Received: 31 January, 2020 | Revised: 19 May, 2020 | Accepted: 04 June, 2020

DOI: 10.1002/nem.2127

SPECIAL ISSUE PAPER

Towards Distributed Emergency Flow Prioritization in SDN Networks

Jerico Moeyersons*[1] | Behrooz Farkiani[2] | Tim Wauters[1] | Bruno Volckaert[1] | Filip De Turck[1]

[1]Department of Information Technology, Internet Technology and Data Science Lab (IDLab), Ghent University - imec, Gent, Oost-Vlaanderen, Belgium

[2]Computer Engineering Department, Amirkabir University of Technology, Tehran, Iran

Correspondence

*Jerico Moeyersons, iGent, Technologiepark-Zwijnaarde 126, B-9052 Gent, Belgium. Email: jerico.moeyersons@ugent.be

Abstract

Emergency services must be able to transfer data with high priority over different networks. With 5G, slicing concepts at mobile network connections are introduced, allowing operators to divide portions of their network for specific use cases. In addition, Software-Defined Networking (SDN) principles allow to assign different Quality-of-Service (QoS) levels to different network slices.

This paper proposes a microservices-based framework, able to run both centralized and distributed, that guarantees the required bandwidth for the emergency flows and maximizes the best-effort flows over the remaining bandwidth based on their priority. The proposed framework consists of an offline linear model, allowing to optimize the problem for a batch of flow requests. For dynamic situations, an online approach is also required in the framework to handle new incoming flows by calculating the path with a shortest path algorithm and utilising a greedy approach in assigning bandwidth to the intermediate flows.

In this article, the linear model is evaluated through simulation, the distributed architecture is evaluated through emulation while the online approach is validated through physical experiments with SDN switches. The results show that the linear model is able to guarantee the resource allocation for the emergency flows while optimizing the best-effort flows with a sub-second execution time. The distributed architecture is able to split up the managed network into different parts, allowing division of work between controllers. As a proof-of-concept, a prototype with Zodiac switches validates the feasibility of the centralized framework.

KEYWORDS:

Network Management, SDN, Linear Programming, Emergency Response

1 | INTRODUCTION

During an emergency event, it is required to prioritize the network traffic that is coming from and going to the emergency services in the presence of large civilian crowds in order to coordinate the relief and response. The enabler for this statement were the terror attacks at Brussels airport and the metro station in Maalbeek on March 22, 2016[1]. Right after the two explosions, the phone networks in Belgium had broken down and saturated as a lot of people were looking to contact the emergency services, friends and family. This also caused communication problems within the emergency services itself. To avoid similar cases in the near future, ASTRID, the specialist telecoms operator for Belgium's emergency and security services, launched priority SIM cards

for specific persons. This will allow these persons (such as the minister of defense, first aid commanders, etc) to have secure and priority access to the mobile network[2]. However, these priority SIM cards cannot be shared with other persons that may need it during a specific emergency situation, because every situation can be different and will require other capabilities. Therefore, a more generalized solution is required that can guarantee the bandwidth of emergency network traffic and can optimize the other, non-priority traffic, over the remaining bandwidth in the network. The solution can be found by first looking into the next generation mobile networks, called 5G.

Since Release Document 15 of the 3GPP (3rd Generation Partnership Project)[3], the 5G system is introduced and explained. In Release 16, expected to be formally released in June 2020, the completion of the 5G specifications as well as enhancements to many early capabilities for 5G standalone mode, including URLLC (Ultra-Reliable and Low Latency Communication), V2X Phase 3 and more are described. While some parts are thus already implemented and rolled out by the industry, the formal release of 5G is expected be sometime in 2021[4]. One of the future aspects of 5G systems is to cater a wide range of services differing in their requirements and types of devices, going further than the traditional human-type communications and thus includes machine-type communications. In that case, the network must be able to take different forms depending on the required service, leading to the slicing of the network on a per-service basis. Technologies such as SDN (Software-Defined Networking) and NFV (Network Function Virtualization) can be used to provide these network slicing concepts, simultaneously providing a multitude of diverse services over a common underlying physical infrastructure[5]. This will allow network operators to provide portions of their networks for specific use cases such as Iot (Internet-of-Things), streaming videos and smart energy grids.

There are three layers needed that enable network slicing in the future 5G networks, namely the infrastructure layer, the network function layer and a service layer[5], all containing the necessary tools for the operators, enterprises, etc. These three layers are managed by a management and orchestration (MANO) controller. The architecture is illustrated in Figure 1 and the infrastructure and network function layer will be further explained. The infrastructure layer refers to the physical network infrastructure including both the RAN (Radio Access Network) and the Core Network (CN) but also the deployment, control and management of the infrastructure. The allocation of resources to slices will also happen in this layer where the resources of each slice can be revealed and managed by the network function layer and eventual extra layers such as the service layer. The network function layer encapsulates the operations that are related to the configuration and life cycle management of the network functions that offer end-to-end service in the network slice. These network functions must however be placed optimally over the virtual infrastructure and chained together to work optimally. In this layer, the industry and researches have already found a consensus about the role of SDN and NFV.[6,7,8] NFV separates network functions from the underlying proprietary hardware appliances,[9] enabling the life cycle management and orchestration of the network functions. The network functions running on dedicated hardware are thus transferred to software-based applications running in datacenters, network nodes, end-user premises etc. SDN on the other hand is an important technology to implement dynamic and flexible network management by separating the data plane from the control plane in networks[10]. Every SDN switch (further called switch) within an SDN network operates as a simple packet forwarding device that is controlled by a logically centralized software program, the SDN controller. An SDN controller performs all complex functions such as routing, naming and security checks. The controller defines the data flows that occur in the network, considering that e.g. the communication is permissible by the network policy. If the controller allows a flow, it computes a route for the flow and adds an entry for that flow in each of the switches along the path. Switches are now responsible for managing their own flow tables whose entries are thus populated by the controller. The switch also performs certain functions in an SDN network. When a new packet of a flow arrives at the switch, the switch forwards and encapsulates it to the controller. The controller can thus decide to add the flow to the flow table of the switch or always drop the packets in that flow. Other packets are forwarded to the specific output port of the switch, based on the entries in the flow table. Some flow tables may include priority information set by the controller. The controller can also decide to drop specific packets, apply bandwidth meters to limit the maximum bandwidth available to certain flows, etc. The communication between the controller and the switches uses a standardized protocol API, most commonly with the OpenFlow specification.[11] In this article we will focus on the network function layer within the network slicing concept in order to provide a generalized solution for the above described problem.

