OpenFlow-Based Segment Protection in Ethernet Networks

Andrea Sgambelluri, Alessio Giorgetti, Filippo Cugini,
Francesco Paolucci, and Piero Castoldi

***Abstract*—Metro and carrier-grade Ethernet networks, as well as industrial area networks and specific local area networks (LANs), have to guarantee fast resiliency upon network failure. However, the current OpenFlow architecture, originally designed for LANs, does not include effective mechanisms for fast resiliency. In this paper, the OpenFlow architecture is enhanced to support segment protection in Ethernet-based networks. Novel mechanisms have been specifically introduced to maintain working and backup flows at different priorities and to guarantee effective network resource utilization when the failed link is recovered. Emulation and experimental demonstration implementation results show that the proposed architecture avoids both the utilization of a full-state controller and the intervention of the controller upon failure, thus guaranteeing a recovery time only due to the failure detection time, i.e., a few tens of milliseconds within the considered scenario.**

***Index Terms*—Ethernet; OpenFlow; Recovery; Segment protection; Software-defined networking (SDN).**

I. Introduction

Software-defined networks are emerging as an attractive solution to permit effective separation of data and control planes while guaranteeing a centralized network intelligence to control the network (i.e., the controller). OpenFlow is an example of a software-defined network in which the controller manages the switching elements (e.g., Ethernet switches) by installing *flow entries* within their flow table [1]. Communications between the controller and the switches use the OpenFlow protocol through a secure channel. Each flow entry consists of a *flow-match* composed of a set of fields to match the incoming packets, an *action* to define how to process matching packets (e.g., forwarding on a specific output port), and several *counters* used for collecting flow statistics. Other information can also be associated with each entry, such as *priority level* and the two timers *hard timeout* and *idle timeout*. When a switch receives a packet not matching any of the installed entries, that packet is sent to the controller using the secure channel. Upon reception, the controller decides how to handle the received packets, e.g., it computes a path and installs a flow entry in each traversed switch to deliver the matching packets to the proper destination.

OpenFlow was originally designed at Stanford University to be used in local and campus area networks [1,2]. However, because of its centralized control approach it is currently also under consideration in other scenarios such as carrier-grade networks [3], optical transport networks [4–8], and industrial area networks [9]. Specifically, transport network carriers tend to prefer centralized control because it is simpler, more manageable, and easier to migrate from the current network management system architecture with respect to the current standard distributed control plane of transport networks [10], i.e., generalized multi-protocol label switching (GMPLS).

In all the aforementioned scenarios, network reliability is a fundamental feature. Indeed, in carrier-grade and optical transport networks, a single failure can disrupt a huge amount of traffic, implying a service degradation to a great number of users. In these networks the target recovery time is 50 ms as guaranteed by the legacy SONET/SDH networks [11,12]. Similarly, in industrial area networks the recovery time cannot exceed the typical *grace time* of industrial plants (e.g., 20 ms for time critical automation systems, 200 ms for general automation systems) [13,14]. Therefore, in all of these cases, fast recovery of network failures is strictly required. However, fast recovery is not a key feature in local area networks (LANs), and the current OpenFlow architecture needs several seconds to recover from a network failure [15].

In order to achieve lower recovery times, this paper proposes an OpenFlow-based segment protection (OSP) scheme enabling fast recovery in case of single link or interface failure in Ethernet-based networks. The concepts behind the OSP scheme can be applied, with slight modifications, in OpenFlow-based optical transport networks [4–8]. Therefore, this work also represents a first step toward the introduction of OpenFlow-based fast recovery in optical transport networks.

The proposed architecture relies on preplanned backup paths guaranteeing that, upon failure occurrence, recovery is locally performed by the switch attached to the failed link, thus minimizing the recovery time. To support backup paths, several enhancements to the OpenFlow architecture are proposed. Indeed, in addition to the auto-reject mechanism, the configuration of alternative flow entries with

Manuscript received March 29, 2013; revised July 12, 2013; accepted July 18, 2013; published August 26, 2013 (Doc. ID 187905).

A. Sgambelluri (e-mail: a.sgambelluri@sssup.it), A. Giorgetti, F. Paolucci, and P. Castoldi are with the Scuola Superiore Sant'Anna, Pisa, Italy.

F. Cugini is with CNIT, Pisa, Italy.

http://dx.doi.org/10.1364/JOCN.5.001066

different priority levels, a renewal mechanism, relying on a novel *renew* packet, is introduced to avoid the expiration of the entries required to support the backup paths. In addition, a novel mechanism is provided for reverting active flows to the original working path once the failure has been physically repaired, thus guaranteeing efficient network resource utilization.

The proposed OSP scheme has been validated on ring topologies through emulation and experimental demonstration of real implementation, both achieved by extending OpenFlow version 1.0 [16]. The obtained results show that the proposed scheme avoids controller scalability issues upon failure occurrence and guarantees a recovery time only dependent on the failure detection time (i.e., a few tens of milliseconds in the considered testbed).

II. Previous Work

The authors of [15] and [17] designed a restoration scheme for OpenFlow carrier-grade Ethernet networks. In the proposed scheme, the switch connected to the disrupted link directly notifies the controller about the topology change. Upon notification, a *full-state* controller (i.e., storing the complete path of all the flows established in the network) identifies the disrupted flows, computes the backup paths, and updates the data plane flow tables considering the failure. Obtained results show recovery times around 200 ms. However, the authors recognize that, in big networks, the full-state controller could be overloaded by recovery requests the full-state architecture of the controller may introduce, thus introducing significant scalability issues.

In [18] two different recovery mechanisms are presented, i.e., restoration and path protection. In the case of restoration, similarly to in [15,17], the controller reacts to the failure notification by deleting affected flow entries, computing backup paths, and installing the new required entries. In the case of path protection, backup paths are precomputed and installed using the *fast-failover groups* functionality of OpenFlow specification 1.1, and working path liveness is monitored by the ingress switch with a mechanism similar to the one proposed in [7]. In the case of failure, emulation results show that the protection solution achieves the recovery process in around 50 ms.

