electronics

Article

Priority-Based Bandwidth Management in Virtualized Software-Defined Networks

Luca Leonardi, Lucia Lo Bello * and Simone Aglianò

Department of Electrical, Electronics and Computer Engineering, University of Catania, Viale Andrea Doria 6, 95125 Catania, Italy; luca.leonardi@unict.it (L.L.); simoagliano@gmail.com (S.A.)
* Correspondence: lobello@unict.it

Received: 4 May 2020; Accepted: 12 June 2020; Published: 17 June 2020

Abstract: In Industrial Internet of Things (IoT) applications, when the network size increases and different types of flows share the bandwidth, the demand for flexible and efficient management of the communication network is compelling. In these scenarios, under varying workload and flow priorities, the combined use of Software-Defined Networking (SDN) and Network Virtualization (NV) is a promising solution, as such techniques allow to reduce the network management complexity. This work presents the PrioSDN Resource Manager (PrioSDN_RM), a resource management mechanism based on admission control for virtualized SDN-based networks. The proposed combination imposes bounds on the resource utilization for the virtual slices, which therefore share the network links, while maintaining isolation from each other. The presented approach exploits a priority-based runtime bandwidth distribution mechanism to dynamically react to load changes (e.g., due to alarms). The paper describes the design of the approach and provides experimental results obtained on a real testbed.

Keywords: software-defined networking; network virtualization; network slicing; resource management; admission control

1. Introduction

The Internet of Things (IoT) has brought significant benefits to several application scenarios, such as smart cities [1,2] and smart homes [3–5], smart metering [6], robotics [7,8], healthcare [9–11], and energy management systems [12]. Moreover, the introduction of IoT in industrial environments, i.e., the so-called Industrial Internet of Things (IIoT) [13–15], plays a key role in the Industry 4.0 context where a large number of different kinds of devices cooperate over a heterogeneous communication network [16,17]. The steady increase of both network complexity and sharing of physical resources in Industry 4.0 scenarios raises the need for flexible and efficient resource management solutions.

Recently, the combination of two emerging technologies, i.e., Software-Defined Networking (SDN) and Network Virtualization (NV), has received significant interest from the industrial and the academic communities [18–21], as a possible approach to successfully face the previously discussed challenges.

SDN allows to achieve flexible and easily manageable networks by providing a clear decoupling between the data plane and the centralized control plane. This way, SDN supports a better resource allocation, thus limiting issues, such as congestion, latency for high priority traffic, and so on. For instance, the management of the available network bandwidth can be performed by using a bandwidth reservation approach according to the application needs. The use of NV allows to split the physical infrastructure of a virtualized network into logical networks, called slices, with specific functionalities and requirements. SDN provides an easy way to define multiple virtual networks (VNs) and to support their complete isolation through the use of a network hypervisor.

Traditionally, the slices in a virtualized network do not share resources with each other to preserve the isolation among slices, i.e., the ability to manage a slice as a separate network, independent from the others. However, for the sake of efficient resource utilization, many scenarios can require that the network slices share some network links while still demanding isolation.

Previous work in [22] addressed an approach based on SDN and network virtualization that allows for shared resources. In such a work a bandwidth management mechanism for a virtualized SDN network allows slices to share links and, through limits on the bandwidth utilization of each slice, to share the bandwidth available on such links too. However, the approach in [22] uses static bandwidth limits and therefore it cannot adapt to traffic changes. Moreover, the work in [22] does not consider any mechanism for safe management of alarm flows. The latter require to find bandwidth available for them (i.e., reserved) at any time they arrive, as they are triggered by the occurrence of critical events and therefore they must alert the system controller (e.g., sink with processing capability) as soon as possible.

Differently from previous work, this paper proposes the PrioSDN Resource Manager (PrioSDN_RM), an SDN-based approach that aims to wisely manage the available network bandwidth through a dynamic priority-based mechanism based on an admission control. This way, the bandwidth limits are dynamically set depending on the traffic load and on the flows priority, i.e., the presented solution supports online network reconfiguration. The proposed solution assumes that alarm flows (i.e., the flows with highest priority) need to be strongly favored over the other flows.

The main contributions of the paper are:

- a novel bandwidth management strategy
- a detailed description of the proposed priority-based admission control
- an implementation of the proposed approach and a discussion of the results obtained through experimental performance assessments.

The rest of the paper is organized as follows. Section 2 provides a background on Software-Defined Networking and network virtualization. Section 3 deals with related works. Section 4 presents the PrioSDN_RM system, whereas Section 5 discusses a use case for the proposed approach. Sections 6 and 7 address the implementation and the experimental evaluation of PrioSDN_RM, respectively. Section 8 presents a comparative assessment with the relevant mechanisms in the literature. Finally, Section 9 gives conclusions and hints for future works.

2. Background

SDN refers to a network architecture with the main features listed below [23].

- Control and data planes decoupling. The network devices (e.g., switches) become simple forwarding elements, since the network logic is provided by the SDN controller.
- Control logic moving to an external entity. The SDN controller offers the key resources and abstractions to allow easy programming of forwarding devices, based on a logically centralized view of the network.
- Network programmability. The network is programmable through software applications running on top of the SDN controller that, in turn, interacts with the underlying networking devices.
- Flow-based forwarding. A flow is a sequence of packets exchanged between source and destination and is defined by a set of packet field values, acting as a match (filter) criterion, and a set of actions (instructions). The forwarding devices manage all the packets of a flow in the same way, thus the forwarding rules are flow-based rather than destination-based. As a result, flow programming offers high flexibility and makes it possible a unified behavior of different types of network devices.

Network Virtualization allows creating logical, virtual networks (VNs) that are decoupled from the underlying networking hardware (i.e., the physical infrastructure). This way, new network services can be added without worrying about the specific details of the physical devices. NV also allows sharing of physical networking resources, i.e., multiple virtual networks can operate over the same physical network.

SDN supports a simple way to define VNs. As a result, each virtual link can be represented as a flow. A VN includes a set of flow rules in different switches.

The virtualization of SDN networks enables to leverage the combined benefits of SDN and network virtualization.

The architecture of a virtualized SDN network is shown in Figure 1.

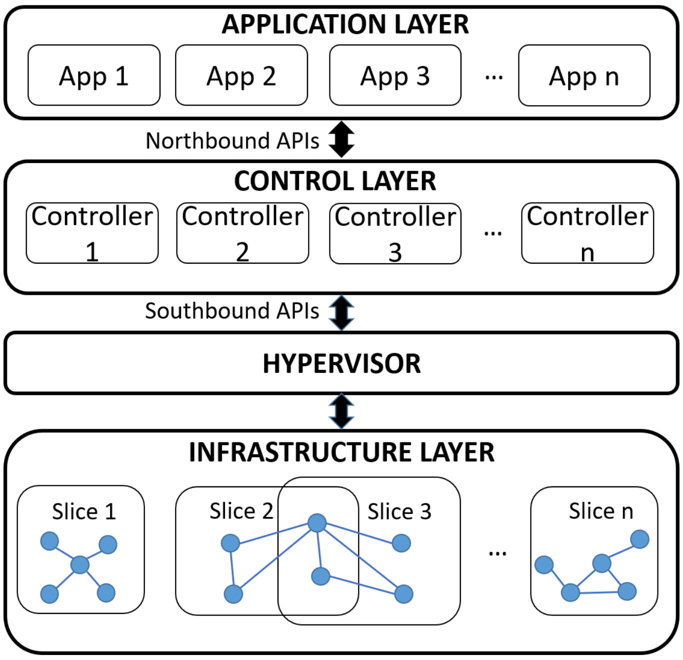

Figure 1. The architecture of virtualized Software-Defined Networking (SDN) network.

The Infrastructure layer consists of network equipment, e.g., end nodes and switches, and merely focuses on data transmission and forwarding.

The Control layer is the "brain" of the SDN network. It consists of one or more SDN controllers implementing software functions and algorithms that manage the forwarding platform of the network.

The hypervisor is located between the Infrastructure layer and the Control layer. It is a critical component for virtualizing SDN networks [24], as it abstracts the physical SDN network by creating multiple logically isolated virtual SDN networks (vSDNs), each one with its own SDN controller. The hypervisor monitors the virtual networks and allocates networking resources (e.g., link capacity) to each virtual network (also referred as a network slice).

The communication between the Infrastructure layer and the Control layer is realized through the southbound Application Programmming Interfaces (APIs). OpenFlow (OF) [25], developed by the Open Networking Foundation (ONF), is the most used communication protocol in SDN environments and enables the SDN controllers to interact with the data plane through a hypervisor or not.

The Application layer consists of a set of network management applications. Each application runs in its own network slice on top of an individual SDN controller, which only presents the network view that corresponds to a specific virtual network. The Application layer and Control layer use the so-called northbound APIs to communicate.