Therefore, this article presents a generalized and containerized framework to guarantee the bandwidth of emergency network traffic by generating SDN high-priority flows while other, non-priority traffic, will receive best-effort resources. The proposed framework is thus a use case within the slicing concept of 5G networks. Because different operators can collaborate in one network topology or because multiple controllers can manage the SDN-based network topology, a distributed approach is necessary. This allows for better management of the topology while guaranteeing the emergency flows and optimizing the best-effort flows. Fog computing and smart cities are other use cases that can benefit from the proposed framework.[12] A simulation, emulation

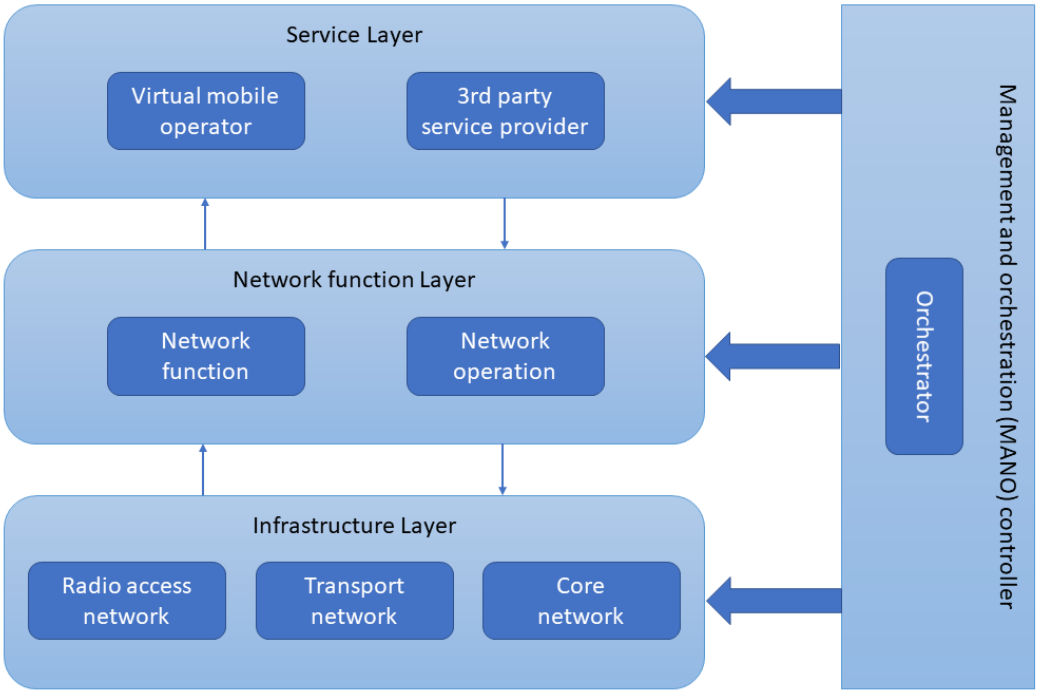

FIGURE 1 Generic framework representing various 5G architectural proposals[5].

and prototype has been implemented in order to evaluate our proposed framework in terms of speed, scalability and accuracy. In real cases, network operators deploying software-defined technologies can allow emergency flows to reserve specific slices of bandwidth for a specific amount of time.

This article contributes to four main topics: *(i)* design of models for guaranteed bandwidth allocation for emergency flows while optimizing best-effort flows over the remaining network resources, *(ii)* design of a joint online-offline approach to practically implement the model, *(iii)* design and implementation of a centralized and distributed microservices-based framework and *(iv)* the validation of the model through simulations, emulations and practical evaluation. The remainder of this paper is organized as follows: Section 2 presents related work. In Section 3, the problem description is given followed by the problem formulation as linear model. Section 4 presents the architectural design and implementation of our framework followed by the evaluation methods and the corresponding result in Section 5. Finally, Section 6 discusses conclusions and future avenues of research.

2 | RELATED WORK

The network slicing concept introduces the possibility to enable new features such as more fine-grained Quality-of-Service (QoS), and a lot of research is done on SDN over the past few years.[13,14] Different algorithms to provide QoS, but without considering bandwidth guarantees, are presented.[15,16] Yan et al.[15] proposed a QoS solution based on SDN technology. They first defined a cost function which assigns a positive value to each link based on bandwidth, length and congestion of the link. Afterwards, they utilized a weighted shortest path algorithm[17] to find multiple paths for each source and destination pair in the network. When a new flow arrives, the path with the lowest cost is selected as the routing path for the flow. Zhang et al.[16] proposed a QoS framework based on the OpenFlow protocol which dynamically calculates a path for each flow. If the flow is a QoS-required flow, an algorithm based on Dijkstra is used to find the path with minimum delay and cost values.

Akella et al.[18] presented an approach to allocate bandwidth and satisfy QoS requirements. They categorized flows into QoS and best effort flows and defined a metric, used in path selection, that considers the requested rates. Shaohua et al.[19] categorized cloud applications into three levels based on the sensitivity to delay and bandwidth. A flow-based adaptive routing algorithm

which utilizes Dijkstra and K-shortest path[20] algorithms with the aim of maximizing the utilization of network resources is proposed and evaluated through simulation by Tomovic et al.[21] Pinto et al.[22] defined four service classes including best effort and bandwidth guaranteed classes. Each new flow is first assigned to the probing class and its behavior is monitored. After some time, if the network can support its bandwidth along the path it will be reassigned to the bandwidth guaranteed class or otherwise the best effort class. A method to provide bandwidth guarantees by using OpenFlow meters and queues is presented by Krishna et al.[23] The authors categorized flows into QoS flows which have minimum guaranteed bandwidth and best effort flows with no requirements. For each QoS flow, first, an admission control process checks whether there is a path that can accommodate the flow rate. After that, by using a meter at the ingress switch, the input rate of the flow is monitored and if it exceeds the defined rate, the packets will be marked. Using three different queues at the egress port of each switch along the path for marked and unmarked QoS and best effort flows, traffic prioritization is made possible. Morin et al.[24] used MPLS tunnels to provide end-to-end bandwidth guarantees, which is similar to the work of Krishna et al.[23] where they used OpenFlow meters at the ingress switches. For each flow the input rate of the flow is monitored and based on that, a priority value is set in the header of each packet. Then, an MPLS tunnel is used to route the packets toward the egress switch and the priority of each packet specifies its output queue. Lu et al.[25] utilized preplanned network slices to both satisfy QoS requirements and maximize the overall throughput of the network. The authors used the traffic history to create network slices which have fixed configurations during the network lifetime. When a flow arrives, it is assigned to a slice by using the VLAN ID of the slice. The MaxStream framework[26] is proposed in order to maximize the number of streaming sessions and bandwidth provisioning. The authors formulated two Integer-Linear Programming (ILP) problems. The first problem maximizes the number of accepted flows by considering the requested rate of the flows. Then, the set of accepted flows is used in the second problem to maximize the total rate of the accepted flows. Since the authors focused on multimedia streams, they ignored best effort flows with no QoS requirements.

More recent work proposes auction-based resource allocation in multi-tenant networks[27] and an SDN-based architecture for providing QoS to high-performance distributed applications.[28] The auction-based resource management scheme provides an online approach by means of a non-cooperative game theory. It achieves gains of up to 5 x reduction in transmission delays, but it does not focus on cases where different types of flows are active in the network. The architecture to develop a QoS provisioning, presented by Oliveira et al.,[28] assumes that network operators implement specific QoS levels in the network topology whereby in our case, we optimize the existing best-effort flows (and the corresponding QoS levels) over the network without pre-configured QoS levels.

The most relevant studies are summarized in Table 1. In our previous article,[29] we have utilized both online and offline approaches to provide bandwidth guarantees for emergency flows and maximize the total rate of best effort flows. The offline approach optimized all existing emergency and best-effort flows while the online approach routed, based on a weighted shortest path algorithm, and allocated sub-optimally new incoming flows through a greedy heuristic in between offline batches. In this article however, we recreated the previous solution as a containerized framework, allowing us to also evaluate the distributed behaviour of our framework in different network topologies. As in the previous solution, only drop meter policies are used in this article because the current OpenFlow versions[30] do not support other policies such as 2-color-marking[31] and 3-color-marking.[32].

