cases (*ii*) and (*iii*)), is the limited number of commands sent at the same time by the controller to the switch. Indeed, the parallel method induces a higher pressure on the SDN control plane as illustrated in Figure 9, where we observe that the parallel approach reaches the 200 event/s that a typical hardware SDN can sustain.

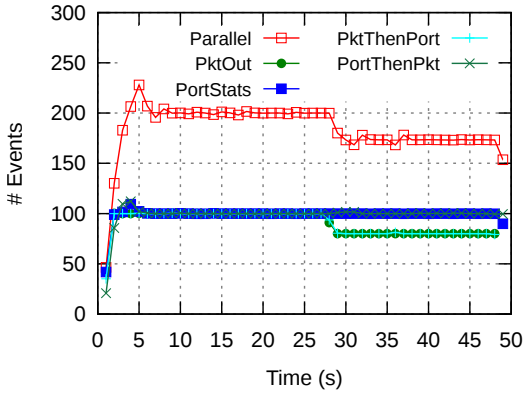


Figure 9: Total number of events triggered per second over every switch in the network.

7.1 Forcing OSPF routers to re-converge faster

In Section 4, we have described various methods that can be used in order to reroute the traffic over OSPF islands while minimizing the impact on the end user quality of experience. In our tests, we considered rerouting traffic using tunnels, while launching the rerouting procedure in parallel, up to the moment where OSPF islands converge to the correct network topology. Installing a full mesh of tunnels between the SDN nodes would provide the best performance, compared to navigating in an overlay of tunnels on multiple consecutive SDN nodes, because the performance degrades as the number of tunnels increases. Unfortunately, installing a full mesh of tunnels in a large scale network can be difficult due to the limitations on the number of tunnels that can be supported by SDN nodes (tunneling increases the CPU consumption).

Another realistic solution that is compatible with current technologies and allows to reroute traffic while avoid conflicting views between SDN and legacy routers is to force legacy routers to reroute to the new segment. The controller can do this by generating routing update messages (e.g., an *LS_Update* with OSPF) that will be forwarded to SDN nodes, which will spread to their OSPF neighbors. This will force legacy routers to recalculate their routing table, leading to a fast update of the network topology.

In this scenario, the SDN nodes can then transmit the traffic on the new path in one of the following ways: (*i*) directly after the update message has been transmitted from the involved SDN nodes to the legacy islands; or (*ii*) after a few milliseconds delay, to enable the non-legacy routers between the SDN nodes to converge to the correct path.

In case the traffic is sent directly after transmitting the update, small losses can be observed due to the delay resulting from the relative speeds of processing update packets, and forwarding traffic. For instance, we have observed that an OSPF router needs around 0.13ms to process and retransmit an LS Update with a link failure update. This implies that, if the maximum distance between a legacy router and any SDN switch is 5 hops, any packet might still follow the failed link during 0.65ms. However, a 5 hops maximum distance from any SDN node implies also that there are 10 legacy routers in the shortest path between 2 SDN nodes, which we believe, is a very long, rare, path length inside an Autonomous System. In contrast, traffic is directly forwarded, at line rate.

To achieve 0 packet losses, we propose to insert a new rule into the SDN node to forward the packet, presumably going through the failed island, towards a nearby dedicated server (e.g., located in the same PoP), which will buffer the packets for a period of time equal to the expected convergence time after the SDN nodes releases the update messages (e.g. 0.65ms assuming a 0ms propagation delay between the SDN node and the server and 10 hops between the upstream and the downstream SDN nodes). This proposition has not been integrated yet in our PRoPHYS prototype, but has been already successfully tested in a server leveraging a NFQUEUE Netfilter [24].

8 Conclusion

In this article, we presented the PRoPHYS solution that can decrease the down-time from 50ms to 25ms in hybrid SDN/OSPF networks. This is achieved by using the SDN controller to quickly detect failures within the SDN and non SDN part (OSPF islands) of the network and also by carefully interacting with OSPF routers during their re-convergence period using tunnels.

We proposed two techniques to detect failures, PortStat and PktOut, and evaluated their relative merits in terms of detection time and false positive rate for key scenarios involving UDP or TCP traffic. We also discussed various options or requirements to obtain the minimum detection interval.

As a future work, we plan to thoroughly evaluate the combinations of the PortStat and PktOut methods in large scale hybrid ISP network topologies using real traffic.

References

[1] Alia Atlas, George Swallow, and Ping Pan. Fast Reroute Extensions to RSVP-TE for LSP Tunnels. RFC 4090, May 2005.

[2] Ed. B. Davie and J. Gross. A Stateless Transport Tunneling Protocol for Network Virtualization(STT). IETF draft-davie-stt-01, March 2012.

[3] Deborah Brungard, Malcolm Betts, Satoshi Ueno, Ben Niven-Jenkins, and Nurit Sprecher. Requirements of an MPLS Transport Profile. RFC 5654, September 2009.

