In-Band Control, Queuing, and Failure Recovery Functionalities for OpenFlow

Sachin Sharma, Dimitri Staessens, Didier Colle, Mario Pickavet, and Piet Demeester

Abstract

In OpenFlow, a network as a whole can be controlled from one or more external entities (controllers) using in-band or out-of-band control networks. In this article, we propose in-band control, queuing, and failure recovery functionalities for OpenFlow. In addition, we report experimental studies and practical challenges for implementing these functionalities in existing software packages containing different versions of OpenFlow. The experimental results show that the in-band control functionality is suitable for all types of topologies. The results with the queuing functionality show that control traffic can be served with the highest priority in in-band networks and hence, data traffic cannot affect the communication between the controller and networking devices. The results with the failure recovery functionality show that traffic can be recovered from failures within 50 ms.

In recent decades, the Internet has grown from being an experimental research network to a broadband commercial platform. At the same time, the Internet has been facing many technical challenges such as complexity and inflexibility to meet changing requirements. To solve these challenges, numerous research activities such as the Clean Slate Internet program [1] and the Future Internet [2] have been started to develop appropriate solutions. A major outcome of the former is the idea of decoupling the control plane from the data plane in Internet devices (e.g. switches/routers) and embedding the control plane into one or more servers, called controllers. This enables independent evolution of the control and data plane. In addition, an interface between the data and control plane has been proposed. The most prominent protocol implementing such an interface is the OpenFlow protocol [3].

The current research of OpenFlow focuses mainly on an out-of-band network (Fig. 1a) in which control traffic (traffic to or from the controller) is sent on a separate network [4]. Such an out-of-band network has the following main advantages:

- High security is provided for control/management information because a separate network is used for communication.
- Access to the switches is possible through the separate network even if there are failures in the data traffic paths.

However, these networks are expensive to build due to the requirement of a separate network. Also, building a separate network may not be feasible in some scenarios (e.g. widely distributed central offices in access networks).

To solve the above problems, OpenFlow is required to be implemented for an in-band control network (Fig. 1b) in which control traffic is sent on the same infrastructure as the data plane [5]. However, for such a network OpenFlow does not describe how a switch can establish a communication path (e.g. control traffic path in Fig. 1b) with the controller. Without configuring these paths automatically, operators may face many manual configuration problems such as going into the field to configure the switches. In this article, we implement a method (known as bootstrapping) that inserts this information automatically in in-band networks. We refer to this as *in-band control functionality*.

In in-band control networks, control traffic may compete with data traffic for network resources (e.g. bandwidth) [6] as both share the same infrastructure. Therefore, due to an increase in data traffic, the control plane operations (such as new service establishment, failure recovery, load sharing) may suffer significant delay, and the controller and switches may even disconnect. To solve these problems, we extend the in-band functionality by implementing separate queues for control and data traffic, and by serving the control traffic queue before the data traffic queue. We refer to this mechanism as *queuing functionality*.

In in-band control networks, failures in the data plane (switch or link failures) can affect both data and control traffic. As a loss in data traffic causes a disruption of service, and a loss in control traffic prevents any new service establishment from the switches affected by failures, failure recovery is important for both data and control traffic. For failure recovery, some networks offer carrier-grade quality (RFC 5654), meaning that a network should recover from failures within 50 ms. Therefore, we explore two well known techniques, restoration and path protection, for fast failure recovery of both control and data traffic. In restoration, an alternative path is established after a failure. In path protection, a disjoint alternative path is established before a failure, and when the failure is detected, traffic is redirected to the alternative path. For failure detection in restoration, loss-of-signal (LOS) can be used because it can detect failures in any forwarding port. However, as LOS cannot detect failures in any path in protection, bidirectional forwarding detection (BFD) (RFC 5880) can be used.

We proposed in-band control and failure recovery functionalities in [7] and [8] respectively. In this article, we extend these functionalities with queuing functionality and integrate BFD in OpenFlow switches for fast failure recovery. In addition, we report practical challenges for implementing these in existing OpenFlow packages, containing different OpenFlow versions.

The authors are with Ghent University.

0890-8044/16/$25.00 © 2016 IEEE

Furthermore, we implement these functionalities in one of the OpenFlow software packages and perform extensive experiments. The experiments with in-band control show that the implemented method is suitable for all types of topologies. The experiments with queuing show that data traffic does not affect the communication between the controller and switches, and the experiments with failure recovery show that carrier-grade quality can be achieved in OpenFlow.

The following section describes our proposed functionalities, the third section describes practical challenges, the fourth describes experimentation, and the final section concludes the article.