3. Related Work

From the state of the art, it is widely known that bandwidth reservation is a key network management mechanism to provide adequate levels of Quality of Service or real-time guarantees. For example, thanks to bandwidth reservation, the IEEE Audio-Video Bridging (AVB) set of standards is able to flexibly support multiple traffic classes even under high workloads, thus offering adequate performance to real-time flows both in automotive [26–28] and industrial communications [29]. However, the need for flexible and efficient resource management introduces new challenges, such as how to perform dynamic network reconfiguration while meeting the required Quality of Service (QoS). SDN allows the designers to easily implement dynamic and adaptable QoS control solutions for both wired and wireless networks [30,31]. Several works proposed resource management approaches suitable for various scenarios, such as data centers [32], 5G SDN-based networks [33], Industrial Wireless Sensor Networks [34,35], and IoT-enabled smart homes [36]. Other notable software-defined bandwidth management mechanisms were proposed in [37–40]. In particular, the work in [37] proposed an architecture for software-defined Ethernet Passive Optical Networks (EPONs) that allows the SDN controller to manage different dynamic bandwidth allocation algorithms based on the network status. In the works [38,39] two different mechanisms able to reserve bandwidth to each node of a software-defined network were proposed. Such mechanisms address a node-oriented bandwidth management, differently from our approach, that is flow-oriented. The work in [40] proposes an adaptive resource provisioning scheme for network flows that is based on continuous monitoring of several thresholds on the range of allowed transmission rates. Differently from our approach, none of such previous works deals with virtual networks.

A network hypervisor that includes an admission control was proposed in [41]. However, differently from our approach, it does not allocate bandwidth to the flows based on their priority, but it only checks whether enough bandwidth is available when a request for creating a network slice arrives.

A secure virtual controller (SVC) architecture was proposed in [42], that uses SDN in combination with network function virtualization. However, compared to our approach, such a work has a different target, as the aim of the SVC is to balance the traffic load and reduce the energy consumption.

A FlowVisor-based network virtualization layer, called EnterpriseVisor, was proposed in [43]. EnterpriseVisor proposes a dynamic resource management scheme that manages the distribution of network resources among slices to improve the overall network resource utilization. As a result, EnterpriseVisor is not able to provide Quality-of-Service (QoS) guarantees to high-priority flows.

The work in [44] proposed an extension of the Floodlight controller to dynamically manage link and switch resources in a virtualized SDN environment. The controller, based on the network status, performs path migrations by adding, modifying or deleting flow rules from the switches. Such an approach aims to improve the utilization of the network resources, whereas our goal in this paper is avoiding link overload to cope with bandwidth limits and to handle flows with different priorities.

This paper addresses the management of network link resources in virtualized SDN-based networks. In particular, we propose a mechanism, based on an admission control, to dynamically allocate the available bandwidth per link based on the priority of the flows.

Among previous works, the ones that are more relevant to our paper are the ones in [22] and [45]. In fact, both of them propose bandwidth management mechanisms, based on an admission control, that exploit SDN and network virtualization while supporting resource sharing. In particular, the approach in [22] proposes the use of static values for limiting the bandwidth utilization of each slice and supports high priority and low priority flows. However, such an approach is not able to react to critical events that cause changes in the network traffic. On the other hand, in [45] a dynamic bandwidth distribution framework, called DART, is described, that is able to dynamically distribute the network bandwidth between various services to allow an efficient use of the network resources. In DART, the SDN controllers cooperate to distribute the bandwidth among slices in order to mitigate congestion on the shared links. The admission control defines a priority limit and performs a simple task. Every time a new flow with priority higher than the priority limit asks for bandwidth, it is admitted for transmission and a flow rule is sent to the switch. Otherwise, the flow is prevented from transmitting. Comparing with DART, the PrioSDN_RM here proposed differs in some important respects. First, the admission control here proposed takes decisions based on both the flow priority and the network traffic, i.e., it does not directly discard low priority flows, as they can be admitted according to the policies discussed in Section 4. Second, we fix two different moving bandwidth thresholds that change when alarms occur. Third, in the PrioSDN_RM a portion of bandwidth is reserved to alarm flows to promptly handle critical events.

4. PrioSDN_RM: Bandwidth Management Strategy

This section presents the design of the PrioSDN_RM and the logic behind the admission control. The description of the reference system includes a generic virtualized SDN network. The admission control is presented in a detailed way, as it defines the strategy that allows a dynamic bandwidth management based on the priority of the flows.

4.1. System Design

The proposed resource management mechanism can be applied to any virtualized SDN network. The reference architecture is shown in Figure 2. Compared to a general virtualized SDN network, our solution includes a Priority-based Admission Control (PAC), which is an extension to a plain SDN controller and consists of a custom resource manager including specific policies for the bandwidth allocation. We assume to use OpenFlow for the communication between the infrastructure layer and the control layer. The physical network can be divided into multiple slices (thanks to the use of a hypervisor). The architecture can include both fully isolated slices and slices that share a part of the physical network to increase the resource utilization efficiency. Conversely, if a part of the network is shared between two or more slices, the shared resource allocation module is devoted to statically or dynamically negotiate the bandwidth portion reserved to each slice.

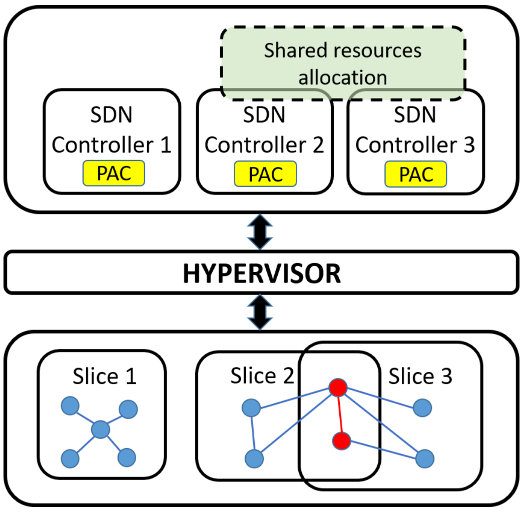

Figure 2. The PrioSDN Resource Manager (PrioSDN_RM) architecture.

In particular, Figure 2 depicts a network divided into three slices, therefore with three SDN controllers. In such a network, the slice 1 is isolated from the others, while the slices 2 and 3 share one network link (i.e., the one drawn in red in Figure 2). Consequently, both SDN 1 and SDN 2 can operate on the shared link.

4.2. Priority-Based Admission Control-Basic Concepts

The admission control is the core of the priority-based bandwidth management in the SDN controller. In the case of a typical SDN controller, whenever the first message of a specific flow is sent by an end node or the flow rule (i.e., the routing rule for that flow) is not found in the flow table of the switch (i.e., a table-miss rule event occurs), the switch queries the SDN controller to obtain the relevant flow rule. After that, the switch inserts the flow rule provided by the SDN controller in its routing table and forwards the message accordingly. Please note that the SDN controller does not perform any bandwidth consideration, e.g., it does not prevent overloading of the network links and therefore dropped messages (e.g., due to overflow of the network resources). Consequently, the SDN controller cannot guarantee a specific level of Quality of Service (QoS) for the flows, for instance in terms of Packet Loss Ratio (PLR).

For this reason, we propose a priority-based admission control (PAC) for the SDN controllers. Please note that in our proposal we consider three different types of flows, i.e., low priority, high priority and alarms. In particular, we assume that an alarm is an event-driven flow (i.e., it is triggered by the occurrence of a specific event). In our model, alarm flows have the highest priority in the network.

The PAC module, shown in Figure 3, includes a compute module (CM) and a storage in which the information about the current network state (e.g., the current bandwidth utilization for each network link) is maintained. The data in the storage are always updated thanks to the OpenFlow functionality that allows to retrieve the flows currently registered in the routing table and to compute the current bandwidth utilization. This way, when the switch asks the SDN controller for a flow rule, the latter checks whether there is enough bandwidth on the links to allocate a flow with a specific priority (see Section 4.3). If no bandwidth is available for this flow, the flow rule is not established.

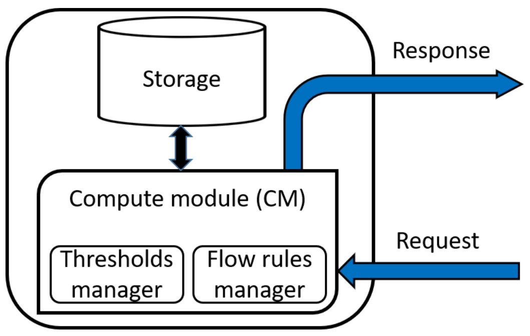

Figure 3. The priority-based admission control.

The decision on the bandwidth allocation is based on specific criteria. Each SDN controller keeps in the PAC storage the bandwidth limit value for each network link relevant to the slice that it manages. As a result, the compute module can allocate bandwidth for the flows of the slice up to the bandwidth limit of a specific link. Moreover, the available bandwidth per link is partitioned into three parts, as shown in Figure 4, using two moving thresholds, i.e., the alarm threshold (θ_{alarm}) and the high priority threshold (θ_{hp}). Such thresholds delimit the portions of bandwidth reserved for alarms and high priority flows, respectively.

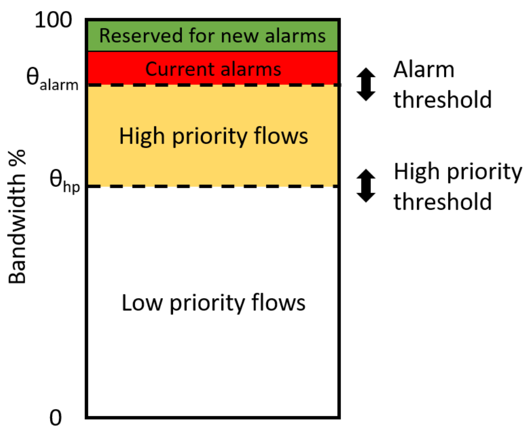

Figure 4. The bandwidth subdivision.