[4] Cing-Yu Chu, Kang Xi, Min Luo, and H Jonathan Chao. Congestion-aware single link failure recovery in hybrid SDN networks. In *IEEE INFOCOM*, 2015.

[5] Cisco. Cisco visual networking index: Forecast and methodology, 2015–2020. Tech. Rep., June 2016.

[6] Paul Congdon. Link layer discovery protocol. RFC 2922, July 2002.

[7] Dino Farinacci, Stanley P. Hanks, David Meyer, and Paul S. Traina. Generic Routing Encapsulation (GRE). RFC 2784, March 2000.

[8] Pierre Francois, Clarence Filsfils, John Evans, and Olivier Bonaventure. Achieving subsecond IGP convergence in large IP networks. *ACM SIGCOMM CCR*, 35(3):35–44, 2005.

[9] B. Hamzeh. Network failure detection and prediction using signal measurements. US Patent 9,100,339, August 4 2015.

[10] Brandon Heller, Rob Sherwood, and Nick McKeown. The controller placement problem. In *Hot Topics in SDN*, pages 7–12. ACM, 2012.

[11] Luuk Hendriks, Ricardo de O Schmidt, Ramin Sadre, Jeronimo A Bezerra, and Aiko Pras. Assessing the quality of flow measurements from openflow devices. *International Workshop on Traffic Monitoring and Analysis (TMA)*, 2016.

[12] Hewlett Packard Enterprise Development LP. Bidirectional forwarding detection (BFD), 2016.

[13] David Ke Hong, Yadi Ma, Sujata Banerjee, and Z. Morley Mao. Incremental deployment of SDN in hybrid enterprise and ISP networks. In *ACM Symposium on SDN Research*, 2016.

[14] Ryan Izard. Floodlight documentation, march 2015.

[15] Cheng Jin, Cristian Lumezanu, Qiang Xu, Zhi-Li Zhang, and Guofei Jiang. Telekinesis: Controlling Legacy Switch Routing with OpenFlow in Hybrid Networks. In *ACM SIGCOMM Symposium on Software Defined Networking Research*, 2015.

[16] Dave Katz and David Ward. Bidirectional Forwarding Detection (BFD). RFC 5880, June 2010.

[17] Dave Katz and David Ward. Bidirectional Forwarding Detection (BFD) for Multihop Paths. RFC 5883, June 2010.

[18] Diego Kreutz, Fernando M. V. Ramos, Paulo Veríssimo, Christian Esteve Rothenberg, Siamak Azodolmolky, and Steve Uhlig. Software-Defined Networking: A Comprehensive Survey. *Proceedings of the IEEE*, 103(1), 2015.

[19] Dan Levin, Marco Canini, Stefan Schmid, Fabian Schaffert, Anja Feldmann, et al. Panopticon: Reaping the Benefits of Incremental SDN Deployment in Enterprise Networks. In *USENIX Annual Technical Conference*, 2014.

[20] ACIRI Mark Handley and Jon Crowcroft. Internet multicast today. *The Internet Protocol Journal*, 2(4), 1999.

[21] James McCauley, Zhi Liu, Aurojit Panda, Teemu Koponen, Barath Raghavan, Jennifer Rexford, and Scott Shenker. Recursive SDN for carrier networks. *ACM SIGCOMM CCR*, 46(4), 2016.

[22] Mininet Team. Mininet. http://mininet.org/, 2016.

[23] John T. Moy. OSPF Version 2. RFC 2328, April 1998.

[24] Netfilter. Netfilter.

[25] Open Network Foundation Solution Brief. SDN Migration Considerations and Use Cases. Tech. Rep. #ONF TR - 506, November 2014.

[26] Ponemon Instituter. Cost of data center outages. Tech. Rep., January 2016.

[27] Matt Sargent, Jerry Chu, Vern Paxson, and Mark Allman. Computing TCP Retransmission Timer. RFC 6298, June 2011.

[28] Tom Scholl. BFD: Does it work and is it worth it? NANOG 45.

[29] Mike Shand and Stewart Bryant. IP Fast Reroute Framework. RFC 5714, January 2010.

[30] Sachin Sharma, Dimitri Staessens, Didier Colle, Mario Pickavet, and Piet Demeester. Openflow: Meeting carrier-grade recovery requirements. *Elsevier Computer Communications*, 36(6):656–665, 2013.

[31] Baohua Yang, Junda Liu, Scott Shenker, Jun Li, and Kai Zheng. Keep forwarding: Towards k-link failure resilient routing. In *IEEE INFOCOM 2014*, pages 1617–1625, 2014.

[32] Zuse-Institute Berlin. Problem germany50–d-b-l-n-c-a-n-n). http://sndlib.zib.de/home.action.