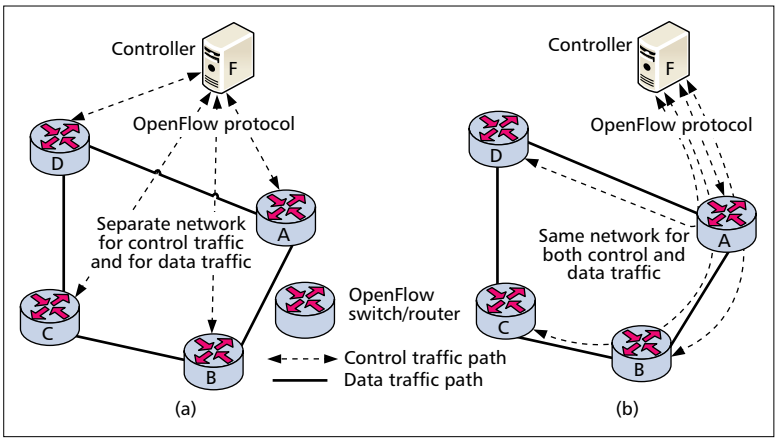

Figure 1. OpenFlow network: a) out-of-band; b) in-band control.

Functionalities for OpenFlow

In-Band Control Functionality

For in-band control, each switch and the controller have to establish an OpenFlow session over a transport layer protocol such as TCP, SCTP, or UDP. As switches and the controller need a reliable connection between each other, TCP or SCTP are preferred over UDP. In addition, as not all the platforms support SCTP, TCP is mostly used for establishing sessions.

The method for in-band control may differ with the types of OpenFlow switches used in the network. Today, there are two types of OpenFlow switches: pure and hybrid [9]. Pure switches support only OpenFlow operations for forwarding packets. Hybrid switches support both OpenFlow and traditional switching operations (e.g. layer 2 Ethernet switching, layer 3 routing, and VLAN isolations) for forwarding packets, and are common with many manufacturers such as Brocade, Juniper, and Cisco. We implement a loop free in-band control method using hybrid switches. In this method, at the time of bootstrapping, a switch applies Ethernet switching operations to forward its own traffic and applies OpenFlow operations to forward control traffic of other switches.

We frame the following three challenges for implementing in-band control:

- Each switch needs to configure a unique IP address for itself.
- Each switch needs to know the IP address and transport layer port (e.g. TCP port) of the controller.
- Communication paths need to be established for the switches (B, C, and D in Fig. 1b) that are not directly connected with the controller.

To solve the first and the second challenge, the following steps are performed:

- Each switch runs a DHCP (Dynamic Host Configuration Protocol) client.
- Each switch runs a hybrid stack to forward its own traffic (e.g. DHCP traffic).
- The DHCP server is configured with a vendor-specific option containing the IP address and transport layer port of the controller.
- Either the DHCP server is located on the controller or the DHCP server and the controller share IP (sub) networks (direct communication) with the same switch (switch A in Fig. 1b).

To solve the third challenge, the controller establishes communication paths through the switches that have already established OpenFlow sessions.

Our in-band control method using DHCP solves the problems of configuring each switch with a unique IP address and other transport layer parameters (e.g. TCP port) of an OpenFlow session.

The four steps to perform in-band control are:

1. Notification of the required network parameters.
2. Establishment of a TCP session.
3. Establishment of an OpenFlow session.
4. Discovering the topology.

For the first step, each switch periodically sends DHCP messages to its neighbors until it receives a reply from the DHCP server. If a neighbor is the DHCP server, it replies to the switch (A in Fig. 1b). Otherwise, the neighboring switch may forward or drop the messages, depending on whether it has an OpenFlow session with the controller. In case the neighboring switch has the session, the controller allows the neighboring switch to forward the DHCP messages to the DHCP server. When a switch knows its IP address and the IP address of the controller (using DHCP), it runs ARP to know the MAC address of the controller. After knowing the MAC address, the switch performs the second step.

In the second step, the switch establishes a TCP session with the controller. Either it establishes the session directly (in case of switch A) or the controller specifies a session path (shortest path) through the switches having an OpenFlow session.

In the third step, the switch instantiates an OpenFlow session with the controller [9].

In the fourth step, the controller discovers links of a switch after establishing the session with it. For this, the controller allows the switch to flood probe messages (Link Layer Discovery Protocol messages). From the received messages, the controller discovers links of the switch [10]. In addition, the controller discovers links of DHCP clients (for switches B, C, and D) and the DHCP server on reception of DHCP messages from them, and the same flooding mechanism (as probe messages) is exploited to infer the location of the DHCP server. In this case, instead of probe messages, DHCP messages are flooded.