In particular, a portion of bandwidth is set-aside for alarm flows. These are the highest priority flows, as they have stringent timeliness and reliability requirements. For this reason, a fixed amount (a configurable value) of bandwidth, i.e., the green portion in Figure 4, is left "free" to be allocated to new alarm flows. Consequently, every time a new alarm flow is generated by an end node and then allocated by the SDN controller, the thresholds manager module of the PAC changes the alarm threshold by moving it downwards, thus enlarging the portion of bandwidth reserved for alarm flows (i.e., the sum of the green and red portions). This way, the red portion in Figure 4 (i.e., the bandwidth currently used by alarm flows) increases when a new alarm flow is generated, while the green section remains constant. Conversely, when an alarm flow is stopped (i.e., the alarm ends) the portion of bandwidth reserved to alarm flows is reduced by moving the alarm threshold upwards.

The choice to use a bandwidth reservation approach for alarm flows introduces a limited bandwidth waste, since a portion of available resources (i.e., the green portion) is not used. However, this makes it possible to promptly allocate bandwidth to alarm flows and start their transmission without introducing delay due to the lack of available bandwidth. This way, we provide the highest QoS to alarm flows. Please note that in PrioSDN_RM the portion of bandwidth that is left "free" to be allocated to new alarm flows is an application-dependent configurable parameter. To reduce the potential bandwidth underutilization to a minimum, we suggest to set the alarm threshold so as to leave "free" the bandwidth portion that is needed to accommodate one alarm flow only, i.e., the most demanding one among all the alarms that the application could generate.

The portion of bandwidth reserved for high priority flows (i.e., the orange portion in Figure 4), is a fixed amount (a configurable value) set-aside for the flows with a priority higher than a given application-dependent value (that is maintained in the PAC storage). Since the orange bandwidth portion must be fixed, the high priority threshold follows the changes of the alarm threshold, i.e., the two thresholds move in the same direction every time an alarm flow is generated or stopped. As a result, our approach applies a runtime monitoring of the two thresholds that automatically triggers the redistribution of resources to the flows based on their priorities.

Please note that in Figure 4, θ_{alarm} represents the maximum portion of bandwidth that can be used by high priority flows, while θ_{hp} is the maximum bandwidth available to low priority flows. In other words, this approach allows the high priority flows to use the bandwidth up to θ_{alarm}, i.e., they can use both the white section and the orange one in Figure 4.

4.3. Priority-Based Admission Control-Bandwidth Allocation Procedure

We assume that the end nodes can send multiple data flows, each one with a specific level of priority depending on the data importance. As a result, each end node can transmit flows with different priority levels.

The main aim of the PAC is to check the priority of the data flows and allocate bandwidth (if available) on the links that will be used for the flow transmission.

Whenever a new flow is generated by an end node, a request is sent to the switch the end node is directly connected to. This request includes the required bandwidth and the flow priority. As the switch does not have any forwarding rule for the messages of the new flow (i.e., the relevant flow rule is missing), it forwards the request to the SDN controller that manages the slice to which both the end node and the switch belong. Such a request is managed by the SDN controller and, in particular, it is delegated to the PAC module, which acts as follows.

The compute module extracts the flow parameters, i.e., the required bandwidth and priority, from the allocation request. Then, the flow rules manager submodule queries the PAC storage to check the current status of the network slice. In particular, what is needed is the current bandwidth utilization and the threshold values of the links within the slice managed by the SDN controller.

Once the flow rules manager checks if enough bandwidth is available on the links on which the flow should be transmitted, then it can decide about the allocation of the bandwidth for the flow according to the proposed algorithm. Since the admission control algorithm runs on each SDN controller in the network, henceforward we will refer to an individual SDN controller and therefore to a specific network slice to simplify the description of the approach here proposed.

The flow rules manager decides based on three parameters, i.e., bandwidth limit (BW_{lim}), thresholds (i.e., θ_{alarm} and θ_{hp}) and priority level ($prio_{lev}$). Each one of these parameters is defined for each link within the considered network slice. The BW^{l}_{lim} defines the maximum bandwidth of the link l that the network slice can use. If the link l is shared among two or more slices, then BW^{l}_{lim} will be only a portion of the total bandwidth available on the link. If a bandwidth allocation request would cause an increase of the bandwidth utilization of the link l above BW^{l}_{lim}, that request is not accepted.

The alarm threshold (θ_{alarm}) and the high priority threshold (θ_{hp}) allow to partition the bandwidth to provide different QoS levels in case of tight bandwidth availability. In such a case, the high priority flows (and in particular, the alarm flows) are favoured over the low priority flows. The $prio_{lev}$ defines a priority level that distinguishes high priority flows from the low priority ones. The flows with priority below $prio_{lev}$ are considered to have low priority. A smaller priority value represents a lower priority. For instance, if flows can have 8 different priority levels (i.e., from 0 to 7) and the priority level for the link l ($prio^{l}_{lev}$) is 4, then the priority values from 0 to 3 are considered as low priority (in particular, 0 is the lowest priority). The θ_{hp} parameter delimits the maximum portion of bandwidth that low priority flows can use. The bandwidth portion between θ_{hp} and θ_{alarm} is set-aside for the high priority flows. The bandwidth portion above θ_{alarm} is reserved to alarm flows.

If a request for a low priority flow to be transmitted on link l is forwarded to the SDN controller when the current bandwidth utilization exceeds the θ^{l}_{hp} threshold, the flow rules manager rejects it. Otherwise, the flow rules manager accepts the request.

Similarly, if a request for a high priority flow is sent when the current bandwidth utilization is above θ^{l}_{alarm}, the flow rules manager rejects it. Conversely, the requests for alarm flows are always accepted, as a fixed portion of bandwidth is reserved for them (see Section 4.2).

If the request can be accepted, the compute module gives a positive response and therefore the SDN controller sends the flow rule to the switch, which stores it in the flow table. Otherwise, the flow rule is not registered and the messages of the flow are not transmitted.

Once a flow rule is inserted in the flow table of a switch, a timer is set with a timeout value. When the flow rule timeout expires, the flow rule is dropped from the flow table. Consequently, a subsequent message of the flow would not be directly transmitted, as the switch will not find the relevant flow rule in its flow table. The switch will forward the request to the SDN controller, which creates a new flow rule or drops the flow, according to the PAC module mechanism. The value of the flow rule timeout allows the choice of the desired trade-off between the QoS and the overall delay introduced by the admission control. In fact, a lower timeout value introduces a higher delay, as the switch will experiment a higher number of table-miss rule events, and therefore the admission control task will be performed more times. On the other hand, this choice increases the proposed mechanism capability to quickly adapt to changes in traffic flows, as the admission control would evaluate the current state of the network more frequently. Otherwise, a higher value of the timeout will bring the opposite outcome.

Please note that the value of the timeout is per-flow and can be changed at runtime, i.e., the SDN controller can set the timeout with different values (according to a specific algorithm) every time the flow rule expires. This way, different values of timeout can be used for the low priority flows and the high priority ones. For instance, the use of a smaller timeout for the low priority flows is expected to increase the QoS offered to high priority flows in case of traffic congestion. In fact, we assume that low priority flows use as much bandwidth as possible (i.e., all the white section in Figure 4) and the high priority flows use a portion (%) of bandwidth equal to $(\theta_{alarm} - \theta_{hp})$%. If a new high priority flow asks for bandwidth, such a request will fail, at least until the low priority flows release the required bandwidth, i.e., at least until the timeout of a low priority flow expires. As a result, a lower timeout value for low priority flows guarantees a better management of the high priority flows.

5. A Use Case for the PrioSDN_RM

The PrioSDN_RM can be applied to several real scenarios, such as the one discussed in [45]. In particular, the network topology is the same, while the kind of application is slightly different as in [45] no alarm flows are present. Here we consider a factory-wide communication network, that includes a surveillance system in which the bandwidth is dynamically allocated based on the priority of the flows. The network is partitioned into several slices to keep separated virtual network domains (e.g., shop floor, sales office, administrative office, etc.) and to differentiate the QoS levels across the slices. However, some of the network services have to be available for more than one slice simultaneously and therefore resource sharing between slices has to be supported too.

The end nodes send messages, organized in flows, with different priorities, according to their QoS requirements. For instance, the surveillance system includes end nodes with attached cameras that transmit video streams (i.e., they periodically send a sequence of images) to a central station (i.e., the sink). Such end nodes have limited processing capability that only allows them to run simple motion detection algorithms, whereas the video streams are processed in a centralized way, as the sink stores and analyses the images. Such a sink can reside in a shared network slice with limited bandwidth. Consequently, a priority-based bandwidth management should prevent network congestion while satisfying the QoS requirements. Based on the monitored area, the end node with cameras send either low priority or high priority flows according to the importance of the video stream. Moreover, every time an end node detects something suspicious, an alarm flow is generated.

6. Implementation

This section presents an implementation of the proposed PrioSDN_RM. For this purpose we used:

- FloodLight. An open source (Apache-licensed) Java-based OpenFlow SDN controller, commonly used for research purposes [46]. The FloodLight architecture includes multiple modules that can be easily modified and improved.
- FlowVisor [47]. A network hypervisor for virtualizing software-defined networks, based on the OpenFlow protocol. FlowVisor allows both isolation and sharing of network slices [44].
- Zodiac FX. A small OpenFlow switch with an open source firmware. The Zodiac FX represents an excellent option for experimental purposes in research projects, as it is very cheap and it does not require complex setup or configuration.
- Raspberry Pi. A low cost single-board computer used as an end node.

Experimental Setup

A real testbed, shown in Figure 5, was implemented to show the feasibility of the proposed approach.