The work in [19] considers OpenFlow in IP networks. The proposed scheme is similar to that of [15,17], considering a full-state controller that is notified by the switch upon link failure occurrence. In this case emulation results provide a failure recovery time in the range of 200–300 ms.

The work in [7] proposes a monitoring mechanism enabling fast failure discovery in transport networks based on OpenFlow. In the proposed mechanism, each established flow is monitored by sending frequent probe messages. Failures are quickly discovered; however, the controller is then involved for the computation of the backup paths. Thus, the overall recovery time may still suffer from long delays and relevant scalability issues.

In [20], a data plane mechanism is proposed to notify, upon failure, all the switches that are sending traffic through the disrupted link. Upon notification the switches stop these traffic flows to avoid waste of bandwidth. The actual recovery is then performed by the controller relying on the standard OpenFlow mechanism.

The authors of [21] propose a mechanism enabling the utilization of a backup controller in an OpenFlow-based network to avoid the problem of having a single point of failure, which is a typical weak point of centralized approaches.

With respect to the restoration schemes proposed in [15,17–19], the OSP scheme does not involve the controller upon failure occurrence and does not require a full-state controller. Moreover, with respect to the path protection scheme proposed in [18], OSP implements segment protection; thus it locally switches the traffic on the backup path and does not require a per-flow monitoring mechanism. For the aforementioned reasons the OSP scheme is expected to be more scalable and to achieve lower recovery time with respect to existing schemes.

III. OpenFlow-Based Architecture Enabling Segment Protection

In OpenFlow switches each flow entry is installed with two associated timers, i.e., hard timeout and idle timeout. The entry is deleted by the switch upon expiration of one of the two timers. The hard timeout is never refreshed and is used to set the maximum duration of each entry. Conversely, the idle timeout is refreshed every time a packet matches the associated entry; it expires owing to lack of activity.

Referring to OpenFlow Switch Specification 1.0 [16], failure recovery operates as follows. When the switch detects a failure, it notifies the controller with a port status message. Thus, the controller updates the network topology and the successive path computations are performed using the updated topology. However, the switch does not delete the entries using the failed link. Therefore, established flows are not actually recovered till one of the aforementioned timers expires. Since these timers are usually set to several seconds, current OpenFlow implementations do not guarantee fast recovery [15].

Therefore, as a first step, an *auto-reject* mechanism should be implemented to promptly delete the entries using the failed link. Specifically, with the auto-reject mechanism, upon failure occurrence, the two switches connected to the failed link delete all their flow entries having the input or output port on the failed link. Moreover, upon reception of a new entry to be installed, an auto-reject-enabled switch checks the status of the ports used by the entry; when one of the ports uses a failed link, the entry is not installed.

However, extending OpenFlow with only the auto-reject mechanism does not guarantee fast recovery; indeed, failures would be handled as follows. Switches apply auto-reject and send port status to the controller. Because of auto-reject, subsequent packets of disrupted flows do not match any entry and are sent to the controller. While

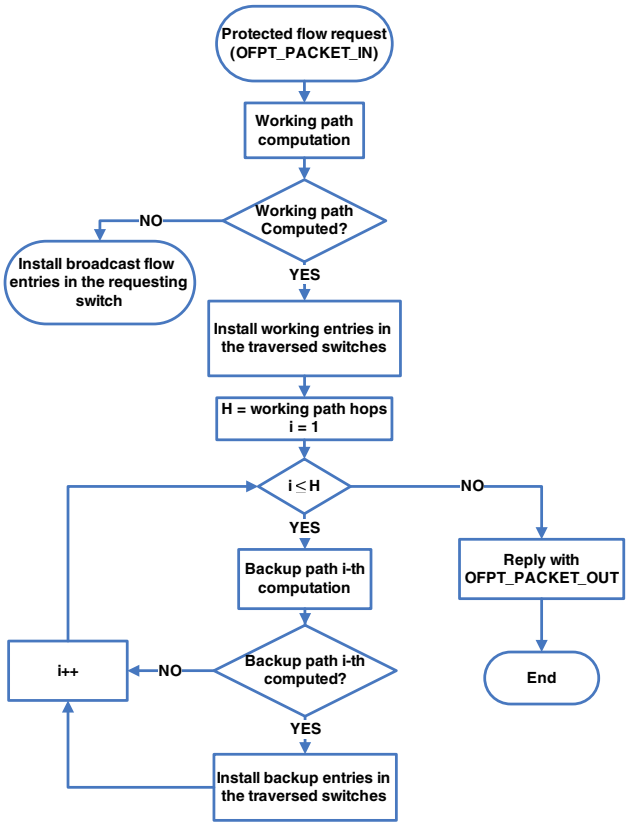

Fig. 1. Flowchart of the *Recovery* module at the controller. This flowchart refers to the case of a flow request protected against link and interface failures.

the controller is updating the topology, it keeps replying with flow entries using the disrupted link, which are refused by auto-reject. Only after topology update is a feasible path computed and proper entries installed in all the involved switches. This procedure requires several seconds. Thus, besides auto-reject, a set of backup entries have to be preconfigured for enabling fast recovery.

The rest of this section describes the architecture enabling the proposed OSP scheme. The flow entries required for each protected data flow are detailed in Subsection III.A; the mechanism needed for avoiding the expiration of the installed entries is described in Subsection III.B. Finally, Subsection III.C summarizes the actions taken in case of link failure and recovery.

A. Flow Entries Installation

Figure 1 shows the flowchart of the *Recovery* module that we have implemented at the controller, for the case of a flow request protected against single link and interface failures. Specifically, upon request of a protected data flow between the hosts (*Src-Dst*), the controller computes and installs the entries for supporting the working path (i.e., *working entries*). Then, for providing protection against link and interface failures only, a backup path is computed considering a failure in each link traversed by the working path. Conversely, if protection is also required against single node failures, a backup path is computed considering a failure in each intermediate node traversed by the working path. After backup path computation the required *backup entries* are installed to enable the switching of packets along the backup paths. During failure-free operation, data are routed along the working path. If a failure disrupts the working path, the flow is deviated by the switch attached to the failed link so that it will reach *Dst* along the backup path (see Figs. 2–4).