Queuing Functionality

A part of queuing functionality, i.e. the creation of queues, is out-of-scope for OpenFlow. However, with the OpenFlow protocol, a packet can be redirected through an already created queue. For the creation of queues, switches can rely on a separate protocol such as OF-Config (OpenFlow Configuration and Management Protocol) or OVS-DB (Open vSwitch Database Management Protocol) [11]. For the case when switches do not support these protocols, vendor specific options of the OpenFlow protocol can be used for the creation of queues. Many switches such as HP, Reference, Indigo, Trafficlab1.1, and Trafficlab1.3 (Table 1) allow queue creation through vendor-specific options. However, with these options, only a few types of queues (such as rate limiting queues) can be created. In this article, we propose to extend the vendor-specific option of switches to create queues with different priorities. In our proposal, the queue creation message of the vendor-specific option is extended to add a priority number, and therefore

	Software component	Reference switch (license BSD) www.openflow.org	Indigo (license Eclipse) www.openflowhub.org	Open vSwitch (up to VER 1.11.0) (license Apache 2.0) www.openvswitch.org	Trafficlab1.1 (license BSD) https://github.com/TrafficLab	Trafficlab1.3 (license BSD) https://github.com/CPqD
In-band	DHCP client	Yes	Yes	Not functional for in-band control	Not functional	Not functional
	NORMAL stack	Yes	Yes	Yes	Yes	Not functional
	TCP stack	Yes	Yes	Yes	Yes	Yes
	OpenFlow stack	Yes	Yes	Yes	Yes	Yes
Queuing	Queue forwarding	Yes	Yes	Yes	Yes	Yes
	Queue creation	Yes (with VS)	Yes (with VS)	Yes (with OVS-DB)	Yes (with VS)	Yes (with VS)
	Queues with priorities	No	No	Yes (with OVS-DB)	No	No
Failure recovery	LOS failure detection	Yes	Yes	Yes	Yes	Yes
	BFD failure detection	No	No	No	No	No
	Group-table (fast-failover)	No	No	No	Yes	Yes

Table 1. Feature required/present in different implementations of OpenFlow switches. VS is the vendor-specific extension; OVS-DB is the Open vSwitch database management protocol; VER is the Open vSwitch version.

on reception of this message, a switch can create queues having different priorities using switch traffic control commands (e.g. Linux traffic control commands in the Reference, Trafficlab1.1, and Trafficlab1.3 switches).

In queuing functionality, the aforementioned extension is used for creating different queues for control and data traffic. The control traffic queue is given the highest priority, and hence is served before any other queue. When all switch port information is received, the controller creates the control traffic queue on each port of the switch. For data traffic, the controller can create queues either in advance or on reception of data traffic. In addition, when the controller receives traffic to define its forwarding, the controller adds a forwarding entry to redirect control traffic to the control traffic queue and data traffic to the data traffic queue. For separating control and data traffic, the controller uses the source IP address, destination IP address, and transport layer parameters of an OpenFlow session in a forwarding entry of control traffic.

Failure Recovery Functionality

In OpenFlow, a switch sends an echo-request to the controller after an idle timeout. If it does not receive an echo-reply before an echo timeout, it declares failures. The switch then tries to establish a new session. If it fails, it waits for a backoff timeout to re-establish the session. As the minimum value of idle, echo, and backoff timeouts are 1 second, failure recovery cannot be achieved in milliseconds. Therefore, we implement two fast recovery techniques, restoration and path protection, for single failure scenarios in in-band networks.

For implementing restoration, the controller depends on a failure notification (PORT_STATUS [9]) instead of the echo timeout to declare failures. The controller receives the notification when a switch detects LOS and still has a connection with the controller. The challenge behind restoration is that the controller has lost communication with the affected switches and therefore it cannot establish paths from (or along) these switches.

To overcome the challenge, during bootstrapping the controller establishes a one-hop restoration path together with the working path for control traffic. In this path, the source switch floods its own traffic, only one neighbor (which is along the working path) forwards the traffic, and other neighbors just drop it. On the failure notification, the controller first makes a list of affected switches that can be restored first and then restores the affected switches according to the list. This is done because the restoration path of affected switches may be along the switches that are affected by the failure, and hence before establishing the path these switches are needed to be restored.