3 | PROBLEM DESCRIPTION AND FORMULATION

In this section, the problem described in Section 1 is analyzed in detail. Afterwards, a linear model to solve this problem is presented, aiming to guarantee emergency flows while the best-effort flows are optimized over the remaining bandwidth in the network. This approach is designed for topologies where all the network flows are gathered based on prior knowledge or predictions and no new flows will be created. In a more realistic case, where new flows arrive dynamically, a second approach, further called the online approach, is described. It combines the solutions from the linear model, further called the offline approach, with a sub-optimal solution to handle new incoming flows.

3.1 | Problem description

Within an SDN network, OpenFlow-enabled switches are connected to one or more SDN controllers and different best-effort flows in each of the switch flow tables are responsible for the correct routing of the network traffic. A flow is described using a tuple `<source, destination, class>` whereby the class describes the traffic class of the flow based on a priority value and the corresponding lower and upper bound bandwidth rates. In an emergency situation where e.g. a video feed must be transferred

TABLE 1 Summary of related research

Reference	Offline/Online	Objective	Path Selection	Evaluation
Akella et al.[18]	Online	Satisfy the QoS requirements of the QoS flows	Greedy	Geni Testbed[33]
Pinto et al.[22]	Online	Admission control and traffic management	Sink tree	Mininet[34]
Krishna et al.[23]	Online	Satisfy the minimum bandwidth requirements of QoS flows	Widest shortest path	Open vSwitch[35] and physical switches
Morin et al.[24]	Online	Guarantee bandwidth of QoS flows	SAMCRA[36] and Dijkstra	Mininet and physical switches
Samani et al.[26]	Online	Maximize the number of streaming sessions and bandwidth provisioning	Two ILP problems	Mininet
Previous Paper[29]	Online & Offline	Maximize the total rate of the best effort flows and guarantee bandwidth of high priority flows	Dijkstra (online), ILP problem (offline)	Mininet & Physical switches
This Paper	Online & Offline	Maximize the total rate of the best effort flows and guarantee bandwidth of high priority flows in a distributed manner	Dijkstra (online), ILP problem (offline)	Mininet

over the network, emergency flows will be requested for prioritizing this traffic. These emergency flows need to be satisfied by guaranteeing the requested bandwidth while the remaining bandwidth of the network should be allocated to the other best-effort flows. The priority of the traffic classes will be used to optimize the best-effort flows where a higher priority requires a larger share of the available network bandwidth.

This article proposes a solution to maximize the total input rate of the best-effort flows in the network while the requested rates of the emergency flows are satisfied and the bandwidth capacity constraints of the network are respected. The assumption is made that the requested rate for the emergency flows is not higher than the total available rate in the network. A linear model, LP, is defined aiming to solve this problem. This LP formulation uses the principles of flow splitting, allowing flows to be separated over different links which optimize the bandwidth resource allocation.[37] The packet reordering effect that can occur when using flow splitting, can be mitigated using hash-based splitting and packet tagging.[38] However, flow splitting is not supported by every OpenFlow-enabled switch, and a more generalized linear model is needed. Therefore, a second formulation, ILP, is defined where flows cannot be split up and each flow needs to be assigned to a single path from source to destination. Both offline models are evaluated afterwards.

The formulations of these two offline models are provided in Section 3.2 and Section 3.3. Notations used in the formulations are summarized in Table 2. Some described constraints contain a multiplication of a continuous and a binary variable and because this cannot be directly solved by state-of-the-art solvers, they need to be linearized first. These formulations will optimize the best-effort flows over the remaining bandwidth that is not used by emergency flows. In case the offline models are not able to run in real-time, the online approach manages new incoming flows in between offline batches, providing the shortest (but possible sub-optimal) path with a greedy-based solution to allocate bandwidth to these new flows.

3.2 | The ILP formulation

The ILP formulation will maximize the sum of the traffic rates of the best-effort flows, multiplied by their assigned weights over the remaining bandwidth after allocating the emergency flows. This is illustrated in (1), subjected to the constraints [(2) - (5)]

TABLE 2 Notations summary

Variables	
$y^i_{u,v}$	Equals 1 if the traffic for flow i passes through link (u, v)
R^i	The rate assigned to best effort flow i
Parameters	
$F \equiv M \cup B$	Set of all flows
M	Set of all emergency flows
B	Set of all best effort flows
$G = (V, E)$	The graph of the network. V is the set of nodes and E is the set of physical links. All links are bidirectional with different capacity in each direction
$Z^i_{u,v}$	The rate of flow i on link (u, v)
$Cap^{(u,v)}$	The bandwidth of the link between u and v, in the direction from u to v
W_i	The weight assigned to flow i based on the traffic class it belongs to
τ_i	The requested rate for emergency flow i
$minRate^i$ $maxRate^i$	The lower bound and upper bound for the rate of flow i based on the traffic class it belongs to.
$Source\,(i)$ $Destination\,(i)$	The source and the destination of flow i

and explained further below.

$$\max \sum_{i \in B | \{u = Source(i), (u,v) \in E\}} W_i \times Z^i_{u,v} \tag{1}$$

Subject to:

$$\sum_{(u,v)\in E} y^i_{u,v} - \sum_{(v,u)\in E} y^i_{v,u} = \begin{cases} 1 & u = Source\,(i) \\ 0 & \text{otherwise} \\ -1 & u = Destination\,(i) \end{cases} \quad \forall u \in V,\ i \in F \tag{2}$$

$$\sum_{i \in B} y^i_{u,v} \times R^i + \sum_{i \in M} y^i_{u,v} \times \tau_i \leqslant Cap^{(u,v)} \tag{3}$$

$$\sum_{(u,v)\in E} y^i_{u,v} \leqslant 1 \quad \forall u \in V,\ i \in F \tag{4}$$

$$\sum_{(v,u)\in E} y^i_{v,u} \leqslant 1 \quad \forall u \in V,\ i \in F \tag{5}$$

$$R^i \in \left[minRate^i,\ maxRate^i\right] \tag{6}$$

$$y^i_{u,v} \in \{0, 1\} \tag{7}$$

$$Cap\,(u, v) \geqslant \tau_i \quad \forall i \in M \tag{8}$$

(2) is the flow conservation constraint, guaranteeing a path from source to destination. (3) enforces the capacity limit of each physical link and (4) and (5) are used to prevent loops as much as possible. (6) and (7) specify the bounds for the assigned rate and whether traffic is passing through link `(u, v)`. Finally in (8), the assumption is made that the network is at least able to handle all requested emergency flows.