Working and backup entries are installed with different priorities. High-priority levels are used for working entries: H_i in the ingress switch, H_t in transit switches, and H_e in the egress switch. Low-priority levels are used for backup entries: L_i in the ingress switch and in switches where a backup path diverges from the working path (e.g., switch B in Fig. 2), L_t in transit switches, and L_e in the egress switch). Each received packet is forwarded considering the currently installed matching entry with the highest priority.

Figure 2 summarizes the flow entries required in the illustrated mesh topology when a single protected data flow is configured between the host pair (*Src-Dst*). For the sake

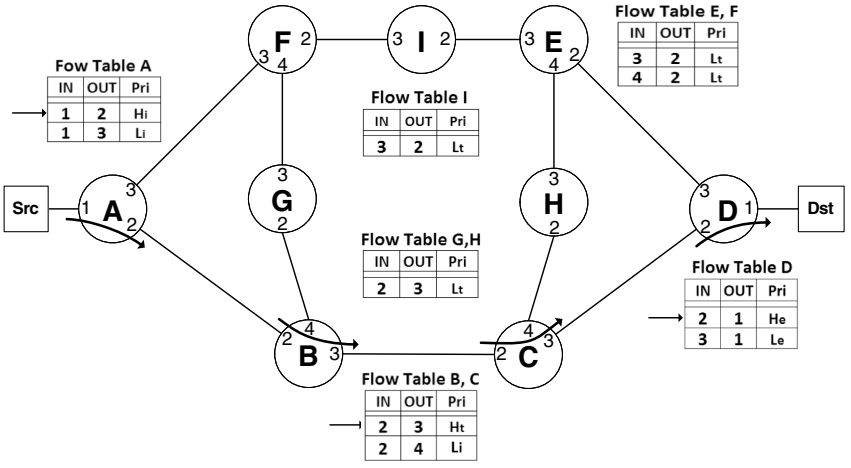

Fig. 2. Protected traffic flow in a mesh network. Only the flow entries related to host pair (*Src-Dst*) are shown.

of clarity, the flow entries are shown in Fig. 2, just including the triplet composed of the input port (IN), outport port (OUT), and priority level (Pri). However, other parameters [i.e., *Src* and *Dst* media access control (MAC) addresses] are used for flow matching.

Specifically, in Fig. 2 the working path is *A*–*B*–*C*–*D*. Backup paths are configured for multiple points of failure along the working path. At ingress switch *A*, the flow table is configured with one working entry $(1, 2, H_i)$ and one backup entry $(1, 3, L_i)$. The working path entry enables the packet switching from the host *Src* toward switch *B*. The backup path entry is configured for the case of failure of the adjacent link *A*–*B* to enable the forwarding of packets toward switch *F*.

The transit switches *B* and *C* are configured with a working entry $(2, 3, H_t)$ and a backup entry $(2, 4, L_i)$. The backup entry is configured for a case of a failure of the adjacent link (*B*–*C* for switch *B*, *C*–*D* for switch *C*). For switch *B*, this causes traffic to be sent to switch *G*, and for switch *C* this causes traffic to be sent to switch *H*. Egress switch *D* is configured as a working and a backup entry in the same way as previously described. The other transit switches along the backup paths are configured with one or more backup entries, e.g., $(2, 3, L_t)$ in the switches *G* and *H*.

Figures 3 and 4 show the backup paths used in the case of a failure occurring in the working path along link *A*–*B* and link *C*–*D*, respectively. Referring to Fig. 4, the failure of link *C*–*D* is detected by switches *C* and *D*; at switch *C* the auto-reject mechanism removes the flow entry $(2, 3, H_t)$. This leaves a flow entry for the configured backup path $(2, 4, L_i)$. The resulting backup path is *A*–*B*–*C*–*H*–*E*–*D*.

B. Backup Flow Entries Renewal

Since, in the proposed architecture, no packets are routed along the backup paths during failure-free operation, the expiration of the idle timeout may determine the deletion of backup entries. Therefore, a mechanism is used to avoid the deletion of backup entries related to active working flows, without involving either the controller or the hosts *Src* and *Dst*.

First, for each flow, the refresh of working entries automatically determines the refresh of all the backup entries associated with the same flow. With reference to Fig. 2, this will guarantee the liveness of all backup entries in switches *A*, *B*, *C*, and *D*.

Second, a specifically designed *renew* packet is built and periodically sent by each switch on the output port of entries with L_i priority (e.g., switch *B* in Fig. 2). The renew packet fields are set to match the considered flow. Moreover, a specific field is included so that the egress switch can distinguish renew packets from data packets. Since the generation rate of renew packets depends on the considered idle timeout, they use a negligible amount of bandwidth. In Fig. 2, the ingress switch *A* periodically sends renew packets along the backup path *A*–*F*–*I*–*E*–*D*, these frames do not reach the host *Dst*, as they are dropped by the egress switch *C*. In this way, the backup entries in switches *F*, *I*, and *E* are preserved. Finally, switches *B* and *C* periodically send renew packets on port 4 so that backup entries in switches *G* and *H* are also preserved.

C. Failure Recovery and Reversion

If the failure affects a working path, auto-reject is triggered at the switches attached to the failed link. After auto-reject of the working entry, the related backup flow entry is automatically used for switching the packets along the backup path. Data packets are lost only in the period of time between the failure and the auto-reject (i.e., *switch-over* time). Indeed, during this period packets are forwarded on the failed link. In a similar way, if the failure affects a backup path, the auto-reject deletes the related backup entry. In this case, data packets are not affected; however, renew packets periodically sent along the backup path arrive at the switch attached to the failed link. These renew packets do not match any entries and are therefore forwarded to the controller using `OFP_PACKET_IN` packets. To limit requests to the controller, these packets are handled as described in Subsection IV.C.