In addition to restoration, failure recovery can be achieved by protection. Protection removes the need of establishing an alternative path after a failure by installing it in advance. We implement 1:1 path protection in which the ingress switch redirects traffic to a pre-established disjoint alternative path when a failure is detected in the working path. For pre-establishing the path, the controller uses the group-table concept (fast-failover) [9] at the ingress switch and uses the flow-table concept in all other switches along the paths. With the group-table concept, two rules are kept for traffic forwarding. Before a failure, the ingress switch applies the first rule (which corresponds to the working path), and after the failure it applies the second rule, which corresponds to the alternative path.

In addition to control traffic, we apply the above restoration and protection techniques for data traffic. However, in the case of restoration of data traffic, the one-hop restoration paths are not established in advance, and after recovering control traffic, data traffic is restored.

In the case of protection of both control and data traf-

fic, BFD can be used. In BFD, a pair of switches transmits BFD packets periodically between each other, and if a switch stops receiving the packets, the path between the switches is assumed to be failed.

OpenFlow does not define how to run BFD in the switches. Therefore, we propose to integrate BFD in the local networking stack of switches and add a vendor-specific extension in the OpenFlow protocol to run it through the switches. With this vendor-specific extension, the controller sends a message containing information about a BFD session (RFC 5880). Upon reception of this message, the switch runs the BFD session on the local networking stack, which allows the switch to send BFD packets through its local port (reserved port of the switch).

In protection, when the controller establishes the working and alternative path between two edge switches, the controller sends vendor-specific messages to edge switches to start a BFD session between them. In addition, the controller establishes a path for the BFD session, which follows the same path as the working path. Hence, when BFD detects the failure, the ingress switch declares the working path as a faulty path and therefore, the ingress switch can now apply the second rule.

Practical Challenges

Evolution of OpenFlow Specifications

Stanford University released specifications for OpenFlow known as version 1.0 and 1.1 in 2009 and 2011, respectively, and industrial players such as Deutsche Telekom, Google, Microsoft, and Yahoo! have shown substantial interest in this technology. These companies then formed ONF (Open Networking Foundation) to standardize and release the versions of OpenFlow according to their demands. Since then, six more versions (1.2, 1.3.0, 1.3.1, 1.3.2, 1.3.3, and 1.4.0 [9]) have been released publicly. Hence, the challenge is to choose which version to use for implementation of our functionalities. In addition, as OpenFlow is evolving quickly, not all versions or all enhancements of the released versions are implemented for OpenFlow.

Availability of Required Switch Components

Table 1 shows the availability of the required components in existing implementations. The functions of these components are described below:

With a DHCP client [12], a switch can generate/receive DHCP messages from the local port. With the NORMAL stack [9], a source switch can forward its messages using L2 learning when it does not have an OpenFlow session with the controller. Using the TCP stack, a switch can establish a TCP session. Using the OpenFlow stack [9], a switch can establish an OpenFlow session. Using queue forwarding [9], traffic can be forwarded through queues. With the queue creation component, queues can be created in switches. Using queues having different priorities, queues can be served on a priority basis. Using LOS, a switch can detect failures in restoration. Using BFD, switches can detect failures in protection, and with the group-table fast-failover type, switches can change the actions of forwarding packets without contacting the controller.

The existing implementations for required components are Reference switch, Indigo, Open vSwitch, Trafficlab1.1, and Trafficlab1.3 (Table 1). Reference switch [3] is the first software release of OpenFlow version 1.0. Indigo is a hardware-switching release based on Reference switch. Open vSwitch is a production quality release of OpenFlow. Trafficlab1.1 is the extension of Reference switch to incorporate version 1.1, and Trafficlab1.3 contains version 1.3.0.

Table 1 shows that not all the required components are present in a single implementation. Therefore, the challenge is to integrate all the required components in one implementation. In addition, vendor-specific extensions of switches are required to configure queues with different priorities. Furthermore, BFD is not present in any implementations. Hence, it is needed to be either implemented fully or imported from any open-source implementations of BFD. Some modifications related to BFD are also required. The modifications are: running BFD sessions on the local networking stack; listening or sending BFD packets on the local port; and modifying a group-table entry on detection of a failure.

Availability of Required Controller Components

The required controller components are:

- In-band control, which can run in-band functionality on the controller.
- Queuing, which can establish queues (with different priorities) in the switches.
- Failure recovery, which can implement restoration or protection for control and data traffic.

Currently, none of the available controllers (e.g. NOX, POX, Floodlight) implement these components.

For the implementation of these components, the available controllers can generate several events. The events, which are important for the required components, are:

- Switch-join, which is generated when a switch establishes an OpenFlow session with the controller.
- Switch-leave, which is generated when a switch disconnects from the controller.
- Port-config, which is generated when the controller receives all port information.
- Packet-in, which is generated when a packet is received to decide its forwarding action.
- Port-status, which is generated when an LOS is detected or repaired in one of the ports in a switch.

