Chapter 2
Traffic Management in Networks with Programmable Data Planes

Davide Sanvito

2.1 Software-Defined Networks (SDN)

Traditional computer networks include vertically integrated devices running both the control plane (CP) and the data plane (DP). The former computes the forwarding decisions about how to handle the traffic, while the latter effectively process it. The heterogeneity of low-level vendor-proprietary configuration interfaces, together with the proliferation of specialized devices (middleboxes, e.g.. firewalls, network address translators, load balancers and deep packet inspection boxes), made networks complex and difficult to manage. The situation was further exacerbated by the need of quickly respond to network dynamics such as failures and changes in the traffic patterns, in the network topology and in the forwarding policies.

Software-Defined Networking (SDN) is a computer networking paradigm based on the decoupling of the control plane from the data plane. The control plane is logically centralized in an external node (called the network controller) which configures the network devices through a well-defined application programming interface (API). Network devices (e.g.. switches) become simple elements running just the data plane and forwarding the traffic according to the decisions taken by the external controller. Network operators can now directly operate on the global network view offered on top of the controller without designing complex distributed protocols to achieve a desired global network-wide behaviour. SDN brings to the networking domains all the software engineering best practices such as code modularity and reusability. SDN enables the reconfiguration of the network at software speed by simply running a different application on top of the controller. The controller is in charge of maintaining the global view of the network status and can even provide to the applications running above an abstract view of the network by limiting or transforming the observable

D. Sanvito (✉)
Dipartimento di Elettronica, Informazione e Bioingegneria (DEIB),
Politecnico di Milano, Piazza Leonardo da Vinci, 32, 20133 Milan, Italy
e-mail: davide.sanvito@polimi.it

A. Geraci (ed.), *Special Topics in Information Technology*,
PoliMI SpringerBriefs, https://doi.org/10.1007/978-3-030-62476-7_2

topology in order to provide isolation to different applications. At the same time the emergence of new network programming abstractions and high-level programming languages simplifies the network management and facilitates the evolution of the network. The augmented programmability brought by SDN enables a wide range of network applications ranging from traditional network functions (e.g.. switching and routing) to traffic engineering (e.g.. load balancing, QoS and fault tolerance), from network monitoring to security domains (e.g.. firewall, network access control, DoS attack mitigation).

OpenFlow [14] represents one of the most successful instances of the SDN principles. Despite the efforts towards an augmented network programmability started several years ago with Active Networks [10], the mainstream adoption of SDN principles took place only with OpenFlow. Thanks to its pragmatic compromise between the need for innovation from researchers and the need for closed platforms from the vendors, it became the de-facto standard programming interface. OpenFlow is nowadays supported by a large number of switches, both hardware and software, and SDN controllers. An OpenFlow switch exposes to the controller a pipeline of match-action tables. The *match-action* abstraction is a prominent example of programming abstractions provided by SDN. All the packets belonging to the same flow (defined by a set of packet field values, i.e. the *match*) are subject to the same treatment (i.e. the *action*, for example the forwarding or dropping of the packet or the modification of its header fields). The controller is in charge of configuring the intended network behaviour by filling the match-action tables with a set of flow entries. The match-action abstraction unifies different forwarding behaviours which can be quickly reconfigured at software speed. A device, for example, can be configured as a router if all the rules match on the IP destination address or as firewall if the rules match on Ethernet type, IP addresses and TCP ports. The controller can proactively provision the devices with flow rules and reactively add new ones whenever a packet does not match any of the available rules. The OpenFlow specification [4] defines the details of the format of the control channel messages exchanged among the control plane and the data plane and the set of header fields and packet actions a switch has to support.

Most recent advances in programmable network devices [8] take a step further and enable the network operator to program the packet parser and to define custom packet actions by combining primitives offered by the switching chip. Not only the switches' resources are more flexibly allocated and tailored to the traffic the network will deal with in terms of protocols, but the format itself of protocols (i.e. the set of header fields a switch can match on) is not fixed a priori from the chip vendor and can be customized by the network operator. The impacting factor for the success of these devices is that their increased flexibility does not come with penalties in the performance, cost and power compared to fixed-function chips. These devices are typically configured using an high-level language, such as P4 [7], and in principle a P4 program can fully describe an OpenFlow-enabled switch.