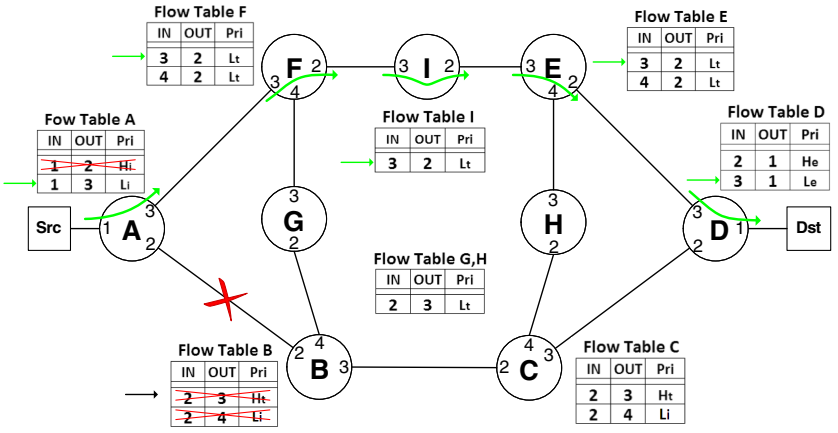

Fig. 3. Actual backup path used upon failure of link *A*–*B*.

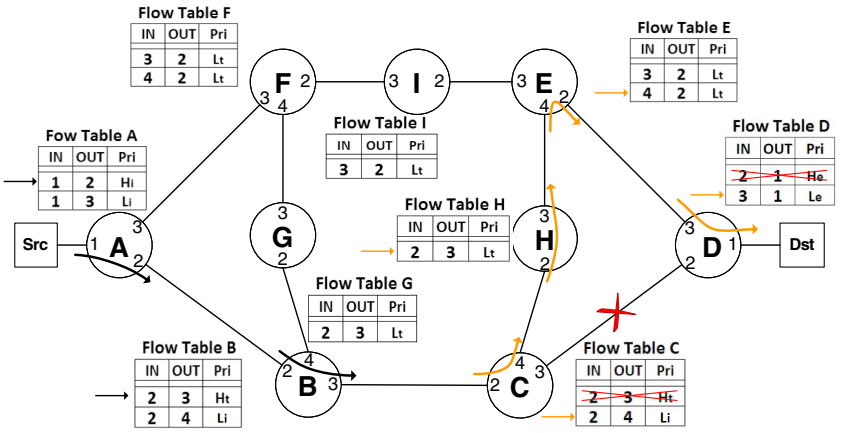

Fig. 4. Actual backup path used upon failure of link *C–D*. Traffic is wrapped at node *C*.

When the link failure is physically repaired, the attached switches notify the controller about the topology change, sending a port status message. Then, for reverting the routing of the active flows to the original working path as before the failure, a `OFP_FLOW_RESTORE` packet is sent, for each installed flow match, to the controller and handled as described in Subsection IV.C. For instance, working entries in the switches downstream to a failure occurred along the working path may expire after the failure. This rerouting procedure involves the controller and can therefore require several seconds; however, this is not an issue because, in the case of link recovery, data traffic is not disrupted.

IV. Implementation

The proposed OSP scheme has been implemented by extending the OpenFlow Switch Specification 1.0 [16,22]. More recent specifications include additional features (i.e., fast-failover groups) that could further facilitate the OSP implementation. However, stable implementations of these specifications were not available at the time of architecture design. This section describes the extensions to the OpenFlow protocol, to the OpenFlow switch, and to the OpenFlow controller, which have been implemented to enable the OSP scheme.

A. OpenFlow Protocol

The standard OpenFlow version 1.0 protocol has been extended with the novel `OFPT_FLOW_RESTORE` packet. Its fields include the standard OpenFlow `header` object, the `match` object specifying the flow-match, the `cookie` object, and the `priority` object as depicted in Fig. 5.

B. OpenFlow Switch

The OpenFlow switch implementation is based on Open vSwitch version 1.1.1, which is compliant with OpenFlow Switch Specification 1.0 [16,22]. Moreover, the following novel features have been implemented.

1) Flow Auto-Reject: As described in Section III, a switch detecting a failure on a local interface deletes all the flow entries having the failed interface as an input or output port. Moreover, a new flow entry is installed only if the output port is working correctly.

2) Flow Renewal: With reference to Subsection III.B, renew packets have been implemented using a specifically designed Ethernet frame (i.e., `ETH_RENEW`). In particular, `ETH_RENEW` frames are of 64 byte size, source and destination MAC addresses are set to match the considered flow, and the EtherType field is set to the specific value 0 × 88dd. For each flow, one `ETH_RENEW` frame is sent every 20 s by backup path ingress switches. A switch is recognized to be a backup path ingress of a particular flow if its flow table includes an entry with priority L_i. A switch is recognized to be the egress of a particular flow if its flow table includes an entry with priority H_e and L_e; see Fig. 2.

3) Flow Restore: As mentioned in Subsection III.C, the `OFPT_FLOW_RESTORE` packet is sent 20 s after a link recovery has been detected on a local interface. This 20 s delay is empirically estimated so that, upon failure, the controller receives the port status messages from the switches

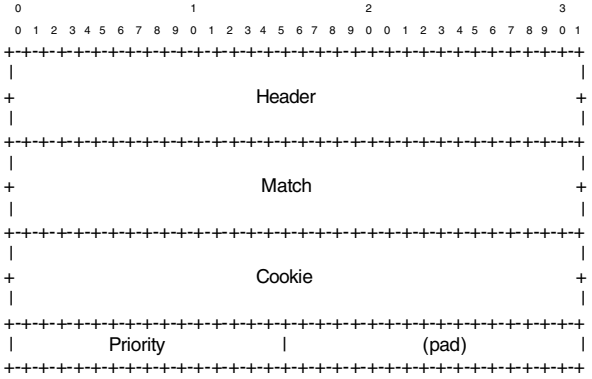

Fig. 5. `OFPT_FLOW_RESTORE` packet format.

attached to the recovered link; then it has time to update the network topology before the reception of the OFPT_FLOW_RESTORE.