Using these events, all the proposed functionalities can be implemented in all the available controllers.

Experimental Studies

We implemented our proposed functionalities in Traffliclab1.3 and in its compatible controller (NOX1.3). We used this switch because at the time of our implementation this was the only available soft switch containing the latest version and the group table concept. We implemented all the unavailable and non-functional software components (Table 1) in this switch using the mechanisms provided above.

The experiments are carried out using the emulated topologies described in Table 2. The pan-European topologies in Table 2 are basic reference topology (BT), core topology (CT), large topology (LT), ring topology (RT), and triangular topology (TT). The topologies that vary with the degree of meshedness are RT and TT, and the topologies in accordance with the number of switches are CT and LT. The other used topologies are ring, star, random regular graph, and balance binary tree. For experiments, one of the switches is physically connected with the controller, and the DHCP server is located on the controller.

For the in-band experiments, we use a single node of the iMinds island of the OFELIA testbed [13], and mininet [14] is used for emulating topologies. For all other experiments, a node of the island is dedicated to a single switch or the controller, and the topologies are generated using the emulab interface of the island.

	Topologies		#switches	#links	Switch degree		
					Min	Mean	Max
1	Pan European topologies	Core topology	16	23	2	2.88	4
		Basicreference	28	41	2	2.93	5
		Large topology	37	57	2	3.08	5
		Ring topology	28	34	2	2.43	4
		Triangular topology	28	61	2	4.36	7
2	Ring		100	99	2	2	2
3	Random regular graph		100	150	3	3	3
4	Balanced binary tree (height=5)		63	62	1	1.97	3
5	Star		100	99	1	1.98	99

Table 2. Emulated topoligies.

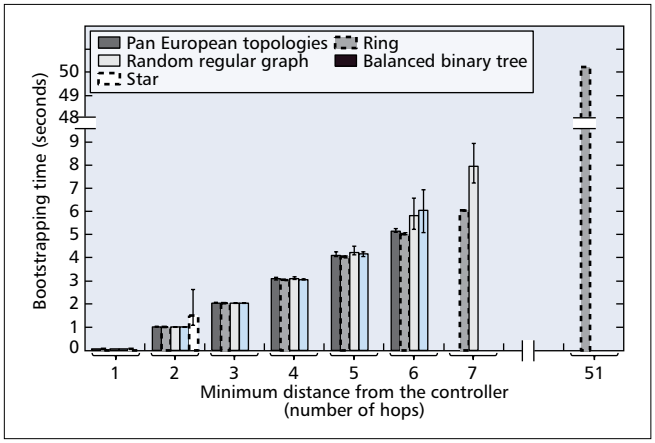

Figure 2. Bootstrapping time for all emulated topologies. The error bars show the minimum, average, and maximum values of the bootstrapping time.

In-Band Control Experiments

In the case of in-band control experiments, the DHCP retransmit timeout is kept as 1 second (minimum value) and the bootstrapping time (the total time to establish OpenFlow sessions) of switches is calculated with respect to the distance from the controller. As the bootstrapping time for all pan-European topologies is approximately the same, we show the bootstrapping time of all these topologies by a single bar (Fig. 2). In addition, at the distance where there is no switch in the topologies, no bar is shown in Fig. 2.

For one-hop, the bootstrapping time is approximately zero because the switch at one-hop does not wait for the DHCP timeout to retransmit a DHCP request. As the distance from the controller increases, Fig. 2 shows an increase in the bootstrapping time, because the switches, which are n-hop (n>1) away from the controller, are able to establish the session, if at least one of its neighbors has an OpenFlow session with it. When more switches are located at a certain distance from the controller (at distance 2 for star, at distance 6 for balance-binary tree, and at distance 6 and 7 for random-regular graph), we found a significant increase in the bootstrapping time, because in this case the in-band component of the controller receives lots of messages at about the same time. Until the controller replies, the messages stay in the packet-in buffer, increasing the bootstrapping time. In addition, as the buffer can overflow at some point, some of the messages have to be dropped. If a dropped message is a DHCP request, a switch waits for the next DHCP timeout to retransmit the DHCP request, and hence delays the bootstrapping time for an additional 1 second.

With Queuing and Without Queuing Experiments

In these experiments, the rate of data traffic (Poisson distributed on an average interval) is varied on each link of the CT topology, and the impact of data rate on control plane operations such as new switch connection (bootstrapping), new service installation, and failure recovery is calculated using queuing (Q) and without using queuing functionality (WQ). Each link of the topology is assigned a capacity of 10 Mb/s, and the size of data packets is 1000 bytes. All the results are calculated 50 times, and the average is shown in Fig. 3.