The second constraint contains a multiplication of a continuous and a binary variable as in $\sum_{i \in B} y^i_{u,v} \times R^i$. The constraint can be linearized as follows:

$$Z^i_{u,v} \leqslant Cap^{(u,v)} \times y^i_{u,v} \tag{9}$$

$$Z^i_{u,v} \leqslant R^i \tag{10}$$

$$R^i + Cap\,(u,v) \times y^i_{u,v} - Z^i_{u,v} \leqslant Cap^{(u,v)} \tag{11}$$

$$Z^i_{u,v} \in \left[0,\ \ maxRate^i\right] \tag{12}$$

3.3 | The LP formulation

The LP formulation will use the principles of flow splitting to solve the described problem. The main objective is again to maximize the sum of the traffic rates of the best-effort flows multiplied by their assigned weights over the remaining bandwidth after allocating the emergency flows. This is illustrated in (13), subjected to constraints [(14) - (16)] and explained further below.

$$\max \sum_{i \in B} W_i \times R^i \tag{13}$$

Subject to:

$$\sum_{(u,v) \in E} y^i_{u,v} - \sum_{(v,u) \in E} y^i_{v,u} = \begin{cases} -R^i & u = Source\,(i) \\ 0 & \text{otherwise} \\ -R^i & u = Destination\,(i) \end{cases} \quad \forall u \in V,\ i \in B \tag{14}$$

$$\sum_{(u,v) \in E} y^i_{u,v} - \sum_{(v,u) \in E} y^i_{v,u} = \begin{cases} \tau & u = Source\,(i) \\ 0 & \text{otherwise} \\ -\tau_i & u = Destination\,(i) \end{cases} \quad \forall u \in V,\ i \in M \tag{15}$$

$$\sum_{i \in F} y^i_{u,v} \leqslant Cap^{(u,v)} \tag{16}$$

$$R^i \in \left[minRate^i,\ maxRate^i\right] \tag{17}$$

$$y^i_{u,v} \in \mathbb{R}_{\geqslant 0} \tag{18}$$

$$Cap\,(u,v) \geqslant \tau_i \ \ \forall i \in M \tag{19}$$

(14) and (15) are the flow conservation constraints for the best effort and emergency flows respectively. (16) enforces the bandwidth capacity limits of physical links. The LP formulation is solvable in polynomial time.[39,40]

3.4 | Online approach

In a realistic scenario, the LP or ILP model runs in batches in order to continuously optimize the network topology. When a new flow arrives in between this batch of the linear model, it should be handled appropriately in order to avoid long delays in assigning this new flow. Therefore, the online approach handles new incoming flows through a sub-optimal solution in between the offline batches. The shortest path between source and destination is determined by using a weighted shortest path algorithm based on Dijkstra's algorithm.[17] A new incoming emergency flow will obtain its requested bandwidth while newly arriving best-effort flows will be temporary assigned to the average best-effort traffic class. In case there is no bandwidth available for the new flow, a greedy heuristic determines which other best-effort flows should be decreased in bandwidth until the offline batches optimizes this again. This is illustrated in Algorithm 1 and the complete online approach is described in Algorithm 2. Note that the solution from the online approach is only temporary because it will be replaced with the optimal results from the linear model.

Algorithm 1 Greedy heuristic

```
τ ← requested bandwidth
C ← traffic classes sorted by priority (low to high)
for i in count(C) − 1 do
    a ← τ/2
    τ ← τ/2
    band[i] ← a
end for
band[count(C) − 1] ← τ
for each traffic_class in C do
    i ← index
    s ← number of meters in traffic_class
    for each meter in traffic_class do
        meter ← meter − band[i]/s
    end for
end for
```

Algorithm 2 Online approach

```
R ← average best-effort traffic rate
B ← best-effort flows
while batch is running do
    X ← new incoming flow
    if X is emergency then
        τ ← requested bandwidth by X
    else
        τ ← R
    end if
    if τ is not available then
        apply_greedy_heuristic()
    end if
    apply_flows()
end while
run_batch()
```

4 | FRAMEWORK DESIGN

In this section, the micro-service based framework design is explained. First, the different architectural components are identified. Afterwards the different components are prototyped in order to create the proposed framework.

4.1 | Architectural components and tasks

To be able to identify the microservices-based architecture of our proposed framework, different components need to be identified based on the problem described in Section 1 and Section 3.1. The main goal is to manage SDN network topologies, so an SDN controller is the first component of our architecture. Next, the offline and online problem should be able to manage existing and new incoming flows, and will be called the solver. The offline problem requires network topology information in order to calculate the optimal solution for our problem, a data store is thus needed. Finally, a REST API is required in order to communicate with and manage the different components. An overview of the different components is depicted in Figure 3.

Every component is responsible for a specific set of tasks. The controller component is responsible for the management of the corresponding network topology. It will handle new incoming flows by forwarding them to the solver in order to receive

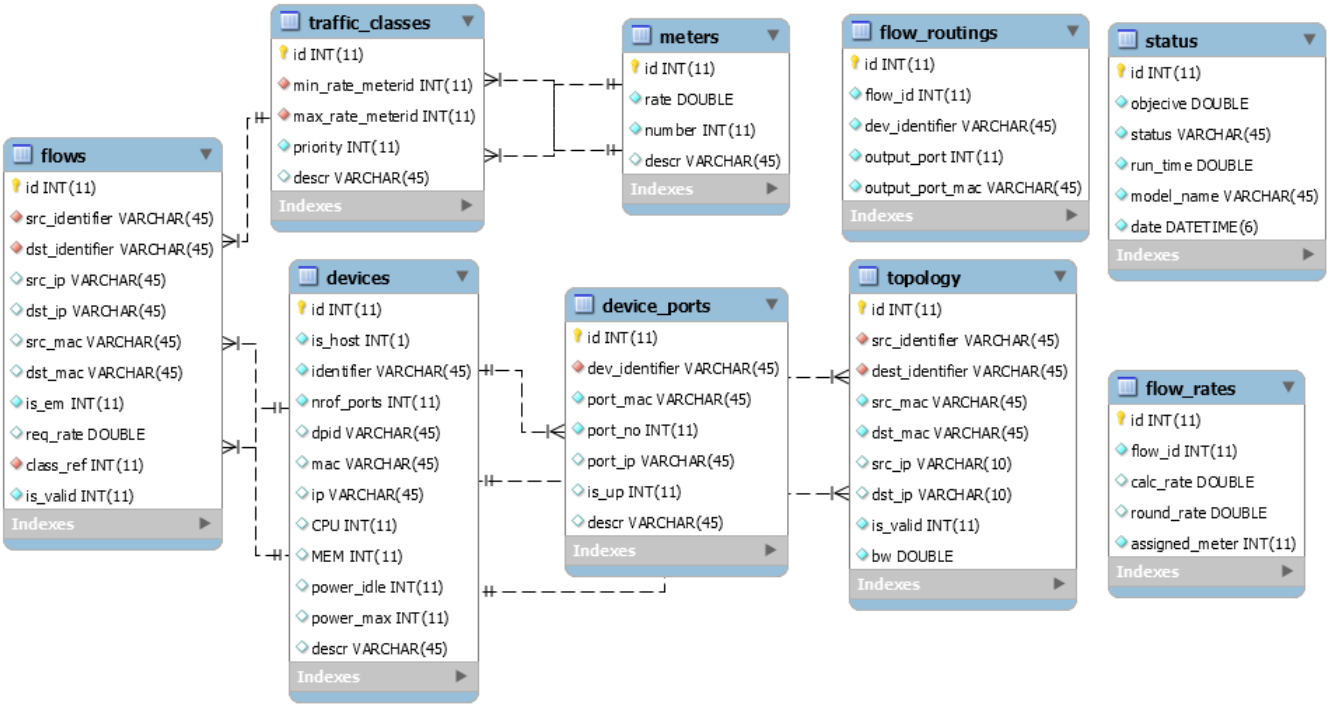

FIGURE 2 Topology of the MySQL database. The tables devices, device_ports, meters, topology and traffic_classes are filled in based on the network topology. The table flows contains the required flows in the topology and flow_rates and flow_routings contain the optimized best-effort and emergency flows after solving the offline problem.

a correct path and traffic class. The controller will also update the database with the discovered topology and when the solver notifies the controller that a new solution is available, the controller pulls this new information from the database. The main task of the solver is thus to optimize the current flows in the current operational network topology, both with the online and offline method described in Section 3. Finally, the REST API component creates an API in order to manage and retrieve information from the solver and the controller components. It is also responsible for managing multiple instantiations of this architecture, as will be explained in the next section.