C. OpenFlow Controller

The OpenFlow controller implementation is based on the NOX controller version 0.9.1-full-beta [23]. The novel *Recovery* module has been implemented to process the packets arriving from the switches and to install the required working and backup entries with the proper priorities, according to the flowchart of Fig. 1. Moreover, the existing *Routing* module has been slightly extended to enable the computation of backup paths. In the considered scenario, both working and backup entries are installed using a MAC layer flow-match: input port, source MAC addresses, and destination MAC address; see Subsection III.A. Idle timeout of 120 s, and hard timeout 0 s are used; i.e., entries are deleted only upon idle timeout expiration.

The OFP_PACKET_IN packets are handled depending on the EtherType field:

1) *Packet With EtherType* 0×800 *(IPv4):* This is the request for a new data flow. All data flows are currently installed with protection; however, hosts can be allowed to require protected and unprotected data flows, e.g., setting specific flags in the IP header.
2) *Packet With EtherType* $0 \times 88dd$*:* This is a renew packet sent to the controller due to a failure occurring along the backup path (see Subsection III.C). If the failure has been recovered, the backup paths are recomputed and the related entries are reinstalled in the traversed switches; otherwise a *permanent* null entry is installed with a low-priority value (i.e., both hard timeout and idle timeout set to 0). In such a way the next renew packets are dropped at the switch to avoid the unnecessary overhead of the controller; when the failure is recovered, the null flow entry is replaced with a new valid backup flow entry.

The OFP_FLOW_RESTORE packets are handled as follows. The controller extracts the match, computes working and backup paths, and installs the proper working and backup entries in the involved switches.

V. Performance Evaluation

The proposed OSP scheme has been tested through both emulation and experimental demonstration of real implementation. In both cases, a ring topology composed of $N = 5$ switches has been considered. Indeed, the ring topology is the most common in industrial area networks [14,24], and it represents the worst case for segment protection schemes in terms of both recovery time and network resource utilization. The ring topology is a critical case for the proposed OSP scheme because backup paths typically can partially overlap the working path. Specifically, Fig. 6 illustrates the flow entries required in the considered ring topology when a single protected data flow is configured between the host pair (*Src-Dst*). In the case of the failure of link *B–C* the traffic is wrapped back to port 2 by switch *B*; thus the backup path is *A–B–A–E–D–C*, where link *A–B* is traversed twice.

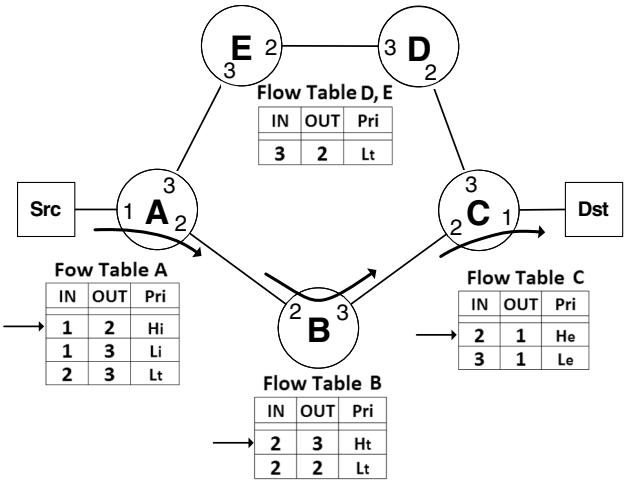

Fig. 6. Protected traffic flow in the test ring network. Only the flow entries related to host pair (*Src-Dst*) are shown.

In the emulation environment, two servers have been employed. The first server (i.e., Intel Pentium 4 CPU 3.00 GHz, 1 GB RAM, Ubuntu 11.10 kernel 3.0.0-16-generic) acts as the OpenFlow controller, running the NOX version 0.9.1-full-beta, extended with the described recovery functionalities. The second server (i.e., Intel Core i5-2400 CPU 3.10 GHz, 4 GB RAM, Ubuntu 11.04 kernel 2.6.38-8-generic) emulates the network. In particular, the Mininet tool is used for emulating rings composed of N OpenFlow-enabled Ethernet switches, extended with the new recovery functionalities, and M hosts connected to each switch [25]. Each emulated OpenFlow switch is configured with two interfaces for the ring links; one interface is for the connection to the controller, and the other interfaces are used for connecting the hosts.

In the experimental demonstration of real implementation, the server hosting the emulating network was replaced with a real OpenFlow-based Ethernet ring network composed of $N = 5$ switches. The five switches were implemented in five personal computers (i.e., Intel Core 4 CPU 2.40 GHz, 2 GB RAM, Ubuntu 10.04 kernel 2.6.32-25-generic) using an extended release of Open vSwitch version 1.1.1 to support the proposed recovery mechanism. Each computer was equipped with a network interface card (Intel Quad Port server adapter PCI-Express) providing four Ethernet interfaces, i.e., two for the ring links, one for the controller, and one for the connection to an end host ($M = 1$).

Subsections V.A and V.B report the results obtained in the system performance tests, considering the required number of flow entries and the time needed to perform the recovery of the active traffic in ring topologies with different N and M values.

A. Number of Entries

The emulated environment has been used to assess the required number of per-switch flow entries. Indeed, as described above, to speed up the switchover time, the OSP scheme performs local traffic protection without involving the controller in the recovery process. However, considering working and backup paths, a significant number of flow entries have to be installed in each network switch to support the OSP scheme.

Figure 7 illustrates the required entries per switch, considering the worst case in which a protected flow is active between each couple of hosts connected on different switches (i.e., *any-to-any* communication). N values higher than 13 are not supported by the server running Mininet. Equations (1) and (2), in accordance with Fig. 7, respectively detail the amount of working entries W and backup entries B in each switch of a ring network as a function of N and M, in the any-to-any communication worst case:

$$W = 2M^2(N-1) + 2M^2 \sum_{i=0}^{\frac{N+1}{2}-2} i, \tag{1}$$

$$B = M^2\left[N^2 + N - 4 + 2u(N-5)\sum_{i=0}^{\frac{N-1}{2}-2} i\right]. \tag{2}$$

Since in the current field programmable gate-array implementation of OpenFlow switches, up to 64,000 contemporarily active entries can be easily supported [2]; the results illustrated in Fig. 7 show that the OSP scheme can scale up to rings composed of several tens of switches and hosts.