Figure 3 shows that under a low load (*load* < 0.9), bootstrapping, new service installation, restoration, and protection time is comparable for Q and WQ. However, at a high load (*load* > 0.9), due to congestion WQ takes a significantly longer time than Q for bootstrapping, new service installation, and restoration. In this case, as the load increases, switches drop more control and data packets. After dropping control packets, switches and the controller have to retransmit these packets after their timeouts, increasing the delay in completing bootstrapping, new service installation, and restoration. Moreover, at a *load* >1.04, WQ has a lower protection time (less than 40 ms) than Q. This is because due to congestion, switches have dropped some BFD packets (sent interval = 20 ms and timeout = 40 ms) just before a failure, allowing BFD to detect failures faster than in a normal condition. Furthermore, after a *load* ≥ 1.08, a large number of BFD packets drops in WQ due to congestion, and therefore BFD declares its timeout without the presence of the actual failure (link failure). This is the reason for zero protection time in WQ at a *load* ≥ 1.08 as traffic is already on the protection path at the time of failure. The results also show that all control plane operations take significantly shorter time in in-band control with queuing (Q) in all load conditions. Indeed, queuing functionality circumvents the competition between control and data traffic by implementing separate queues.

Failure Recovery Experiments

We performed the following three types of failure recovery experiments for in-band OpenFlow using queuing functionality:

- Control and data traffic.
- Multiple topologies.
- Switches disconnection experiments.

In control and data traffic experiments, the failure recovery time is calculated for one of the combinations of restoration and protection for control and data traffic. In multiple topology experiments, the recovery time is calculated for different types of topologies, and in switch disconnection experiments, the recovery time is calculated to show the impact of the increased number of disconnected switches along the recovery path. In the experiments, a failure is given by disabling Ether-

net interfaces, and for restoration, LOS is used to detect failures and the failure detection time is between 50 to 60 ms. For protection, BFD is used and the failure detection time is about 40 ms. All the results are calculated 50 times and the average is shown in Fig. 4.

In the control and data traffic experiments, the number of data flows (240 to 8400) is increased in the CT topology, and one of the combinations of restoration and protection is applied for control and data traffic. These combinations are:

- Restoration of both control and data traffic (Rest-Rest).
- Restoration of control traffic and protection of data traffic (Rest-Prot).
- Protection of control traffic and restoration of data traffic (Prot-Rest).
- Protection of both control and data traffic (Prot-Prot).

In all the combinations, Fig. 4 shows that restoration does not meet the carrier-grade requirement of 50 ms, while protection meets the requirement. In addition, the restoration time of data traffic (Rest-Rest and Prot-Rest) increases with the increase in the number of affected data flows, because as the number of affected data flows increases, a higher number of data traffic paths need to be configured after the failure.

In the multiple topology experiments, different types of pan-European topologies (CT, BT, and RT) are used, and we found that the restoration time increases with the number of switches in a topology, because in our implementation the path calculation time grows as $O(n^2)$, where n is the number of switches. In addition, as the degree of meshedness increases, the restoration time decreases, because in this case fewer hops are required for the restoration path, and therefore the controller needs to configure fewer switches in the network. Furthermore, protection does not require controller intervention, and therefore it is far less dependent on the network topology.

In the switch disconnection experiments, ring topologies are used, and the restoration time follows a linear relationship with the number of affected switches along the restoration path. For protection, the recovery time is always within 50 ms and meets the requirement.

Conclusion and Future Work

In this article, we have explored OpenFlow for in-band control, queuing, and failure recovery functionalities, and have performed extensive experiments. The in-band control experiments conclude that the proposed method allows bootstrapping in all types of topologies. With this method, switches of emulated pan-European topologies have taken a maximum of 5 seconds to perform bootstrapping. The queuing experiments demonstrate that in-band control traffic can be served first before any other traffic, and hence it can avoid competition with data traffic for network resources. The failure recovery experiments conclude that restoration in OpenFlow does not allow achieving 50 ms recovery, and protection for both control and data traffic allows achieving recovery within 50 ms. In our results, we did not take into account the propagation delay. Among all the presented results, the restoration time may significantly increase with the increase in the propagation delay, further strengthening the conclusion of the article that restoration cannot meet the requirement of 50 ms. As future work, the effects of propagation delay can be studied to quantify the degradation of the restoration time with an increase in propagation delay.

Based on the presented emulation results, we believe that our functionalities can be applied in production networks. However, to improve the accuracy of results, our experiments can also be performed on real environment testbeds such as GENI (Global Environment for Network Innovations) or FIBRE (Future Internet testbeds/experimentation between Brazil and Europe).