The implemented topology includes three nodes (Raspberry PIs), i.e., two senders and one receiver, and is partitioned into two slices. The link between the switch and the receiver node is shared between the slices. For each slice, a FloodLight controller (FL) is needed. The Zodiac FX OpenFlow switch is equipped with four Ethernet ports. Three of them are connected to the Raspberry PIs, while the other one is connected to a laptop, that runs both the hypervisor and the SDN controllers. In particular, FlowVisor runs in a virtual machine, while the FloodLight controllers operate on the host. The Zodiac FX switch communicates with the FloodLight controllers through FlowVisor.

The proposed priority-based admission control module, whose architecture is shown in Figure 3, was implemented as an extension of FloodLight. We disabled the forwarding module of Floodlight, as it automatically manages flow rules. This way, every time a message of a new flow arrives to a switch, the latter forwards the request to the SDN controller that includes the PAC module. The latter processes the request and decides whether to accept or prevent the flow transmission according to the previously defined policies. If the transmission is allowed, a flow rule is inserted in the relevant switch flow table.

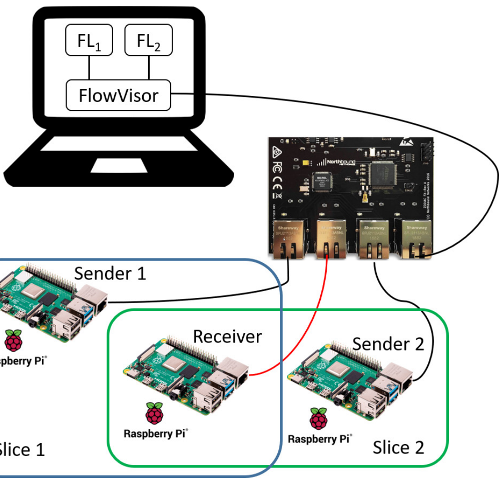

Figure 5. The testbed setup.

IEEE 802.1Q frames were used for the communications in the implemented testbed. The IEEE 802.1Q frame format, shown in Figure 6, includes four additional bytes compared to the Ethernet frame format. In particular, it adds the Virtual Local Area Network (VLAN) tag, that is placed between the Destination/Source MAC addresses and the Length/Type fields of an Ethernet frame to identify the VLAN to which the frame belongs.

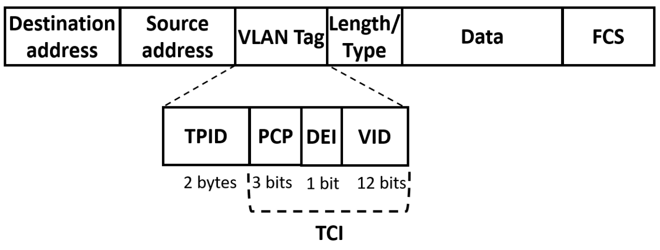

Figure 6. IEEE 802.1Q frame.

The first two bytes of the VLAN tag consists of the Tag Protocol Identifier (TPID) and indicates the frame type, e.g., the value 0 $\times$ 8100 indicates an IEEE 802.1Q frame. The other two bytes are used for the Tag control information (TCI) that, in turn, contains the Priority code point (PCP), Drop eligible indicator (DEI), and VLAN identifier (VID). The PCP is a 3-bit field (its value is in the range from 0 to 7) that refers to the class of service and maps to the frame priority level. Different PCP values can be used to prioritize different traffic classes. A higher value indicates a higher priority. The DEI allows to mark the frames that are eligible to be dropped in case of congestion. Finally, the VID indicates the VLAN to which a frame belongs.

The senders periodically generate and send messages belonging to different flows, each one with a specific priority. The periodic flows of each sender are shown in Table 1. The bit rate (and therefore the required bandwidth) is encoded in all the messages that are sent to the FloodLight controller. Moreover, sporadic alarm flows (i.e., flows whose messages have PCP equal to 7) can be generated by the senders. Each sender can generate multiple alarm flows at the same time. The receiver creates logs to record statistics about the received messages.

We consider a static partitioning of the bandwidth among the slices, as they have the same workload. The admission control parameters for the shared link (sl) were set as follows. The bandwidth limit (BW_{lim}^{sl}) for both slices was set to 200 bps, while the high priority (θ_{hp}^{sl}) and alarm (θ_{alarm}^{sl}) thresholds were initialized to 70% and 90% of the BW_{lim}, i.e., to 140 bps and 180 bps, respectively. The priority level ($prio_{lev}$) was set to 3.

Table 1. Network slice periodic flows.

Flow ID	Bit Rate (Bit/s)	PCP
1	90	1
2	25	1
3	90	2
4	30	5
5	40	0
6	50	4

7. Performance Evaluation

In the implemented network, we evaluated the performance in terms of throughput and Frame Drop Ratio (FDR) measured on the shared link. The FDR is a measure of the number of frame transmission requests dropped compared to the total number of requests. It is expressed as a percentage according to Equation (1),

$$FDR = \left(\frac{n_{dropFrm}}{n_{totFrm}}\right) \times 100 = \left(1 - \frac{n_{accFrm}}{n_{totFrm}}\right) \times 100 \tag{1}$$

where n_{totFrm}, $n_{dropFrm}$ and n_{accFrm} are the number of total, dropped and accepted frame transmission requests, respectively, measured on the shared port that connects the switch to the receiver.

We also evaluated the overhead introduced by the admission control.

7.1. Throughput

The throughput was measured on the receiver side, i.e., considering the shared link between the switch and the receiver, with and without PrioSDN_RM for both slices 1 and 2.

If the PAC module is disabled, the SDN controllers provide flow rules regardless of the flows priority, the bandwidth requirements and the bandwidth limits. Consequently, as shown in Figure 7, if we sum the bandwidth used by both slice 1 and slice 2, the throughput of the shared link grows up to the bandwidth saturation (400 bps) in about 30 s.

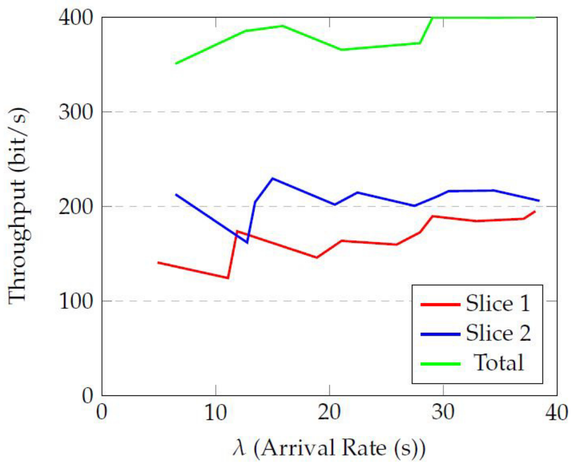

Figure 7. Throughput of both slices without admission control.

The same experiment was performed with the PAC module enabled. The results are shown in Figures 8, where the bandwidth limit and the thresholds are also depicted by horizontal lines. The results prove that the PAC module successfully controls the bandwidth utilization. If fact, in both slices the throughput does not exceed the bandwidth limit, fixed on 200 bps.

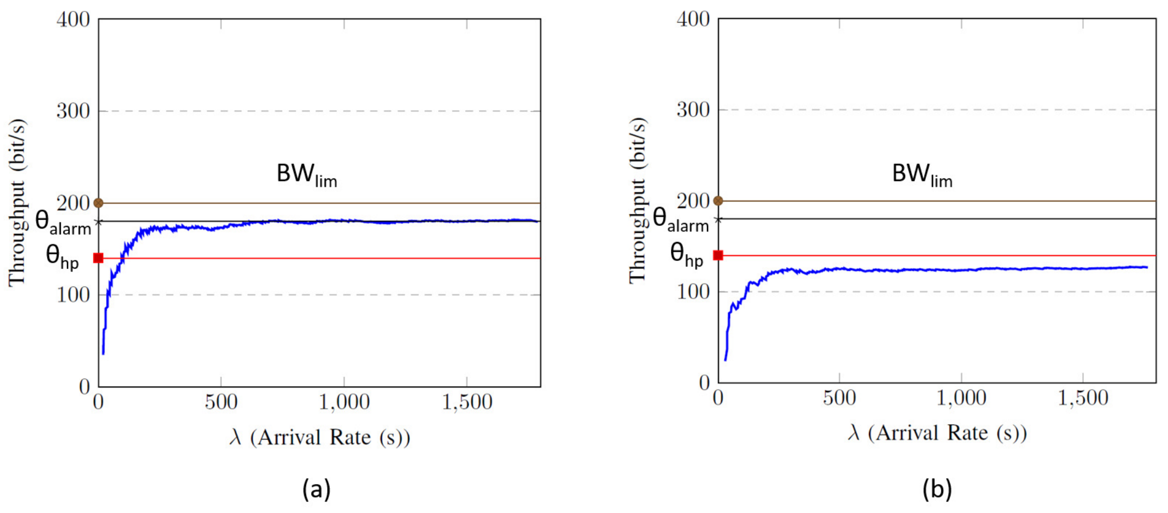

Figure 8. Throughput with PrioSDN_RM; (**a**) slice 1; (**b**) slice 2.

In particular, in the slice 1 the throughput went over the high priority threshold and sometimes over the alarm threshold, but it was always below the bandwidth limit. In the slice 2 the throughput remained below the high priority threshold, and therefore below the bandwidth limit.