2.2 Control Plane Programmability

One of the prominent use cases enabled by SDN is the execution of Traffic Engineering (TE) algorithms on top of the controller. Traditional approaches from service providers design the routing considering the worst case traffic scenario. However, this leads to a network operating most of the time in sub-optimal conditions. SDN provides the needed flexibility to update the network more frequently, enabling online traffic optimization based on periodic traffic measurements and predictions in order to improve the network performance, reduce the operational costs and balance the utilization of network resources. The maximum achievable network reconfiguration rate is however limited by two aspects.

First of all, the changing nature of the traffic affects the optimality of the computed routing configuration. Despite the traffic pseudo-periodicity under some time scales due to ordinary daily fluctuations, an accurate traffic estimation is hard to achieve and, especially in case the traffic is significantly different from the expected scenarios, computing routing configurations which are too tied to specific traffic scenarios might lead to the congestion of the network or to unfeasible configurations. It is thus desirable to take into account some robustness considerations, by computing routing policies which are able to deal with multiple traffic scenarios under a same configuration.

The second limiting aspect is instead related to the low speed of flow programming in hardware. The transition across two network configurations is a critical procedure which might lead to broken connectivity, forwarding loops or violations of the forwarding policies. Ideally the network should be atomically updated, i.e. traffic should be forwarded either using the old configuration or the new one, and not some combination of the two. The decoupling of the control plane from the data plane makes however the set of the controller and the switches a complex asynchronous distributed system. Consistent network update mechanisms [15] are techniques able to deal with this problem by computing a set of intermediate network configurations to be sequentially scheduled to move the network from the current configuration state to a target configuration. These techniques ensure that in each intermediate configuration the consistency properties are guaranteed (i.e. the connectivity is preserved and either the old policy or the new one holds) while switch resources constraints are not violated (i.e. the additional flow rules installed must not exceed the memory available in the devices). Unluckily, these multi-steps mechanisms are not the only responsible for making the update process not atomic. A recent analysis on commercial SDN devices showed the existence of a mismatch between the switch-local control plane status reported to the centralized controller and the actual status of the data plane [13]. This discrepancy makes not completely trustworthy an information which is instead essential for the consistent update mechanisms to correctly schedule the various steps in the transition towards the final network configuration. Depending on the level of utilization of the flow tables and on the current load of the data plane,

the state of the data plane might fall behind the control plane from seconds up to several minutes. This means that the time required to properly complete each update step increases and this contributes to further decrease the achievable rate of updates.

2.2.1 Traffic Engineering Framework

The first contribution of the thesis is the design of a centralized SDN Traffic Engineering framework, CRR,[1] to decide whether, when and how to reconfigure the network. Given a set of of measured Traffic Matrices (TMs) over a given period (e.g.. one day), we defined an optimization model which computes a set of routing configurations to be proactively applied during the following period. Traffic matrices are clustered in the traffic, time and routing domains and we compute, for each cluster, a routing configuration which is robust against variations within the corresponding discrete traffic subspace defined by each cluster. First of all we take into consideration the traffic domain, i.e. we look at the values of the demands over time. We also considered the time domain, i.e. we avoid clustering together TMs not adjacent in time that would require too frequent network reconfigurations. Finally, we also took into account the ultimate effect of the routing on the TMs in terms of network congestion. We indeed include the same metric commonly adopted to optimize the routing (the network congestion, in terms of Maximum Link Utilization) to also guide the clustering logic. By tuning the size of clusters (i.e. the minimum number of members) and their number, the optimization model can explore the trade-off between static Traffic Engineering schemes (which compute a single routing configuration on the whole TMs set) and dynamic Traffic Engineering schemes (which instead keep reconfiguring the network each time a new TM measurement is available) while at the same time coping with the practical constraint of limiting the number of network reconfigurations and guaranteeing a minimum holding time for each configuration. Since the transitions from a routing configurations to the next one are not instantaneous even in SDN, adjacent clusters are allowed to partially overlap close to their boundaries (i.e. near routing transitions). This means that each routing configuration will be reasonably good also for a small number of traffic scenarios expected to be handled by the configuration of adjacent clusters. As a side effect, overlaps help the potentially slow multi-step consistent network update mechanisms. The analysis of the influence of errors in traffic predictions showed an interesting trade-off between the cluster length and the prediction accuracy. If the quality of the prediction is good, we can afford to have a larger number of short clusters whose routing is more tailored to specific scenarios. As the prediction quality decreases, it is better to resort to less and larger clusters which are able to better deal with a large variety of scenarios (i.e. they are more robust to traffic uncertainty). This opens a promising research direction where the controller plays an active role in measuring and predicting the

[1] Clustered Robust Routing.

traffic evolution and in estimating a-posteriori the quality of the prediction in order to consequently tune the level of robustness provided by the routing configurations set. More details can be found in [16].