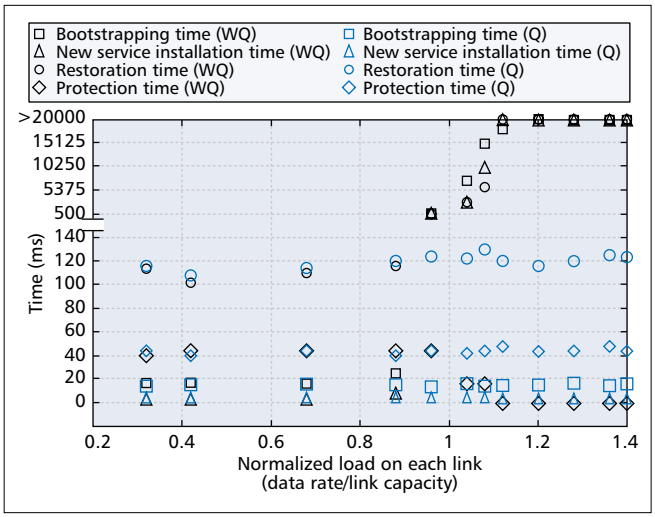

Figure 3. Impact of data traffic on control plane operations. WQ means in-band control without queuing functionality and Q means in-band control with queuing functionality.

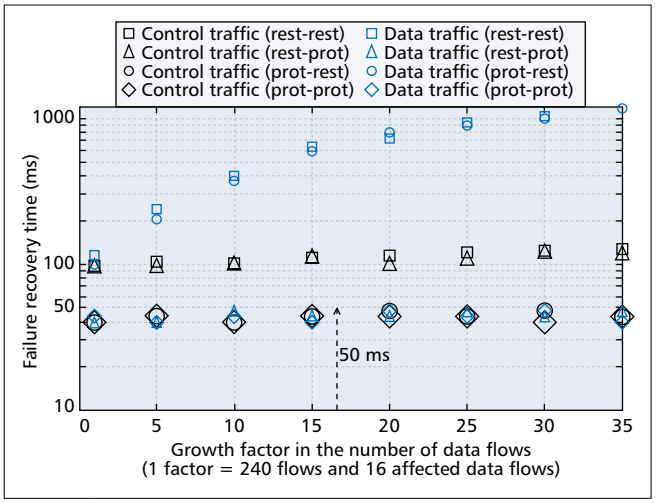

Figure 4. Recovery time of control and data traffic using all combinations of restoration and protection.

Using these testbeds, OpenFlow hardware switches can be used for experimentation and the topologies can be generated in real environment settings. In the experiments, the impact of real environment factors (e.g. hardware dependent parameters such as packet forwarding, processing, and queuing) on the results can be studied. For the bootstrapping time, we believe that this impact will be negligible, as the DHCP retransmit timeout (i.e. 1 second) dominates the bootstrapping time. For the restoration time, the impact can be significant as the restoration time is measured in ms and a small variation due to real factors will influence the results. For the protection time, the impact will be negligible because only the ingress switch along the protection path is involved for the protection activity.

In this article, we have not explored security and controller failure issues for in-band OpenFlow. For security, there can be many concerns related to DHCP [12], transport layer [15], and OpenFlow messages. These concerns are security issues related to:

- TCP or DHCP requests from bad actors.
- DHCP messages from an unauthorized DHCP server.
- Denial of service from the DHCP server or the controller.
- Switch datapath ID conflicts.

Nevertheless, transport layer security (TLS) described in OpenFlow [9] can be applied in the bootstrapping phase.

However, the problem is that OpenFlow does not provide any details of TLS operations. This could lead to interoperability issues. In addition, TLS has many technical barriers for operators. These are:

- Assigning controller certificates.
- Assigning switch certificates.
- Signing the certificates with a private key.
- Installing the keys and certificates into all network devices.

In future work, we will consider the aforementioned security issues and will explore controller failure solutions for in-band OpenFlow. To solve the controller failure issues, we will use two controllers. Hence, when one controller crashes, switches can rely on a backup controller to take actions.

Acknowledgments

This research has received funding from EU FP7 under agreement Nos. 317576 (CityFlow), 258457 (SPARC), and 258365 (OFELIA).