Please note that the different values of throughput for the slice 1 and 2 depend on the internal scheduling of the flows in the sender nodes. In fact, the application layers of the senders are not synchronized, and therefore, the arrival time of the flows transmission requests is different. Consequently, the processing order at the admission control varies and the resulting allocation decisions are not the same. For instance, the throughput trend in Figure 8b is due to the fact that the high priority flows arrive first and receive all the bandwidth they require. Afterwards, the low priority flows arrive, but some of them cannot be transmitted, as this would imply exceeding the θ_{hp}^{sl}.

7.2. Frame Drop Ratio

The frame drop ratio (FDR) on the shared link was obtained collecting data during 20 min long experiments. In particular we evaluated the FDR for slices 1 and 2 with both a plain SDN controller (i.e., without admission control) and the proposed PrioSDN_RM. When the plain SDN controller without admission control was used, due to the overload of the network links, frames of any priority were lost. No significant difference between the number of dropped high priority frames and low priority ones was found, as all the flows were handled in the same way.

Tables 2 and 3 show the FDR for slices 1 and 2, respectively, obtained with the PrioSDN_RM. In particular, two configurations were evaluated. In the first configuration (a) the flow rule timeout (fr_{tout}) was set to 40 s for each flow. In the second configuration (b) the flow rule timeout was set to 40 s for the high priority flows and to 30 s for the low priority ones.

Table 2. PrioSDN_RM: Frame drop ratio-Slice 1.

PCP	**Configuration a**		**Configuration b**	
	FDR	**Timeout (s)**	**FDR**	**Timeout (s)**
1	64%	40	43%	30
1	33%	40	30%	30
2	70%	40	74%	30
5	28%	40	22%	40
0	89%	40	76%	30
4	35%	40	24%	40
7	0%	40	0%	40

Table 3. PrioSDN_RM: Frame drop ratio-Slice 2.

PCP	Configuration a		Configuration b	
	FDR	**Timeout (s)**	**FDR**	**Timeout (s)**
1	52%	40	47%	30
1	58%	40	26%	30
2	84%	40	74%	30
5	29%	40	22%	40
0	100%	40	82%	30
4	60%	40	27%	40
7	0%	40	0%	40

The results demonstrate that the use of the PrioSDN_RM significantly improves the FDR of both alarm and high priority flows comparing with the same scenario without admission control, in which the drop ratio was between 44% and 47% for alarms and between 47% and 53% for high priority flows, respectively. Conversely, the obtained FDRs for the low priority flows is worse than the ones obtained without admission control (i.e., between 49% and 55%), as PrioSDN_RM strongly favors alarm and high priority flows. As we expected, the frame drop ratio of the alarm flow packet is 0%, since a fixed amount of bandwidth is reserved for new alarm flows at any time. The comparison between the FDR obtained in the two different configurations confirmed that, by setting a lower flow rule timeout value for the low priority flows, in configuration b we improve the performance of high priority ones, as discussed in Section 4.3. Moreover, the more frequent evaluation of the current network state allows the PAC module to perform a fairer distribution of the bandwidth among the low priority flows. As it was expected, the lower the flow rule timeout values, the higher the QoS offered by the PAC module.

7.3. Priority-Based Admission Control Overhead

In this section we present the experimental assessment of the delay introduced by the proposed priority-based admission control, that is based on a virtualized SDN-based network. Such a delay was calculated as the cumulative delay introduced by both the hypervisor (i.e., FlowVisor) and the SDN controller (i.e., FloodLight). In particular we assessed the latency experienced when a message cannot be directly forwarded by the switch as the latter does not find the relevant flow rule in its flow table. In the other case, our approach does not introduce a significant delay. As discussed in [43] FlowVisor does not add overhead to the data plane, whereas it adds overhead to the tasks that involve both the control and data plane. In particular, FlowVisor increases the port status response latency by about 0.71 ms when the OpenFlow port status is requested.

The overhead introduced by our approach was obtained as the time difference between the instants of the bandwidth allocation request from the switch and of the reception of the relevant admission control response at the switch, respectively.

We performed 20 min long experiments. The obtained results showed that the delay introduced is always less than 10 ms.

Please note that for each flow the SDN controller processes only the first message plus each message sent after the flow rule expires. The other messages are handled only by the switch. Consequently, the flow rule timeout values impact on the overall overhead introduced by the proposed mechanism: The lower the flow rule timeout values, the greater the delay introduced (as the SDN controller will perform the priority-based admission control task more frequently), and vice versa.

8. Comparative Assessment with the Relevant Mechanisms in the Literature

This section discusses the PrioSDN_RM performance (in terms of throughput, frame drop ratio and overhead) in comparison with that of related mechanisms in the literature, i.e., the mechanism used in [22] and the DART approach in [45].

We considered two different configurations, in terms of periodic flows, in the scenario shown in Figure 5 in order to show how different workloads affect the performance of the three mechanisms. Sporadic alarm flows can be generated by the senders. Each alarm can require 15 bps at maximum. Each sender can generate multiple alarm flows at the same time. The bandwidth limit for both slices was set to 200 bps. We assume that the high priority traffic consists of messages with a priority higher than 2.

The high priority and alarm thresholds of the PrioSDN_RM here proposed were initialized to 150 bps and 185 bps, respectively. The priority level was set to 3. For the sake of comparing the mechanisms under similar operating conditions, the threshold of the approach proposed in [22] was set to 160 bps, while the priority level was set to 3. The priority limit in DART [45] was set to 2 and it is moved to 3 when an alarm occurs.

8.1. Configuration A

The periodic flows of each sender in the Configuration A are shown in Table 4. Such a configuration consists of a set of periodic flows with different priorities. The amount of high priority traffic is similar to the low priority one.

Table 4. Configuration A: Network slice periodic flows.

Flow ID	Bit Rate (Bit/s)	PCP
1	30	0
2	40	1
3	30	1
4	40	2
5	30	2
6	20	3
7	25	4
8	20	5
9	30	5
10	20	6

The same experiments were performed with the three mechanisms. Figures 9–11 show the throughput trend for both slices 1 and 2 using the PrioSDN_RM, the approach in [22] and DART [45], respectively.

In these conditions, PrioSDN_RM obtained an average throughput that is slightly lower than the one of the approach in [22], due to the unexploited bandwidth that, in our approach, is left out for handling new alarm flows. As far as DART is concerned, all the low priority messages (i.e., messages with priority lower than the current priority limit) are dropped, whereas PrioSDN_RM is more flexible, as it accepts allocation requests for low priority flows (i.e., it sends positive responses) provided that there is enough bandwidth available for them. Consequently, as shown in Figure 11, the throughput obtained using DART is lower than the one of the other approaches.

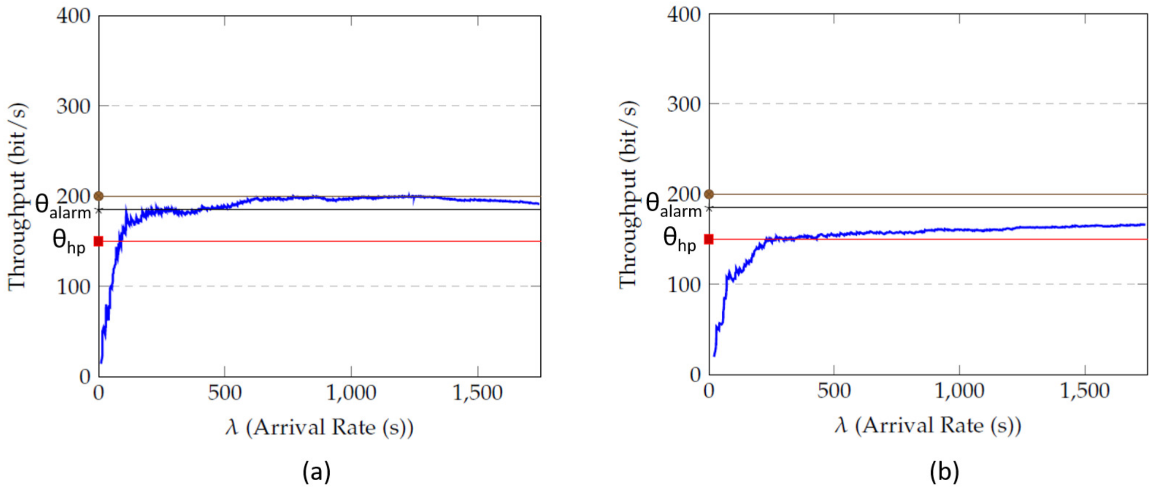

Figure 9. Configuration A: Throughput with PrioSDN_RM; (**a**) slice 1; (**b**) slice 2.

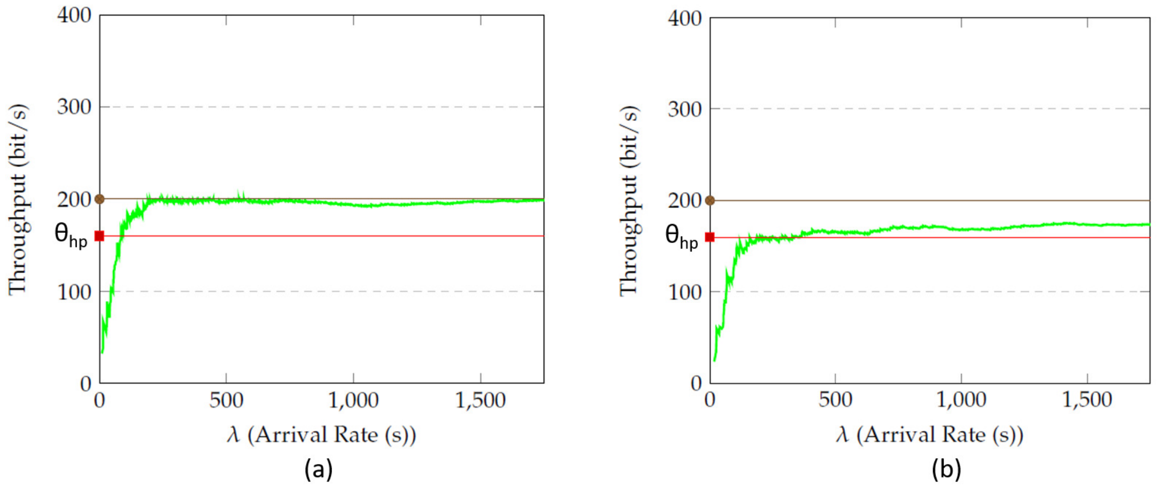

Figure 10. Configuration A: Throughput with the approach in [22]; (**a**) slice 1; (**b**) slice 2.