Figure 8 details the trend of working, backup, and total entries in each switch for $M = 1$. The number of required backup entries grows faster than the number of required working entries with the number of network switches N.

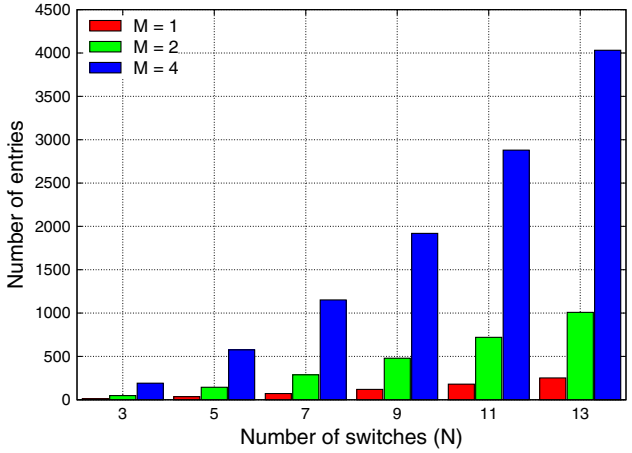

Fig. 7. Flow entries per switch with $M \in \{1, 2, 4\}$ hosts, and $N \in \{3, 5, 7, 9, 11, 13\}$ switches.

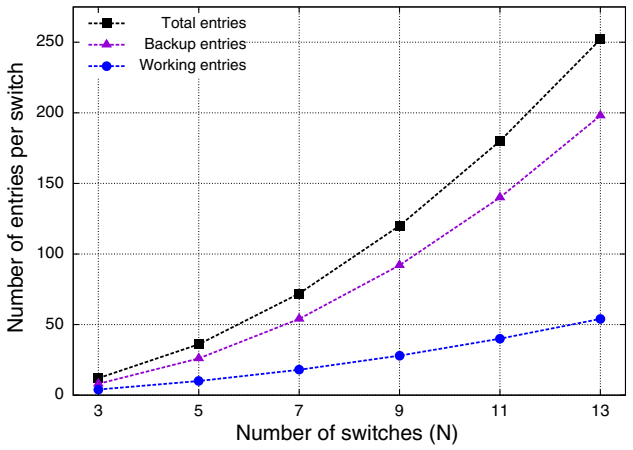

Fig. 8. Working, backup, and total number of entries in each switch with $M = 1$.

B. Switchover Time

As described in Subsection III.C, upon failure occurrence, data packets can be lost during the *switchover time*, T_f. This section evaluates T_f guaranteed by the OSP scheme in both emulated and implemented environments.

1) Emulation Environment: In the emulated environment tests, only one host is connected to each switch, $M = 1$. Conversely, two ring topologies have been considered with $N = 5$ and $N = 13$, respectively. With reference to Fig. 6, the flow between the host *Src*, attached to the switch *A*, and the host *Dst*, attached to the switch *C* (see Fig. 6), uses a ping application to generate a packet every 2 ms so that T_f can be estimated with a precision of ± 2 ms. The working path is *A*–*B*–*C*. Data packets are captured at the *Dst* host (using the TShark network protocol analyzer [26]). Upon failure on link *B*–*C*, data are wrapped back on port 2 at node *B*, and the backup path is *A*–*B*–*A*–*E*–*D*–*C*. T_f is estimated as the time between the reception of the last packet before the failure and the reception of the first packet after the failure at the host *Dst*. The failure on link *B*–*C* is repeated 200 times.

Figure 9 illustrates the obtained T_f distribution with the only active flow between *Src* and *Dst*. Its average value is 79.4 ms. In particular, T_f distribution presents one peak in the interval 20–30 ms, and most of the samples are distributed in the range 20–180 ms. Figure 9 shows the obtained T_f distribution, considering $N = 5$, $M = 1$, and any-to-any communication. In this case the ping application is also used so that each host generates one packet every 1 s toward all other hosts (i.e., in each switch there are $W = 10$ and $B = 26$ active entries). The obtained average of T_f is 79.44 ms with most of the samples concentrated in the range 20–120 ms. Figure 10 shows the obtained T_f distribution, considering $N = 13$, $M = 1$, and any-to-any communication. The obtained result is similar to Fig. 9, with an average value of 80.96 ms, with most of the samples concentrated in the range 20–160 ms. However, in both cases, some T_f samples higher than 180 ms have been measured.

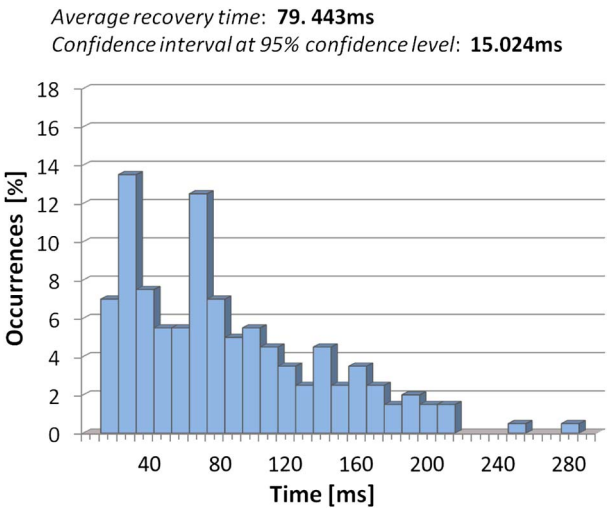

Fig. 9. Emulation T_f distribution with $N = 5$, $M = 1$, and any-to-any communication (i.e., 20 active protected flows).