References

[1] J. Rexford, "Future Internet Architecture: Clean-Slate Versus Evolutionary Research," *Commun. ACM*, vol. 53, no. 9, Sept. 2010, pp. 36–40.
[2] J. Pan, S. Paul, and R. Jain, "A Survey of the Research on Future Internet Architectures," *IEEE Commun. Mag.*, vol. 49, no. 7, July 2011, pp. 26–36.
[3] N. McKeown *et al.*, "OpenFlow: Enabling Innovation in Campus Networks," *ACM Computer Commun. Review*, vol. 38, no. 2, April 2008, pp. 69–74.
[4] R. Ahmed and R. Boutaba, "Design Considerations for Managing Wide Area Software Defined Networks," *IEEE Commun. Mag.*, vol. 52, no. 7, July 2014, pp. 116–23.
[5] C. Tu, P. Wang, and T. Chiueh, "In-Band Control for an Ethernet-Based Software-Defined Network," *ACM SYSTOR*, 2014, pp. 1–11.
[6] P. Skoldstrom and K. Yedavalli, "Network Virtualization and Resource Allocation in OpenFlow-Based Wide Area Networks," *IEEE ICC*, June 2012, pp. 6622–26.
[7] S. Sharma *et al.*, "Automatic Bootstrapping of Openflow Networks," *IEEE LANMAN*, April 2013, pp. 1–6.
[8] S. Sharma *et al.*, "Fast Failure Recovery for In-Band Openflow Networks," *DRCN*, March 2013, pp. 44–51.
[9] OpenFlow specifications; available https://www.opennetworking.org/sdn-resources/onf-specifications
[10] A. S. Tan *et al.*, "Automatic Topology Discovery in Software Defined Networks," *SIU*, April 2014, pp. 939–42.
[11] S. Sharma *et al.*, "Demonstrating Resilient Quality of Service in Software Defined Networking," *IEEE INFOCOM WKSHPS*, May 2014, pp. 133–34.
[12] S. Duangphasuk, S. Kungpisdan, and S. Hankla, "Design and Implementation of Improved Security Protocols for DHCP Using Digital Certificates," *IEEE ICON*, Dec. 2011, pp. 287–92.
[13] M Sune *et al.*, "Design and Implementation of the OFELIA FP7 Facility: The European OpenFlow Testbed," *Computer Networks*, vol. 61, March 2014, pp. 132–50.
[14] V. Antonenko and R. Smelyanskiy, "Global Network Modelling Based on Mininet Approach," *HotSDN*, 2013, pp. 145–46.
[15] P. Casas, J. Mazel, and P. Owezarski, "Knowledge-Independent Traffic Monitoring: Unsupervised Detection of Network Attacks," *IEEE Network*, vol. 26, no. 1, Jan. 2012, pp. 13–21.

Biographies

SACHIN SHARMA (sachin.sharma@intec.ugent.be) received his M. Tech degree in computer applications in 2007 from IIT Delhi. From 2007 to 2010 he worked as a senior R&D Engineer at Tejas Networks, Bangalore. Since 2010 he has been working toward a Ph.D. degree from Ghent University, Belgium. During his Ph.D. work he has been involved in many European funded projects, including: SPARC, CityFlow, and Unify. He has more than 20 scientific publications in international conferences and journals.

DIMITRI STAESSENS received his M.Sc. degree in numerical computer science in 2004 from Ghent University, Belgium, and finished a Ph.D. on survivability of optical networks in 2012. This work led to more than 30 publications, and was performed in European projects such as NOBEL, DICONET, and NoE's ephoton/One and BONE. His current interests are in the control and management of networks, SDN, and future network architectures, where he is active in FP7 projects on RINA.

DIDIER COLLE received his M.Sc. and Ph.D degrees in electrotechnical engineering (option: communications) from Ghent University in 1997 and 2002, respectively, and became an associate professor at the same university in 2011. He is group leader of the Future Internet Department of iMinds. His research has published 300 articles in international journals and conference proceedings. He has been very active in FIRE projects, with a focus on OpenFlow and software defined networks.

MARIO PICKAVET is a full professor at Ghent University, where he is teaching courses on discrete mathematics and network modeling. He is co-leading the research cluster on network modeling, design and evaluation (NetMoDeL). He has published approximately 300 international publications, in journals including *IEEE JSAC*, *IEEE Communications Magazine*, and *Proceedings of the IEEE*, and in conference proceedings. He co-authored the book *Network Recovery: Protection and Restoration of Optical, SONET-SDH, IP, and MPLS*.

PIET DEMEESTER is a professor on the faculty of engineering at Ghent University. He is the head of the research group "Internet Based Communication Networks and Services" (IBCN, Ghent University), and is leading the Internet Technologies Department of the strategic research center iMinds. He has co-authored more than 1000 publications in international journals and conference proceedings. He is a Fellow of the IEEE.