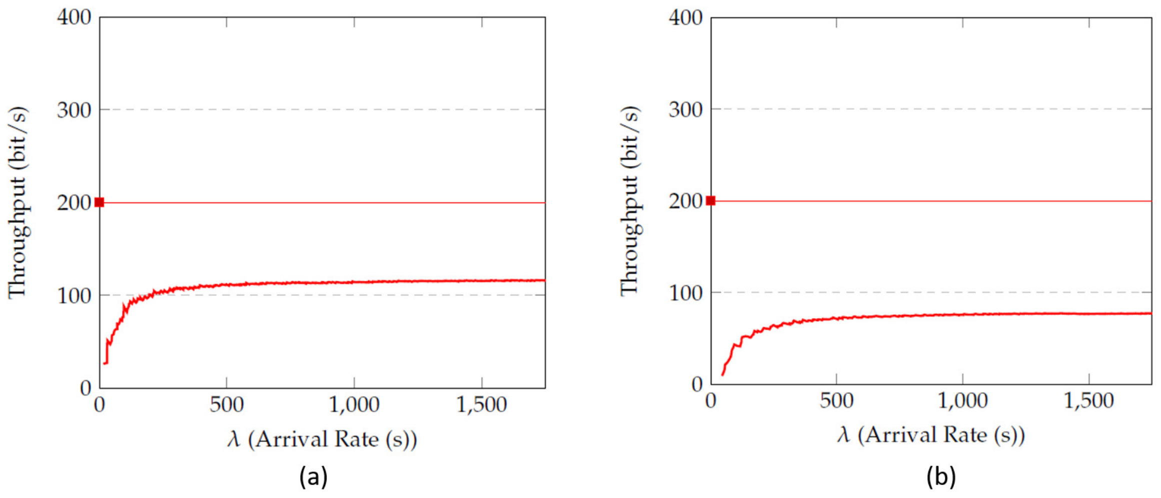

Figure 11. Configuration A: Throughput with DART [45]; (**a**) slice 1; (**b**) slice 2.

Table 5 shows the FDR for slices 1 and 2 obtained with the three approaches compared here.

Table 5. Configuration A: Frame drop ratio.

Flow ID	PCP	FDR-Slice 1			FDR-Slice 2		
		PrioSDN_RM	**Mechanism in [22]**	**DART [45]**	**PrioSDN_RM**	**Mechanism in [22]**	**DART [45]**
1	0	60%	65%	100%	55%	70%	100%
2	1	88%	91%	100%	93%	86%	100%
3	1	86%	61%	100%	81%	69%	100%
4	2	83%	72%	100%	88%	70%	100%
5	2	70%	46%	100%	86%	59%	100%
6	3	0%	9%	72%	4.2%	5%	69%
7	4	6%	30%	0%	4%	12%	0%
8	5	0%	0%	0%	0%	13%	0%
9	5	0%	0%	0%	6.6%	6%	0%
10	6	2%	0%	0%	2%	0%	0%
Alarms	7	0%	15%	0%	0%	14%	0%

As far as the number of dropped messages is concerned, the PrioSDN_RM and the approach in [22] present a similar FDR for both low priority and high priority flows. Conversely, the approach in [22] experiences a higher number of dropped alarm messages, as that approach does not reserve bandwidth for them, while the PrioSDN_RM guarantees that no alarm messages are dropped. In DART all the low priority messages were dropped, while no high priority messages were dropped, as the amount of bandwidth available in each slice is enough to manage these flows. Moreover, although there is no explicit provision for alarm handling and there is no bandwidth reserved for them, no alarms were dropped thanks to the limited amount of bandwidth required by the high priority flows. Some messages with priority equal to 3 were dropped because the priority limit dynamically changes from 2 to 3, and vice versa.

8.2. Configuration B

Table 6 shows the periodic flows of each sender in the Configuration B. Such a configuration involves a higher amount of high priority traffic than the one in Configuration A.

Table 6. Configuration B: Network slice periodic flows.

Flow ID	Bit Rate (Bit/s)	PCP
1	30	0
2	40	1
3	40	2
4	30	2
5	30	3
6	40	4
7	30	5
8	40	5
9	30	6
10	50	6

The throughput measured with the Configuration B with the PrioSDN_RM, the approach in [22] and DART [45] is depicted in Figures 12–14, respectively.

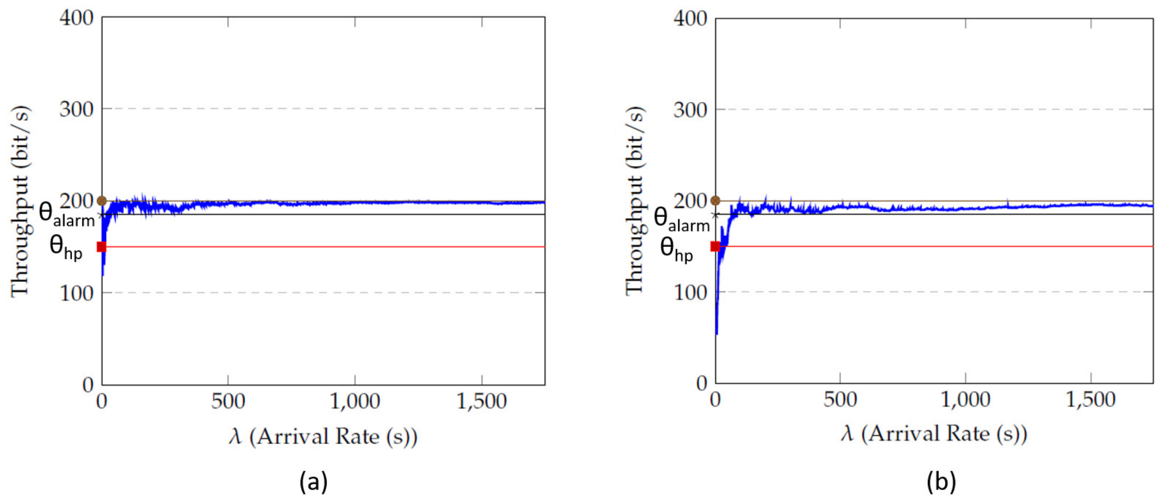

Figure 12. Configuration B: Throughput with PrioSDN_RM; (**a**) slice 1; (**b**) slice 2.

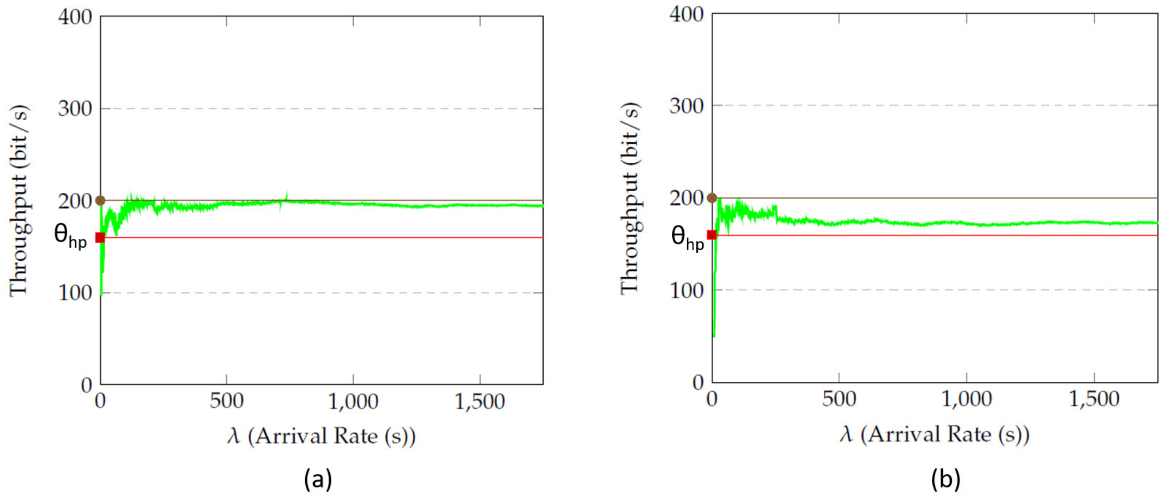

Figure 13. Configuration B: Throughput with the approach in [22]; (**a**) slice 1; (**b**) slice 2.