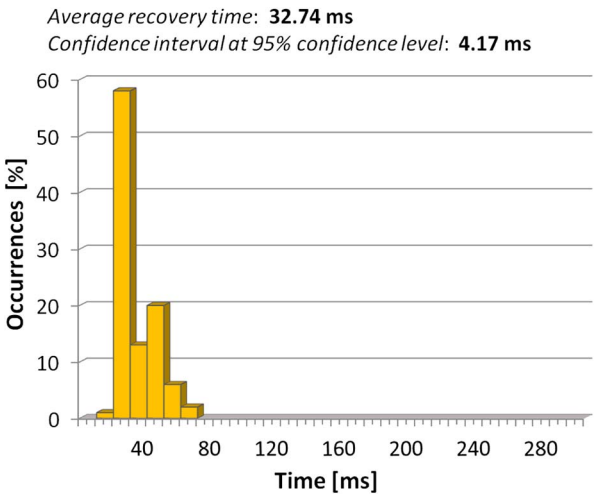

Fig. 11. Implementation T_f distribution with $M = 8$ and any-to-any communication (i.e., 1120 active protected flows).

Also, it is important to note that, in the considered measures, no traffic disruption has been detected during traffic reversion when the failure is physically repaired upon link recovery.

As explained in Subsection III.C, T_f is due mainly to the time required by the switch attached to the failed link for detecting the change of the interface status. This can explain the obtained results, where T_f can be seen as a random variable that is considerably dependent on the used hardware [14]. For this reason, faster T_f are expected in experimental demonstration of real implementation, where each switch has its dedicated hardware, with respect to the results of Fig. 9 and Fig. 10 obtained using the Mininet emulation.

2) Experimental Demonstration in Real Implementation Environment: T_f has been estimated in several tests using the Agilent N2X protocol analyzer to generate the required traffic streams. With reference to Fig. 6, the traffic flow from *Src* to *Dst* has been generated at the rate of one packet every 1 ms. Other traffic flows are generated at the rate of one packet every 1 s. The failure on link *B*–*C* is repeated 100 times, disconnecting the Ethernet cable alternately from the ports of switches *B* and *C*.

Figure 11 illustrates the obtained T_f distribution in the case of $M = 8$ hosts and any-to-any communication; i.e., 2308 entries are active in each network switch. Its average value is 32.7 ms, most of the samples are distributed in the range 20–50 ms, and all cases are included within 65 ms.

As in the emulation environment, no traffic disruption has been detected upon link recovery.

Figure 12 shows the minimum, average, and maximum values of T_f obtained over 100 repetitions as a function of the number of entries installed in each switch. Specifically, the figure reports the cases for $M \in \{2, 4, 6, 8\}$ corresponding to 144, 576, 1296, and 2308 entries per switch. T_f shows

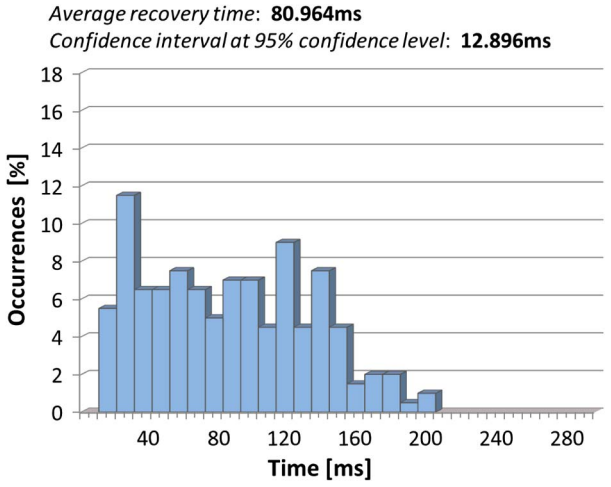

Fig. 10. Emulation T_f distribution with $N = 13$, $M = 1$, and any-to-any communication (i.e., 156 active protected flows).

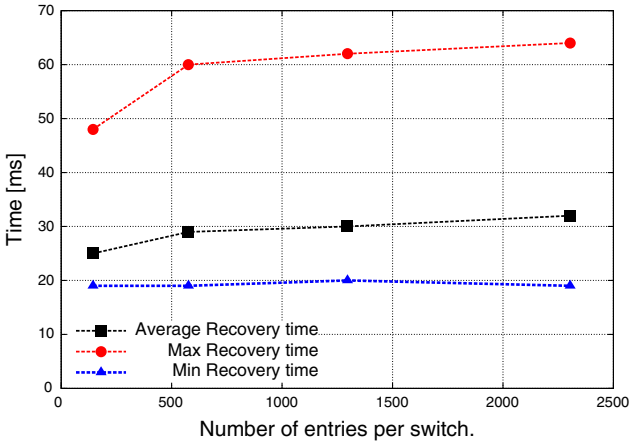

Fig. 12. Minimum, average, and maximum T_f trend as a function of the number of entries per switch.

a very limited increase with the number of entries, thus proving the scalability of the OSP scheme.

VI. Conclusion

This work proposed and implemented the OSP scheme, enabling fast recovery in networks composed of OpenFlow-based Ethernet switches. The OSP scheme does not require a full-state controller and, upon failure, it does not involve the controller in the recovery process. Specifically, The OSP scheme relies on working and backup entries that are configured in the network switches with different priorities, on the renewal mechanism to avoid backup flow expiration, on the auto-reject mechanism deleting the entries involved in the failure, and on a novel message to enable flow reverting on the working path.

Experimental demonstration implementation results showed that the OSP scheme also guarantees recovery below 64 ms with a high number of installed entries, thus preserving network scalability. Indeed, the achieved recovery time is determined only by the failure detection time. The emulation environment showed larger variance in the recovery time. Finally, no traffic disruption has been detected during reversion of flows when the failure is physically repaired or resolved.

Acknowledgments

This work has been partially supported by the project EMOTICON (Extending OpenFlow for Unified Management and Control of Cloud Data Center Resources) approved within the 2nd Call for additional beneficiaries of the OFELIA project. Part of this work was presented in [27].