4.2 | Framework prototype

Now that the different architectural components are identified, each component has been prototyped. The different components are containerized using Docker CE Version 19.03.[41] The controller component is instantiated with a custom Ryu SDN Controller[42] and an API implemented with the Python Flask framework[43] in order to communicate with the other components. The solver component is split up into two containers, the online problem is responsible for the communication with the controller, calculating the shortest path from source to destination for new incoming flows and allocating bandwidth based on the greedy heuristic, both explained in Section 3.4, and finally it runs the offline model in batches. The offline model pulls the topology information from the database and stores the results. The offline model is based on openJDK[44] version 8 update 181 and IBM ILOG CPLEX[45] v12.7. The database component is instantiated as a MySQL version 8.0.18 database[46] and the table design is illustrated in Figure 2 and further explained.

The *flows* table contains the different flows in the network topology. Each flow is connected to a specific traffic class in the *traffic_classes* table and a traffic class is connected to two bandwidth meters, one for specifying the minimum rate and one for the maximum rate. Each flow is also twice connected to the *devices* table, one for specifying the source and one for the destination. This devices table also contains the different switches in the network topology. Each device has one or more ports, enlisted in the *device_ports* table and the links between the different devices, forming the actual network topology, is stored in the *topology* table. The *flow_routings* table contains the different hops per flow and the *flow_rates* table contains the assigned

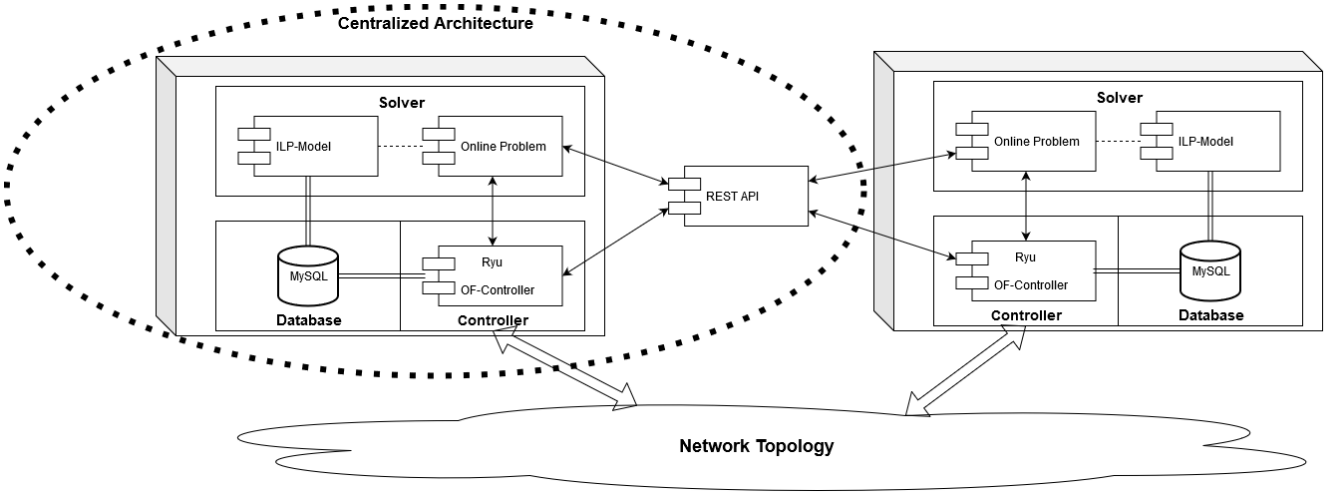

FIGURE 3 The architectural components and the instantiation of our proposed framework. The dotted circle illustrates a centralized architecture while the whole figure illustrates a distributed approach where two network operators collaborate, both having their own solvers, controllers and data stores.

meters per flow. These two tables are filled in by the offline model. The *status* table contains the practical information about the different calculations performed during the processing of the offline model.

The proposed framework can be used both centrally and distributed. In case of a centralized architecture, illustrated in Figure 3 within the dotted circle, only one controller is responsible for the whole network and the network topology is thus added completely to the MySQL database. This approach is similar to the approach used in our previous paper[29]. A distributed approach enables the network topology to be managed by different controllers and is also a more realistic case, as for example different operators are responsible for the entire network. This is shown in Figure 3 and further explained. The two controllers will add a special virtual switch called `xx` to connect it with the border switches[1] of their part of the network. When the online or offline problem calculates flow routes going over multiple network providers, the controllers detect that the `xx` switch is part of the route, and ask the other controller(s) to check which of their border switches are connected with the `xx` switches in their part. Once the next part is found, the controller constructs a path from the border switch under its' control to the border switch in the next part, allowing the network traffic to be sent over different parts in the topology. The connections between the border switches in the different parts of the network topology are based on prior knowledge.

This distributed architecture allows to calculate optimal paths for best-effort flows after guaranteeing the bandwidth for emergency flows. In case when the network topology is great, or the hardware resources to run the framework are limited, the framework offers a division of the topology in several smaller parts in order to calculate the necessary results.

5 | IMPLEMENTATION, SIMULATION AND EVALUATION

In this section, the proposed framework is evaluated in three ways. First, the offline models described in Section 3.2 and Section 3.3 are evaluated by simulation. Next, the framework is implemented and deployed on different systems in order to evaluate the distributed behaviour on an emulated network topology. Finally, the framework is evaluated on a smaller scale together with a practical environment.

5.1 | Simulation Environment

The proposed offline models are first validated using simulations. The evaluated topology, as shown in Figure 4, consists of 16 ingress/egress points of traffic and 32 switches. Switches 16-30 are backbone switches and the backbone network has the same topology as the Internet2 network.[47] This topology is used to simulate a provider network catering to about 2050 flows. Switches

[1] A border switch is a switch in the current part of the topology that is connected to a switch in another part of the topology.