2.2.2 ONOS Intent Monitor and Reroute Service

We then evaluated how to integrate our Traffic Engineering framework, CRR, within a SDN platform. Open Network Operating System (ONOS) [5] is a production-ready open-source SDN network operating system built for Service Provider networks. ONOS provides high performance, scalability and availability thanks to its distributed core and proper abstractions to configure the network. Among the programming abstractions offered by ONOS, intents work at the highest level: developers can express high-level policies (i.e. "intentions") without worrying about how such behaviour is implemented in the network. For example, users can use a *Point-To-Point* intent to require the connectivity between a pair of nodes without providing any information on the path to be used. Intents can be tied to a specific traffic subset (providing a set of values for the packet header fields) and a treatment (a set of actions, for example packet header modifications, to be applied to all the packets the intent refers to). ONOS supports several types of intents and each one includes a compiler which enables the Intent Framework, the ONOS component in charge of handling intents, to translate the high-level policy described by the intent itself to the set of low-level rules to be installed in the network devices. The Intent Framework is also in charge of re-compiling the intents in case of topology changes (for example link failures) to fulfill the high-level policy transparently to the application submitting the intents.

Intents represent an interesting opportunity to integrate the CRR in ONOS because we can decouple the connectivity endpoints from the paths implementing the communication. An user can thus specify the sources and destinations of the traffic (i.e. the traffic matrix pairs) independently from when and how the CRR updates the corresponding paths in the network. The idea is to modify the low-level paths implementing the intents not only as a consequence of topology changes, but also considering changes in traffic statistics, according to the output of the CRR model. Even if the Intent Framework is designed to be extensible with additional intents and compilers, it individually compiles each intent based on its own information. The CRR aims instead at compiling together multiple intents to optimize a global network objective. In addition, integrating a computationally heavy component such as an optimization tool within the same machine running the controller can have a negative impact on ONOS's high performance requirements. We thus developed a new service, ONOS Intent Monitor and Reroute (IMR) service, which enhances the ONOS Intent Framework with an external plug&play routing logic running as an off-platform application (OPA), an application running in a separate process space with respect to the ONOS controller and communicating through REST APIs or gRPC. The IMR service, running within ONOS, is in charge of collecting from network

devices the statistics related to the flow rules implementing the intents and to export them to the OPA by means of a REST API. In turns, the CRR will send back to the IMR service the routing configurations as scheduled by the CRR model.

We envision the following scenario: (1) an ONOS application submits intents to request connectivity (implicitly defining the endpoints of the TMs) and requires the IMR service to monitor their statistics. (2) the ONOS Intent Framework initially routes intents on their shortest paths while, at the same time, the IMR starts the monitoring process. (3) the CRR module, running as an OPA, collects the statistics of the intents to build the set of TMs to be fed to the CRR algorithm. After a given period (e.g.. one day), the OPA solves the CRR, schedules the activation of the routing configurations during the following period and at the same time keeps collecting the statistics to be fed to the next optimization round.

In order to prevent IMR from limiting the Intent Framework in recovering from failures, the routing paths provided by the OPA are treated as soft constraints: in case the suggested paths are not available when requested or a failure happens afterwards, the Intent Framework resorts to the standard shortest path computation for all the intents affected by the failure.

The decoupling of the application submitting the intents from the routing logic allows to re-use a same logic for different ONOS applications or to switch different routing logics for a same ONOS application. The CRR indeed represents one of the possible instances of a routing algorithm running as OPA and the IMR service implements a more general framework to interconnect any ONOS intent-based application to an external plug&play routing logic. This enables to re-use existing Traffic Engineering tools or develop new schemes based either on optimization tools or on Artificial Intelligence and Machine Learning. More details can be found in [18]. Our IMR service has been integrated in ONOS Nightingale version 1.13 as an official open-source contribution [3].