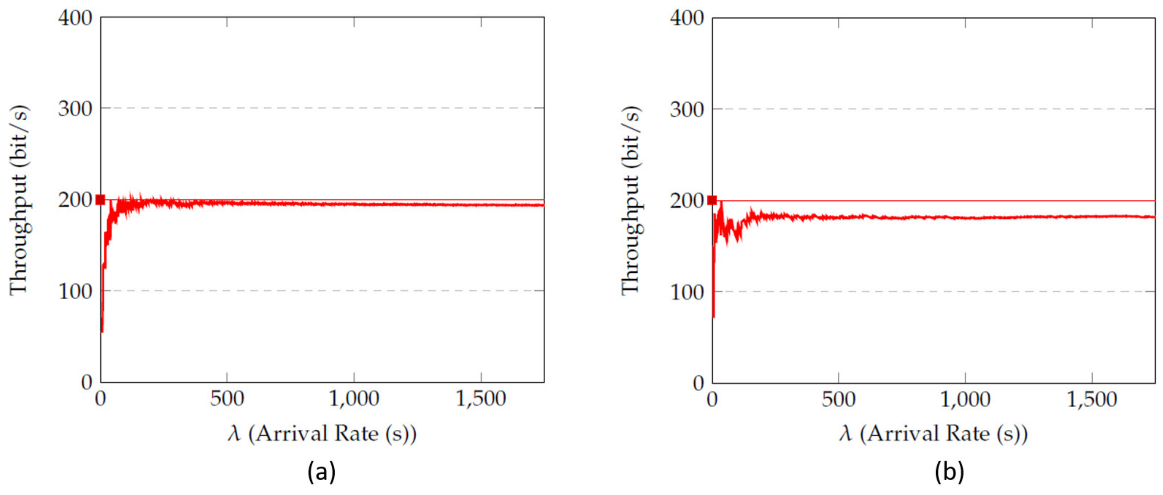

Figure 14. Configuration B: Throughput with DART [45]; (**a**) slice 1; (**b**) slice 2.

In these conditions, the three approaches obtained similar results in terms of throughput due to the significant amount of high priority traffic.

The FDR for slices 1 and 2 obtained with the three approaches is shown in Table 7.

Table 7. Configuration B: Frame drop ratio.

Flow ID	PCP	FDR-Slice 1			FDR-Slice 2		
		PrioSDN_RM	**Mechanism in [22]**	**DART [45]**	**PrioSDN_RM**	**Mechanism in [22]**	**DART [45]**
1	0	84%	85%	100%	85%	83%	100%
2	1	97%	96%	100%	90%	95%	100%
3	2	90%	94%	100%	85%	91%	100%
4	2	91%	95%	100%	92%	94%	100%
5	3	59%	57%	93%	53%	59%	94%
6	4	66%	78%	45%	71%	76%	43%
7	5	18%	34%	32%	25%	41%	27%
8	5	47%	27%	27%	50%	23%	29%
9	6	44%	31%	31%	15%	36%	31%
10	6	13%	45%	37%	19%	42%	34%
Alarms	7	0%	34%	47%	0%	35%	50%

The results in Table 7 show that both the PrioSDN_RM and the approach in [22] present a slightly better FDR for the low priority flows than DART, as the latter drops all the low priority messages. Moreover, our approach guarantees that no alarms are dropped, whereas the other two approaches dropped several alarms, as they do not reserve bandwidth for them. Actually, DART could be configured in different ways to avoid dropped alarms. For example, the priority limit could be set to a value higher than 3 when an alarm occurs. This way, all the messages with a priority below the priority limit (even the high priority messages) would be dropped, whereas the flows with a priority higher than the priority limit would experience an FDR equal to 0%. DART is therefore a very interesting pass/fail mechanism. However, tuning of the priority limit in DART is a critical task under high traffic load.

In terms of overhead, no notable differences between the three mechanisms under consideration are found, as in both cases the delays introduced depend on the admission control implemented in the SDN controller. The obtained results showed that the delay introduced by the three mechanisms is always less than 10 ms.

9. Conclusions

The combination of Software-Defined Networking and Network Virtualization allows to reduce the network management complexity. In particular, the ability to handle multiple logically isolated virtual networks (slices) plays an important role in Industrial Internet of Things applications, as they need to cope with the management of different domains, each one with specific QoS requirements. In this context, this paper presented the design and implementation of the PrioSDN_RM, a dynamic resource management mechanism for virtualized SDN networks in which multiple slices can share some network links. The PrioSDN_RM is based on a priority-based admission control that is aware of the current network state and that exploits two thresholds that limit the portion of bandwidth reserved for the different kinds of flows, i.e., low priority, high priority, and alarms. Experimental evaluations on a real testbed showed the effectiveness of the approach. Comparative assessments with similar mechanisms in the literature, i.e., the one proposed in [22] and DART [45], highlighted that the PrioSDN_RM combines the ability to guarantee that no alarm flow will be dropped with flexibility, as it tries to accommodate as many flows as possible with the available bandwidth. In fact, the PrioSDN_RM can accept allocation requests for low priority flows provided that there is enough bandwidth available for them.

Future work will deal with an extensive performance evaluation of the proposed approach in multi-hop networks.

Author Contributions: Conceptualization, L.L. and S.A.; Investigation, L.L.; Methodology, L.L. and L.L.B.; Software, S.A.; Supervision, L.L.B.; Validation, L.L.; Writing—original draft, L.L.; Writing—review & editing, L.L.B. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Aslam, M.S.; Khan, A.; Atif, A.; Hassan, S.A.; Mahmood, A.; Qureshi, H.K.; Gidlund, M. Exploring Multi-Hop LoRa for Green Smart Cities. *IEEE Netw.* **2020**, *34*, 225–231. [CrossRef]
2. Arasteh, H.; Hosseinnezhad, V.; Loia, V.; Tommasetti, A.; Troisi, O.; Shafie-khah, M.; Siano, P. Iot-based smart cities: A survey. In Proceedings of the 2016 IEEE 16th International Conference on Environment and Electrical Engineering (EEEIC), Florence, Italy, 7–10 June 2016; IEEE: Piscataway, NJ, USA, 2016; pp. 1–6.
3. Qian, Y.; Wu, D.; Bao, W.; Lorenz, P. The Internet of Things for Smart Cities: Technologies and Applications. *IEEE Netw.* **2019**, *33*, 4–5. [CrossRef]
4. Stojkoska, B.L.R.; Trivodaliev, K.V. A review of Internet of Things for smart home: Challenges and solutions. *J. Clean. Prod.* **2017**, *140*, 1454–1464. [CrossRef]
5. Iannizzotto, G.; Lo Bello, L.; Nucita, A.; Grasso, G.M. A Vision and Speech Enabled, Customizable, Virtual Assistant for Smart Environments. In Proceedings of the 2018 11th International Conference on Human System Interaction (HSI), Gdansk, Poland, 4–6 July 2018; pp. 50–56.
6. Kabalci, Y.; Kabalci, E.; Padmanaban, S.; Holm-Nielsen, J.B.; Blaabjerg, F. Internet of Things Applications as Energy Internet in Smart Grids and Smart Environments. *Electronics* **2019**, *8*, 972. [CrossRef]
7. Simoens, P.; Dragone, M.; Saffiotti, A. The Internet of Robotic Things: A review of concept, added value and applications. *Int. J. Adv. Robot. Syst.* **2018**, *15*, 1729881418759424. doi:10.1177/1729881418759424. [CrossRef]
8. Patti, G.; Leonardi, L.; Lo Bello, L. A Novel MAC Protocol for Low Datarate Cooperative Mobile Robot Teams. *Electronics* **2020**, *9*, 235. [CrossRef]
9. Ansari, S.; Aslam, T.; Poncela, J.; Otero, P.; Ansari, A. Internet of Things-Based Healthcare Applications. In *IoT Architectures, Models, and Platforms for Smart City Applications*; IGI Global: Hershey, PA, USA, 2020; pp. 1–28.
10. Catarinucci, L.; De Donno, D.; Mainetti, L.; Palano, L.; Patrono, L.; Stefanizzi, M.L.; Tarricone, L. An IoT-aware architecture for smart healthcare systems. *IEEE Internet Things J.* **2015**, *2*, 515–526. [CrossRef]
11. Leonardi, L.; Lo Bello, L.; Battaglia, F.; Patti, G. Comparative Assessment of the LoRaWAN Medium Access Control Protocols for IoT: Does Listen before Talk Perform Better than ALOHA? *Electronics* **2020**, *9*, 553. [CrossRef]
12. Pasetti, M.; Ferrari, P.; Silva, D.R.C.; Silva, I.; Sisinni, E. On the Use of LoRaWAN for the Monitoring and Control of Distributed Energy Resources in a Smart Campus. *Appl. Sci.* **2020**, *10*, 320. [CrossRef]
13. Wan, J.; Tang, S.; Shu, Z.; Li, D.; Wang, S.; Imran, M.; Vasilakos, A.V. Software-Defined Industrial Internet of Things in the Context of Industry 4.0. *IEEE Sens. J.* **2016**, *16*, 7373–7380. [CrossRef]
14. Sisinni, E.; Ferrari, P.; Fernandes Carvalho, D.; Rinaldi, S.; Marco, P.; Flammini, A.; Depari, A. LoRaWAN Range Extender for Industrial IoT. *IEEE Trans. Ind. Inform.* **2020**, *16*, 5607–5616. [CrossRef]
15. Luvisotto, M.; Tramarin, F.; Vangelista, L.; Vitturi, S. On the Use of LoRaWAN for Indoor Industrial IoT Applications. *Wirel. Commun. Mob. Comput.* **2018**, *2018*, 1–11. [CrossRef]
16. Leonardi, L.; Ashjaei, M.; Fotouhi, H.; Lo Bello, L. A Proposal Towards Software-Defined Management of Heterogeneous Virtualized Industrial Networks. In Proceedings of the IEEE 17th International Conference on Industrial Informatics (INDIN), Helsinki, Finland, 22–25 July 2019. [CrossRef]
17. Lucas-Estañ, M.C.; Raptis, T.P.; Sepulcre, M.; Passarella, A.; Regueiro, C.; Lazaro, O. A software defined hierarchical communication and data management architecture for industry 4.0. In Proceedings of the 2018 14th Annual Conference on Wireless On-Demand Network Systems and Services (WONS), Isola, France, 6–8 February 2018; pp. 37–44.
18. Wan, M.; Yao, J.; Jing, Y.; Jin, X. Event-based Anomaly Detection for Non-public Industrial Communication Protocols in SDN-based Control Systems. *Comput. Mater. Contin.* **2018**, *55*, 447–463.
19. Wang, J.; Yang, Y.; Wang, T.; Sherratt, R.S.; Zhang, J. Big Data Service Architecture: A Survey. *J. Internet Technol.* **2020**, *21*, 393–405.