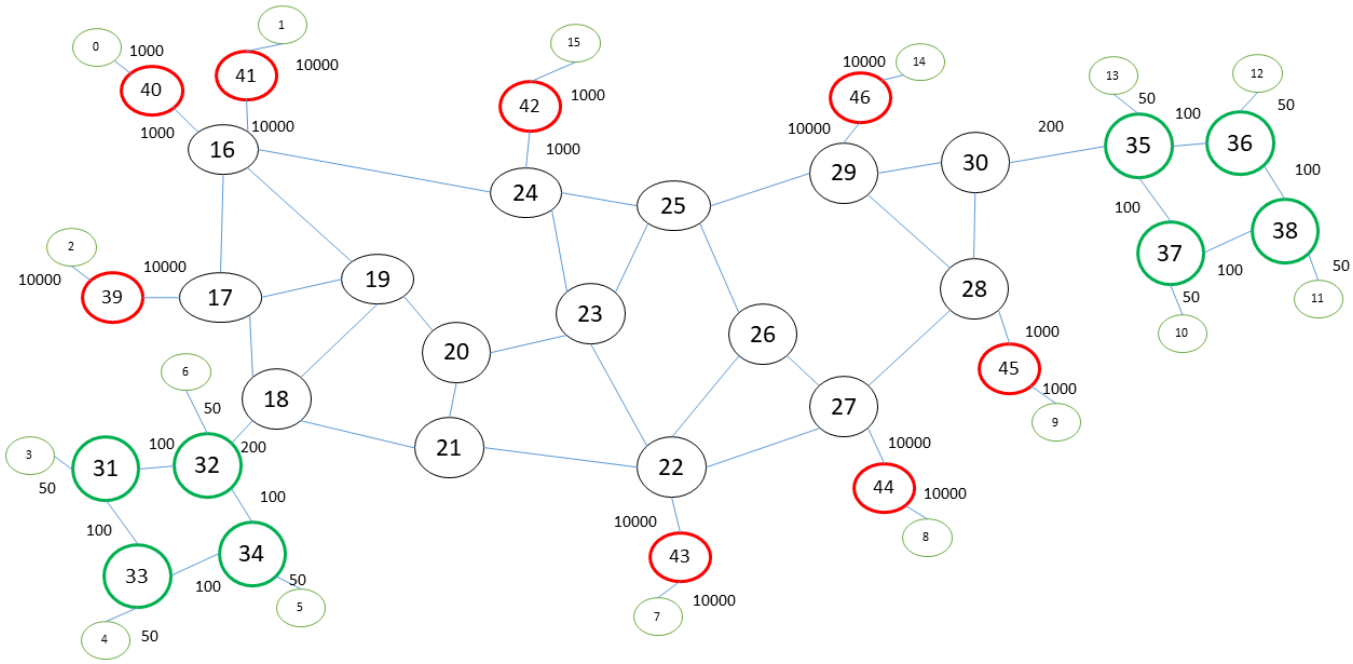

FIGURE 4 The simulation topology based on the Internet2 network.

TABLE 3 Specification of the network scenario

Source/Destination of All Flows	0 - 15
Backbone Network (40 Gbps)	16 - 30
Source/Destination Emergency Flows	{0, 2, 3, 5, 7, 8, 11, 13, 14}
DSL Network	39 - 46
Mobile Network	31 - 38

31-38 are mobile base stations and the ingress/egress points attached to them represent mobile users. Switches 39-46 are DSL switches and the ingress/egress points attached to them represent DSL users. The bandwidth of each the backbone network link is 40 Gbps bidirectional. The specifications of the network scenario are summarized in Table 3. IBM ILOG CPLEX v12.7 is used to implement the models and the simulations are executed on a server with 2 Xeon E5-2690 v4 CPUs operating at 2.6GHz with 16GB of memory.

We defined 3 traffic classes with ranges [0, 25000], [0, 10000] and [0, 5000] Kbps with priorities of 100, 50 and 10, respectively for best effort flows. Moreover, the requested rate of each emergency flow was randomly chosen from set {25000, 10000, 5000} Kbps because of the variation in types of emergency network traffic. Each best effort flow was randomly assigned to a class. Each evaluation result is the average of 30 simulation runs.

5.2 | Simulation Evaluation - Results

The performance of the two models is compared in Figure 5. By increasing the number of best effort flows, the solving time increases in both models. However, the increase rate of the ILP model is exponentially higher than for the LP model. For 2000 best effort flows along with 50 emergency flows, the ILP model solves the problem in almost two minutes. It is worthwhile to mention that the solving time of the ILP model can further be decreased by up to one order of magnitude when using acceleration methods such as the novel algorithm based on the Benders decomposition method as described in Behrooz et al.[48]

TABLE 4 Comparison of the LP and ILP model simulation results

	LP Model	ILP Model
Solving Time-Before(ms)	484	18408
Solving Time-After(ms)	484	15210

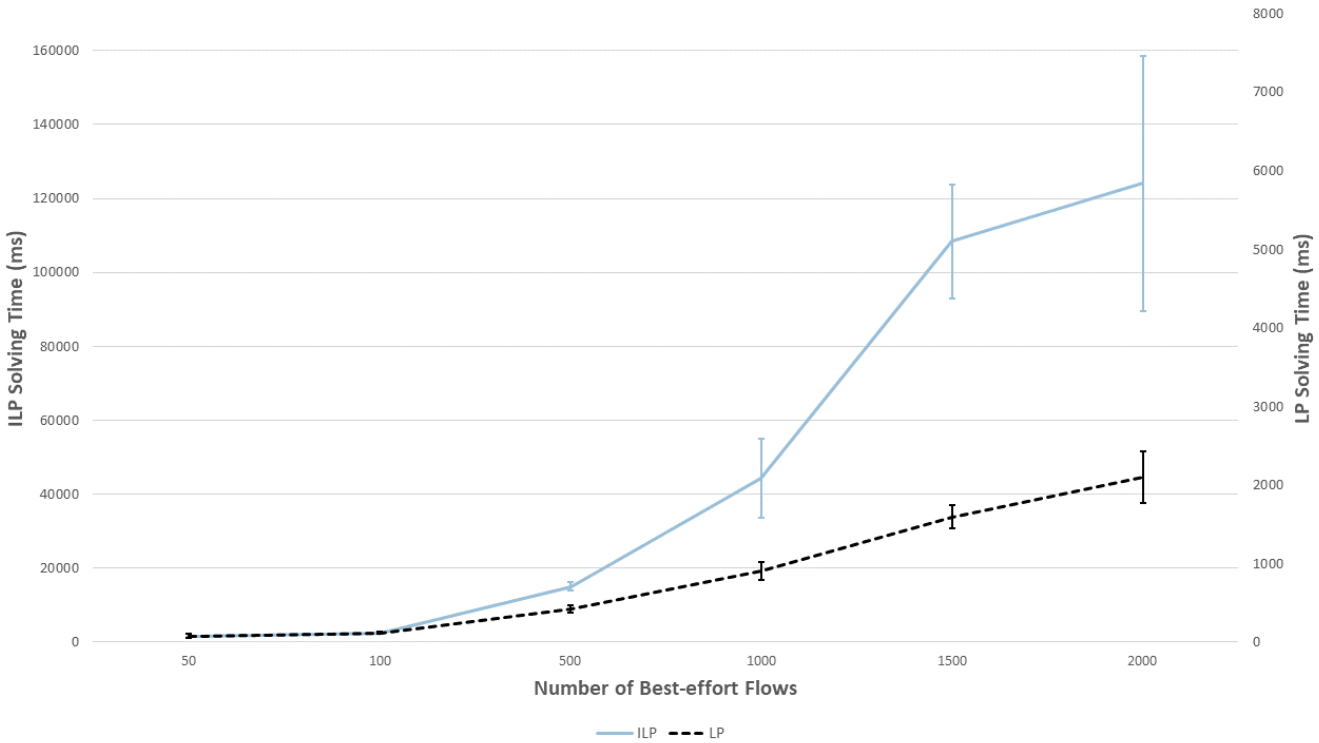

FIGURE 5 The solving time of the ILP and LP models. Standard deviations are shown in the form of error bars

To investigate the operational details of the models, we first generated 500 best effort flows and solved both the ILP and LP models. After that, we added 50 emergency flows and solved the problems again. Both models reported the same optimal values before and after adding the emergency flows which means that the same result is achieved by both models and the LP model solved the problem 30 times faster than the ILP model. After adding the emergency flows, the models decreased the rate of best effort flows to allocate the requested bandwidth of the emergency flows which resulted in a lower optimal value. A summary of the results is shown in Table 4.