2.3 Data Plane Programmability

Despite the robust nature of each routing configuration, significant traffic deviations from the expected scenarios, such as network failures and congestion, need a proper handling to keep the network operating. Albeit the great speed in innovation and flexibility enabled by SDN, one of the main limitations introduced by its two-tier architecture is the strict decoupling of the data plane from the control plane. Network devices are indeed dummy devices unable to modify their forwarding behaviour without relying on the external controller, even if such changes depends entirely on events which are locally observable. This rigid separation of concerns prevents *a priori* the implementation of applications which require a prompt reaction, due to the intrinsic control channel latency and the processing overhead at the controller. Examples of such applications range from the security domain (e.g.. DDoS attacks mitigation) to the network resiliency domain (e.g.. detection and recovery of network failures) up to Traffic Engineering (e.g.. load balancing and congestion control). In

addition, most of the operations supported within the data plane are stateless: each packet is forwarded according to the matching rule without any notion of the past history of the flow it belongs to. Applications which need to keep per-flow states (e.g.. load balancing, NAT or stateful firewall) have to rely on the external controller to support a state-dependent forwarding.

Recent research efforts tried to deal with these limitations by delegating part of the control back to network devices to enable a self-adaptation of the forwarding behaviour. In the context of the European H2020 project BEBA [1], I've contributed to the software prototyping of a stateful extension[2] to OpenFlow and to the design of some use case applications. Open Packet Processor (OPP) [6] enables stateful packet processing in the data plane with a programming abstraction based on Extended Finite State Machines (EFSM). Flows are associated with a persistent context which includes a state and few data variables. Packets are forwarded not only based on their header but also on the persistent state of the flow they belongs to. Each state determines a forwarding policy and transitions among states are triggered, directly in the fast path, according to time-based or packet-based events and conditions evaluated over the packet header and its context. The controller can configure the stateful packet processing in the data plane by providing the set of header fields which defines the flow, i.e. the entity for which an application needs to keep a state, and the architecture of EFSM, in terms of its transitions and its state-dependent forwarding policies. In turns the switch is able to autonomously instantiate at runtime per-flow instances of the state machine without involving the external controller. For those applications where the forwarding evolution depends only on local information, OPP can provide a significantly more scalable and faster solution compared to centralized approaches.

2.3.1 Network Failures

As discussed in Sect. 2.2.2, the ONOS Intent Framework is able to re-actively recover from failures transparently to the application which submits the intents. Even if those unplanned failure scenarios are not taken into account by the CRR during the optimization phase, the Intent Framework guarantees that connectivity is preserved, although in a potentially unoptimized fashion. In order to improve on this situation, it is possible to pre-compute backup path policies on top of each one of the Robust Routing configurations for different failure scenarios and proactively configure them in the network devices.

By exploiting, for example, the OpenFlow Fast-Failover mechanism it is possible to define an alternative forwarding policy to be activated by the switch itself when it detects a failure in the link associated to the current policy. This enables a more prompt reaction thanks to the avoidance of the external controller. The failure detection

[2]Our stateful extension to OpenFlow has been integrated as official open-source contribution [2] to *ofsoftswitch13* [11], an user-space open-source software switch widely used in the research community.

mechanism is however external to the OpenFlow specification without any guarantee on the detection delay and many existing solutions are based on the slow path. In addition, depending on the specific topology and on the computed backup policies, the alternative path for a specific failure might not be available locally to the node which detects the failure. These *remote* failure scenarios still require the intervention of the controller making challenging achieving carrier-grade recovery times.

We designed a scheme, SPIDER,[3] based on the advanced capabilites of stateful data planes, such as OPP, to offload to the data plane both the detection and the recovery of network failures even in the case of remote failures. SPIDER is inspired by Bidirectional Forwarding Detection (BFD) and MPLS Fast Reroute technologies and provides an end-to-end proactive protection to failures independent from controller reachability and with a guaranteed sub-milliseconds detection delay. SPIDER guarantees zero losses after the detection regardless the availability of the controller and for both local and remote failure scenarios. Schemes relying on the controller to activate the backup policy would instead have non-zero losses also during the recovery phase while waiting for its intervention.

Data packets are tagged with different values to select the proper forwarding behaviour (i.e. primary path or designated backup path) and, at the same time, to drive the evolution of the state machines. Tagged data packets are indeed used to implement the two mechanisms for the detection of the failures and their recovery (i.e. the activation of the alternative backup policies) with a fully configurable trade-off between the overhead and failover responsiveness. In addition, in the remote failure scenarios, tagged data packets are also used as an in-band signalling scheme able to trigger a state transition for state machines stored in a device distinct from the one detecting the failure itself. SPIDER is able to handle all the pre-planned single failure scenarios from the data plane. Multiple failures involving a same demand require instead the intervention of the external controller. More details can be found in [9].