References

[1] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling innovation in campus networks," *Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Aug. 2008.

[2] J. Naous, D. Erickson, G. A. Covington, G. Appenzeller, and N. McKeown, "Implementing an OpenFlow switch on the NetFPGA platform," in *Proc. of the 4th ACM/IEEE Symp. on Architectures for Networking and Communications Systems*, 2008, pp. 1–9.

[3] SPARC, http://www.fp7-sparc.eu/.

[4] S. Das, G. Parulkar, N. McKeown, P. Singh, D. Getachew, and L. Ong, "Packet and circuit network convergence with OpenFlow," in *Optical Fiber Communication Conf. and the Nat. Fiber Optic Engineers Conf. 2010*, Mar. 2010, paper OTuG1.

[5] A. Giorgetti, F. Cugini, F. Paolucci, and P. Castoldi, "OpenFlow and PCE architectures in wavelength switched optical networks," in *16th Int. Conf. on Optical Network Design and Modeling (ONDM)*, Apr. 2012, pp. 1–6.

[6] S. Azodolmolky, R. Nejabati, E. Escalona, R. Jayakumar, N. Efstathiou, and D. Simeonidou, "Integrated OpenFlow-GMPLS control plane: An overlay model for software defined packet over optical networks," in *37th European Conf. and Expo. on Optical Communications*, Sept. 2011, paper Tu.5.K.5.

[7] J. Kempf, E. Bellagamba, A. Kern, D. Jocha, A. Takacs, and P. Skoldstrom, "Scalable fault management for OpenFlow," in *IEEE Int. Conf. on Communications (ICC)*, June 2011, pp. 6606–6610.

[8] F. Paolucci, F. Cugini, N. Hussain, F. Fresi, and L. Poti, "OpenFlow-based flexible optical networks with enhanced monitoring functionalities," in *European Conf. and Exhibition on Optical Communication*, Sept. 2012, paper Tu.1.D.5.

[9] J. D. Decotignie, "The many faces of Industrial Ethernet [past and present]," *IEEE Ind. Electron. Mag.*, vol. 3, no. 1, pp. 8–19, Mar. 2009.

[10] L. Liu, T. Tsuritani, I. Morita, H. Guo, and J. Wu, "OpenFlow-based wavelength path control in transparent optical networks: A proof-of-concept demonstration," in *37th European Conf. and Expo. on Optical Communications*, Sept. 2011, paper Tu.5.K.2.

[11] Metro Ethernet Forum (MEF), "Requirements and framework for Ethernet service protection in metro Ethernet networks," Technical Specification MEF 2, Feb. 2004.

[12] B. Niven-Jenkins, D. Brungard, M. Betts, N. Sprecher, and S. Ueno, "MPLS-TP requirements," IETF MPLS Working Group Internet Draft draft-ietf-mpls-tp-requirement-10, Aug. 2009.

[13] M. Huynh, P. Mohapatra, S. Goose, and R. Liao, "RRR: Rapid ring recovery sub-millisecond decentralized recovery for Ethernet rings," *IEEE Trans. Comput.*, vol. 60, no. 11, pp. 1561–1570, Nov. 2011.

[14] A. Giorgetti, F. Cugini, F. Paolucci, L. Valcarenghi, A. Pistone, and P. Castoldi, "Performance analysis of media redundancy protocol (MRP)," *IEEE Trans. Ind. Inform.*, vol. 9, no. 1, pp. 218–227, 2013.

[15] S. Sharma, D. Staessens, D. Colle, M. Pickavet, and P. Demeester, "Enabling fast failure recovery in OpenFlow networks," in *8th Int. Workshop on the Design of Reliable Communication Networks (DRCN)*, Oct. 2011, pp. 164–171.

[16] OpenFlow Switch Specification 1.0.0, June 2012 [Online]. Available: http://www.openflow.org/.

[17] D. Staessens, S. Sharma, D. Colle, M. Pickavet, and P. Demeester, "Software defined networking: Meeting carrier grade requirements," in *18th IEEE Workshop on Local Metropolitan Area Networks (LANMAN)*, Oct. 2011, pp. 1–6.

[18] S. Sharma, D. Staessens, D. Colle, M. Pickavet, and P. Demeester, "OpenFlow: Meeting carrier-grade recovery requirements," *Comput. Commun.*, vol. 36, no. 6, pp. 656–665, Mar. 2013.

[19] Y. Yu, L. Xin, C. Shanzhi, and W. Yan, "A framework of using OpenFlow to handle transient link failure," in *Int. Conf. on Transportation, Mechanical, and Electrical Engineering (TMEE)*, Dec. 2011, pp. 2050–2053.

[20] M. Desai and T. Nandagopal, "Coping with link failures in centralized control plane architectures," in *2nd Int. Conf. on Communication Systems and Networks (COMSNETS)*, Jan. 2010, pp. 1–10.

[21] P. Fonseca, R. Bennesby, E. Mota, and A. Passito, "A replication component for resilient OpenFlow-based networking," in *IEEE Network Operations and Management Symp. (NOMS)*, Apr. 2010, pp. 933–939.

[22] Open vSwitch documentation [Online]. Available: http://openvswitch.org/support/.

[23] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, "NOX: Towards an operating system for networks," *Comput. Commun. Rev.*, vol. 38, no 3, pp. 105–110, July 2008.

[24] G. Yoon, D. H. Kwon, S. C. Kwon, Y. O. Park, and Y. J. Lee, "Ring topology-based redundancy Ethernet for industrial network," in *SICE-ICASE Int. Joint Conf.*, Oct. 2006, pp. 1404–1407.

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

[26] Tshark, http://www.wireshark.org/docs/man-pages/tshark.html.

[27] A. Sgambelluri, A. Giorgetti, F. Cugini, F. Paolucci, and P. Castoldi, "Effective flow protection in OpenFlow rings," in *Optical Fiber Communication Conf. and the Nat. Fiber Optic Engineers Conf.*, Mar. 2013, paper JTh2A.01.