5.3 | Prototype Implementation

To implement the proposed framework, both an emulated network topology (as illustrated in Figure 4) and a practical network topology consisting of Zodiac SDN switches,[49,50] depicted in Figure 6 are used. The emulated network topology, used to evaluate the distributed approach, is implemented using Mininet version 2.3.0d6[51] and contains 420 flows. The default installation of Mininet only supports OpenFlow version 1.0, but because our framework will use OpenFlow meters to implement the different traffic classes, OpenFlow version 1.3 or higher is required. In order to let Mininet use OF 1.3, the CPqD switch[52] must be installed together with Mininet.[2] The distributed approach is evaluated using 2 systems, one with 24GB RAM and 2 CPUs, each with 6 cores and hyper-threading enabled running at 2.4GHz[3] and one with only 2GB RAM and 1 CPU and 1 core (without

[2]To install mininet with the CPqD switch, use the following command: `mininet/util/install.sh -n3f`
[3]The command `nproc -all` outputs 24 (2 CPUs x 6 cores x 2 threads per core). We will use 24 processing units in the remainder of this paper to refer to this server.

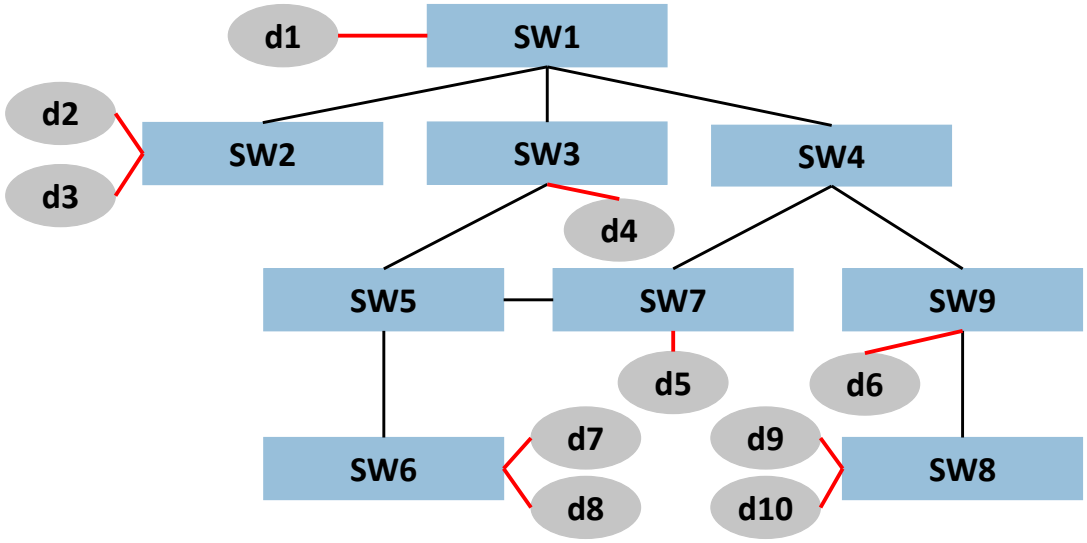

FIGURE 6 Topology of the evaluation environment. Sw1 is a Zodiac GX switch, sw2 - sw9 are Zodiac FX switches and d1 - d10 are Raspberry Pi's 3.

TABLE 5 Traffic classes (all in kbps)

Id	Name	Minimum Rate	Maximum Rate
1	High Priority	0	25000
2	Normal Priority	0	10000
3	Low Priority	0	5000

TABLE 6 Requested rates per flow based on the destination. Rates between 0 and 4999 kbps are part of traffic class 3, rates between 5000 and 9999 kbps are part of traffic class 2 and rates higher dan 10000 kbps are part of traffic class 1.

Destination	Traffic class	Destination	Traffic class	Destination	Traffic class
d1	3	d2	3	d3	3
d4	3	d5	3	d6	2
d7	2	d8	2	d9	2
d10	1				

hyper-threading) running at 1.12GHz. The practical network topology is built with 1 Zodiac GX switch[50] (sw1), 8 Zodiac FX switches[49] (sw2 - sw9) and 10 Raspberry Pi's model 3B (d1 - d10). The Zodiac GX has an uplink of 1 gbps while the Zodiac FX switches have an uplink of 100 mbps. The used traffic classes are illustrated in Table 5 and the requested bandwidth rates based on destination are summarized in Table 6. OpenFlow v1.3 meters were used to specify the upper bound and lower bound rates of each traffic class. Note that with the provided meters in this prototype, the offline model will rather assign the meter with 0 kbps bandwidth to flows with a lower priority in case there is a shortage. When more meters per traffic class are allocated, a downgrade is possible, but this is currently not implemented. The used system to run the framework has 16GB RAM and 4 cores running at 2.8GHz. Because the same framework is used for both the emulated and the practical topology and because the Zodiac switches do not support flow splitting, the offline model is implemented with the slower ILP model.

Assume $\{ m_1, m_2, \ldots, m_n \}$ are n defined meter rates and $m_i \leqslant m_{i+1}\ \forall i$. The weight of best effort flow i is calculated by $\left\lceil P_i \times \frac{m_n}{m_1+1} \right\rceil$ in which P_i is the priority of the class that the flow belongs to and $\lceil x \rceil$ is the ceiling function.

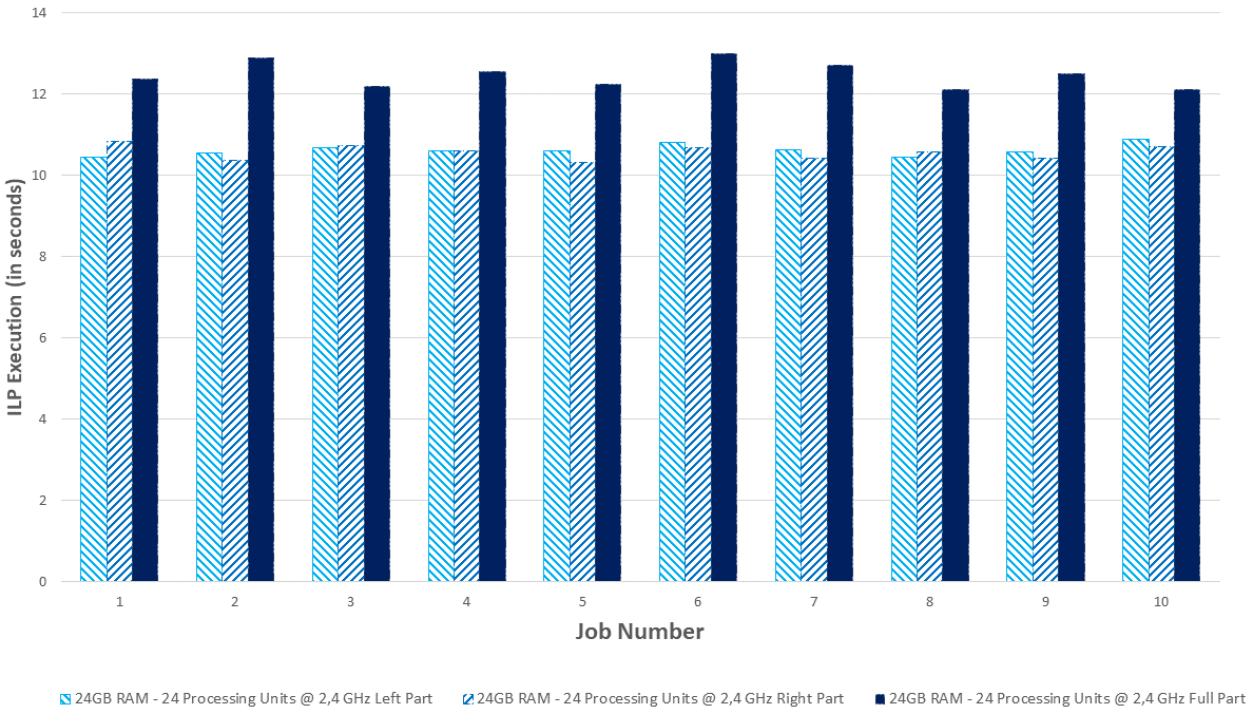

FIGURE 7 The ILP execution time for the left part, right part and full part on a server with 24GB RAM and 24 processing units, each running at 2.4GHz.