2.3.2 Network Congestion

The second network scenario which challenges the strict decoupling of the data plane from the control plane is the network congestion. We focused here on data center networks for their unique characteristics in terms of topology and traffic. Data center networks typically present a multi-rooted tree topology such as Leaf-Spine or Fat-Tree to provide high bandwidth among servers under different racks and a high degree of resiliency. Inter-rack traffic is usually spread across a large pool of symmetric paths using Equal-Cost Multi-Path (ECMP). ECMP selects a path by computing a hash over the identifier of the flow (for example the transport-layer addresses and ports) so that all the packets of a same transport-level flow are consistently sent on the same path without creating out-of-order packets. The decision taken by ECMP is not

[3] Stateful Programmable Failure Detection and Recovery.

aware of the size of the flows and is agnostic to the congestion status of the paths, thus ECMP exhibits an ideal behaviour only when there is a large number of flows of comparable sizes with sufficient entropy across the headers [12]. In reality, traffic in data centers presents often a mice-elephant distribution in terms of flow sizes: there is a large number of small flows, but the largest quota of traffic, in terms of transmitted bytes, comes from a limited set of flows (the elephant flows). This is a problem for ECMP because it might happen that two or more large flows select the same downstream path, creating congestion. This situation affects both categories of flows: elephant flows do not get the bandwidth they might potentially achieve and at the same time they block smaller flows. It is important to quickly react to this condition especially to limit the impact on the mice flows which are typically latency sensitive. Elephant flows are instead more sensitive to the available bandwidth.

Once again, advanced stateful dataplanes offer interesting opportunities for the self-adapatation of the network, enabling a more scalable and prompt reaction compared to approaches relying on the external controller. We designed and implemented CEDRO,[4] an in-switch mechanism to detect and re-route large flows colliding on a same downstream path based on the stateful capabilities of OPP. By default inter-rack traffic is spread using standard ECMP. When CEDRO detects large flows experiencing congestion (i.e. using a path whose utilization is above a predefined threshold) it triggers their re-route on an alternative path by overriding the current choice of ECMP and forcing ECMP to select a new path without considering the current one in its pool. This new choice is permanently stored in the switch, just for those specific flows, thanks to a transition in their state machines. CEDRO can handle both local and remote congestion scenarios from the data plane without involving the external controller. The congestion scenario is *remote* when the switch detecting the congestion condition is not the same able to steer the traffic to an alternative path. For example in a Leaf-Spine topology this happens when two large flows coming from different Leaf switches are assigned to the same Spine switch and their traffic addresses the same Spine switch. By bouncing back tagged data packets we can realize an in-band signalling scheme to trigger a remote reaction in the Leaf switches from the Spine switch.

In this application, the ability to handle the congestion directly from the data plane is important for two reasons. The offloading of the detection improves the scalability because we avoid having the controller orchestrate the monitoring of the flow sizes and of the link utilizations. In addition, by offloading also the re-routing, CEDRO enables a quicker reaction because we avoid paying the control channel delay and processing at the controller which might constitute by themselves a non negligible quota of the lifetime of mice flows.

In summary, the Leaf-to-Leaf macroflow aggregate (i.e. the set of all the transport-layer microflows from a same Leaf switch to the other Leaf switches) is spread over the paths using ECMP and the choice of ECMP is overridden for a selected number of microflows. By paying a small penalty in the 99-th percentile and in the maximum Flow Completion Time (FCT), CEDRO enables to improve the average and 95-th

[4]Congested Elephant Detection and Re-routing Offloading.

percentile of the FCT compared to standard ECMP. Given the high number of mice flows, an improvement of the average metric is relevant for the latency-sensitive nature of such category of flows.

By integrating in CEDRO the in-switch failure detection capabilities of SPIDER, we can add a quick reaction to network failures which might be considered as an extreme case of congestion scenario. The resulting system would provide two levels of reaction. In case of network congestion the system re-routes to other paths just few flows from the rack-to-rack aggregate, while in case of a network failure the entire aggregate affected by the failure gets re-balanced over the remaining set of paths. More details can be found in [17].