20. Zhang, J.; Zhong, S.; Wang, T.; Chao, H.C.; Wang, J. Blockchain-based systems and applications: A survey. *J. Internet Technol.* **2020**, *21*, 1–14.
21. Liu, P.; Wang, X.; Chaudhry, S.; Javeed, K.; Ma, Y.; Collier, M. Secure video streaming with lightweight cipher PRESENT in an SDN testbed. *Comput. Mater. Contin.* **2018**, *57*, 353–363. [CrossRef]
22. Aglianò, S.; Ashjaei, M.; Behnam, M.; Lo Bello, L. Resource management and control in virtualized SDN networks. In Proceedings of the 2018 Real-Time and Embedded Systems and Technologies (RTEST), Tehran, Iran, 9–10 May 2018; pp. 47–53. [CrossRef]
23. Kreutz, D.; Ramos, F.M.V.; Veríssimo, P.E.; Rothenberg, C.E.; Azodolmolky, S.; Uhlig, S. Software-Defined Networking: A Comprehensive Survey. *Proc. IEEE* **2015**, *103*, 14–76. [CrossRef]
24. Blenk, A.; Basta, A.; Reisslein, M.; Kellerer, W. Survey on Network Virtualization Hypervisors for Software Defined Networking. *IEEE Commun. Surv. Tutor.* **2016**, *18*, 655–685. [CrossRef]
25. Sherwood, R.; Chan, M.; Covington, A.; Gibb, G.; Flajslik, M.; Handigol, N.; Huang, T.Y.; Kazemian, P.; Kobayashi, M.; Naous, J.; et al. Carving Research Slices out of Your Production Networks with OpenFlow. *ACM Spec. Interest Group Data Commun. (SIGCOMM) Comput. Commun. Rev.* **2010**, *40*, 129–130. [CrossRef]
26. Alderisi, G.; Iannizzotto, G.; Lo Bello, L. Towards IEEE 802.1 Ethernet AVB for Advanced Driver Assistance Systems: A preliminary assessment. In Proceedings of the 2012 IEEE 17th International Conference on Emerging Technologies Factory Automation (ETFA), Krakow, Poland, 17–21 September 2012; pp. 1–4.
27. Zhao, L.; Pop, P.; Zheng, Z.; Li, Q. Timing Analysis of AVB Traffic in TSN Networks Using Network Calculus. In Proceedings of the 2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), Porto, Portugal, 11–13 April 2018; pp. 25–36.
28. Alderisi, G.; Patti, G.; Lo Bello, L. Introducing support for scheduled traffic over IEEE audio video bridging networks. In Proceedings of the 2013 IEEE 18th Conference on Emerging Technologies Factory Automation (ETFA), Cagliari, Italy, 10–13 September 2013; pp. 1–9. [CrossRef]
29. Ashjaei, M.; Patti, G.; Behnam, M.; Nolte, T.; Alderisi, G.; Lo Bello, L. Schedulability analysis of Ethernet Audio Video Bridging networks with scheduled traffic support. *Real-Time Syst.* **2017**, 53, 526-577. [CrossRef]
30. Ashjaei, M.; Girs, S. Dynamic Resource Distribution using SDN in Wireless Networks. In Proceedings of the 2020 IEEE International Conference on Industrial Technology (ICIT), Buenos Aires, Argentina, 26–28 February 2020; pp. 967–972.
31. Girs, S.; Ashiaei, M. Designing a Bandwidth Management Scheme for Heterogeneous Virtualized Networks. In Proceedings of the 2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA), Turin, Italy, 4–7 September 2018; Volume 1, pp. 1079–1082. [CrossRef]
32. Paliwal, M.; Shrimankar, D. Effective Resource Management in SDN Enabled Data Center Network Based on Traffic Demand. *IEEE Access* **2019**, *7*, 69698–69706. [CrossRef]
33. Trivisonno, R.; Guerzoni, R.; Vaishnavi, I.; Frimpong, A. Network Resource Management and QoS in SDN-Enabled 5G Systems. In Proceedings of the 2015 IEEE Global Communications Conference (GLOBECOM), San Diego, CA, USA, 6–10 December 2015; pp. 1–7. [CrossRef]
34. Lo Bello, L.; Lombardo, A.; Milardo, S.; Patti, G.; Reno, M. Experimental Assessments and Analysis of an SDN Framework to Integrate Mobility Management in Industrial Wireless Sensor Networks. *IEEE Trans. Ind. Inform.* **2020**, *16*, 5586–5595. [CrossRef]
35. Satija, S.; Sharma, T.; Bhushan, B. Innovative approach to Wireless Sensor Networks: SD-WSN. In Proceedings of the 2019 International Conference on Computing, Communication, and Intelligent Systems (ICCCIS), Greater Noida, India, 18–19 October 2019; pp. 170–175.
36. Jang, H.C.; Lin, J.T. Bandwidth management framework for smart homes using SDN: ISP perspective. *Int. J. Internet Protoc. Technol. (IJIPT)* **2019**, *12*, 110–120. [CrossRef]
37. Li, C.; Guo, W.; Wang, W.; Hu, W.; Xia, M. Programmable bandwidth management in software-defined EPON architecture. *Opt. Commun.* **2016**, *370*, 43–48. [CrossRef]
38. Chang, Y.; Chen, Y.; Chen, T.; Chen, J.; Chiu, S.; Chang, W. Software-Defined Dynamic Bandwidth Management. In Proceedings of the 2019 21st International Conference on Advanced Communication Technology (ICACT), PyeongChang, Korea, 17–20 February 2019; pp. 201–205. [CrossRef]
39. Jimson, E.R.; Nisar, K.; bin Ahmad Hijazi, M.H. Bandwidth management using software defined network and comparison of the throughput performance with traditional network. In Proceedings of the International Conference on Computer and Drone Applications (IConDA), Kuching, Malaysia, 9–11 November 2017; pp. 71–76. [CrossRef]

40. Becker, M.; Lu, Z.; Chen, D. An Adaptive Resource Provisioning Scheme for Industrial SDN Networks. In Proceedings of the 2019 IEEE 17th International Conference on Industrial Informatics (INDIN), Helsinki, Finland, 22–25 July 2019; Volume 1, pp. 877–880. [CrossRef]
41. Min, S.; Kim, S.; Lee, J.; Kim, B.; Hong, W.; Kong, J. Implementation of an OpenFlow network virtualization for multi-controller environment. In Proceedings of the 2012 14th International Conference on Advanced Communication Technology (ICACT), PyeongChang, Korea, 19–22 February 2012; pp. 589–592.
42. Priyadarsini, M.; Bera, P. A Secure Virtual Controller for Traffic Management in SDN. *IEEE Lett. Comput. Soc.* **2019**, *2*, 24–27. [CrossRef]
43. Chen, J.; Ma, Y.; Kuo, H.; Yang, C.; Hung, W. Software-Defined Network Virtualization Platform for Enterprise Network Resource Management. *IEEE Trans. Emerg. Top. Comput.* **2016**, *4*, 179–186. [CrossRef]
44. Mijumbi, R.; Serrat, J.; Rubio-Loyola, J.; Bouten, N.; Turck, F.D.; Latré, S. Dynamic resource management in SDN-based virtualized networks. In Proceedings of the 10th International Conference on Network and Service Management (CNSM) and Workshop, Rio de Janeiro, Brazil, 17–21 November 2014; pp. 412–417. [CrossRef]
45. Struhár, V.; Ashjaei, M.; Behnam, M.; Craciunas, S.S.; Papadopoulos, A.V. DART: Dynamic Bandwidth Distribution Framework for Virtualized Software Defined Networks. In Proceedings of the 45th Annual Conference of the IEEE Industrial Electronics Society (IECON), Lisbon, Portugal, 14–17 October 2019; Volume 1, pp. 2934–2939.
46. Asadollahi, S.; Goswami, B. Experimenting with scalability of floodlight controller in software defined networks. In Proceedings of the 2017 International Conference on Electrical, Electronics, Communication, Computer, and Optimization Techniques (ICEECCOT), Mysuru, India, 15–16 December 2017; pp. 288–292.
47. Sherwood, R.; Gibb, G.; kiong Yap, K.; Casado, M.; Mckeown, N.; Parulkar, G. *FlowVisor: A Network Virtualization Layer*; OpenFlow Switch Consortium, Technical Report; ETH Zürich: Zürich, Switzerland, 2009.