2.4 Conclusions

In conclusion, in this thesis work we analyzed the network programmability opportunities for traffic management offered by the Software-Defined Networking paradigm at different layers. We started from the programmability of the control plane and exploited its global view to design a proactive and centralized Traffic Engineering framework to enable online traffic optimization based on periodic traffic measurements and predictions and showed how to integrate it in a production-ready SDN platform. In order to handle the unexpected scenarios which challenge the strict decoupling of the control plane from the data plane we designed and implemented two applications based on stateful extensions to OpenFlow. These applications complement the centralized and proactive approach based on the global state of the network with reactive distributed logic based on a partial local view of the network state and enabling a more prompt and scalable reaction compared to approaches based on the centralized control plane.

References

1. BEBA project. http://www.beba-project.eu/
2. ofsoftswitch13 official contribution GitHub repository. https://github.com/CPqD/ofsoftswitch13/tree/BEBA-EU
3. ONOS Wiki: IMR - Intent Monitor and Reroute service. https://wiki.onosproject.org/x/hoQgAQ
4. OpenFlow 1.5 specification. https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf
5. Berde P, Gerola M, Hart J, Higuchi Y, Kobayashi M, Koide T, Lantz B, O'Connor B, Radoslavov P, Snow W et al (2014) Onos: towards an open, distributed sdn os. In: Proceedings of the third workshop on Hot topics in software defined networking
6. Bianchi G, Bonola M, Pontarelli S, Sanvito D, Capone A, Cascone C (2016) Open packet processor: a programmable architecture for wire speed platform-independent stateful in-network processing. CoRR arXiv:abs/1605.01977

7. Bosshart P, Daly D, Gibb G, Izzard M, McKeown N, Rexford J, Schlesinger C, Talayco D, Vahdat A, Varghese G et al (2014) P4: Programming protocol-independent packet processors. ACM SIGCOMM Comput Commun Rev 44(3):87–95
8. Bosshart P, Gibb G, Kim H-S, Varghese G, McKeown N, Izzard M, Mujica F, Horowitz M (2013) Forwarding metamorphosis: Fast programmable match-action processing in hardware for sdn. ACM SIGCOMM Comput Commun Rev 43(4):99–110
9. Cascone C, Sanvito D, Pollini L, Capone A, Sanso B (2017) Fast failure detection and recovery in sdn with stateful data plane. Int J Netw Manag 27(2):e1957
10. Feamster N, Rexford J, Zegura E (2013) The road to sdn. Queue 11(12):20–40
11. Fernandes EL, Rojas E, Alvarez-Horcajo J, Kis ZL, Sanvito D, Bonelli N, Cascone C, Rothenberg CE (2020) The road to bofuss: the basic openflow userspace software switch. J Netw Comput Appl, 102685
12. Kabbani A, Vamanan B, Hasan J, Duchene F (2014) Flowbender: flow-level adaptive routing for improved latency and throughput in datacenter networks. In: Proceedings of the 10th ACM international on conference on emerging Networking Experiments and Technologies, pp 149–160
13. Kuźniar M, Perešíni P, Kostić D, Canini M (2018) Methodology, measurement and analysis of flow table update characteristics in hardware openflow switches. Comput Netw 136:22–36
14. McKeown N, Anderson T, Balakrishnan H, Parulkar G, Peterson L, Rexford J, Shenker S, Turner J (2008) Openflow: enabling innovation in campus networks. ACM SIGCOMM Comput Commun Rev 38(2):69–74
15. Reitblatt M, Foster N, Rexford J, Schlesinger C, Walker D (2012) Abstractions for network update. ACM SIGCOMM Computer Communication Review 42(4):323–334
16. Sanvito D, Filippini I, Capone A, Stefano P, Jérémie L (2019) Clustered robust routing for traffic engineering in software-defined networks. Elsevier Comput. Commun. 144:175–187
17. Sanvito D, Marchini A, Filippini I, Capone A (2020) Cedro: an in-switch elephant flows rescheduling scheme for data-centers. In: 2020 6th IEEE conference on network softwarization and workshops (NetSoft). IEEE
18. Sanvito D, Moro D, Gullì M, Filippini I, Capone A, Campanella A (2018) Onos intent monitor and reroute service: enabling plug&play routing logic. In: 2018 4th IEEE conference on network softwarization and workshops (NetSoft). IEEE, pp 272–276