When a new flow arrives, it is added to the database by the controller. When the previous batch of the offline model is finished, the online model runs it again after a certain specified amount of time or when the previous calculation is done. The ILP model reads the database information, solves the problem and stores the results in the database. The output of the ILP is the assignment of each flow to one meter and the routing of flows over the network. To assign a flow to a meter, whether it be best effort flows or emergency flows, the implementation rounds down the calculated rate to the nearest defined meter rate. Based on the simulation results summarized in Section 5.2, the ILP model provides optimal results with a high number of flows but not in real-time. To combat this, we run the offline model consecutively while the online approach is used to route and to apply the corresponding meter to new incoming flows. Best-effort flows will be assigned to the average meter with 10000 kbps while emergency traffic will be assigned to their requested rate. To decrease the impact on the current best-effort and emergency flows, a greedy heuristic is applied to reassign available bandwidth from other best-effort flows, based on their priority.

5.4 | Distributed architecture - evaluation

The distributed architecture with the emulated network topology is first evaluated on the system with a lot of resources (24GB RAM and 24 processing units in total). The network topology is split up into two parts, further called the left part and the right part. The left part contains switches *16-21, 23, 24, 31-34, 39-42* and the right part contains switches *22, 25-30, 35-38, 43-46*. The ILP model is executed 10 times for both parts and the results are visualized in Figure 7. The ILP execution time for the whole network topology (full part) is also added in this figure. It is clear that the division of the network results in a speed-up of about 15%. However, the flow routing results of the two smaller parts differ from the full part but the objective from the ILP model is the same. This means that there are different routes in both cases, and with the distributed architecture it is possible that a flow is not routed along its shortest path, but this is without any noticeable delay. Because the objective from the ILP model is the same, the same optimization is achieved in both cases, meaning that the flows received the same traffic classes.

Next, the distributed architecture is evaluated on a system with fewer resources (2GB RAM and 1 processing unit), whereby the network topology is again split up into the same two parts. The results of this evaluation are illustrated in Figure 8. In comparison with the other server, the results of the full part are not included in this figure, because the calculation was not

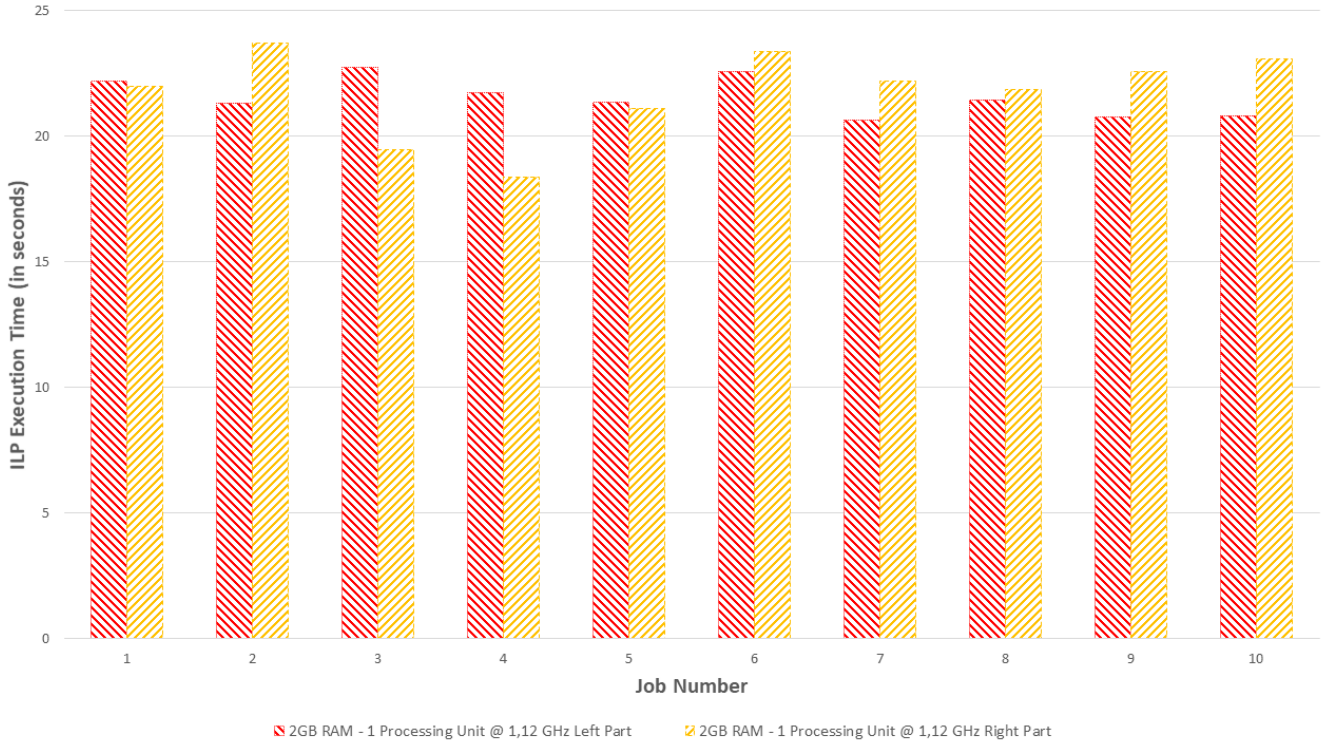

FIGURE 8 The ILP execution time for the left part and right part on a server with 2GB RAM and 1 processing unit, running at 1.12GHz. Note that the ILP execution for the full part had not enough memory and is thus not visualized.

possible due to the lack of memory. This already illustrates the importance of the distributed architecture, because most SDN controllers have limited resources in practice. Both parts achieve the same objective as in the evaluation on the other server, but it takes about 51% more time to come to a solution.

5.5 | Evaluation of the practical network topology - Example scenario

The framework with the centralized architecture is also evaluated on a practical network topology to study the behavior of the online approach and the findings are illustrated in Figure 9. When the batch calculation is running, the online approach will handle the new incoming flows. The flow responsible for the traffic going from d1 to d5, which is part of traffic class 3, is already allocated together with 87 other flows. Next at time t1, a new incoming emergency flow going from d1 to d10 is added to the network, with a requested bandwidth of 25,000 kbps. Because of its priority, the requested bandwidth is allocated and the greedy heuristic reduced the bandwidth from the other best-effort flows. The flows part of traffic class 3 have an average decrease of 284 kbps. Afterwards at time t2, a flow going from d8 to d10 is added, which is part of traffic class 1. As this is a best-effort flow, the average best-effort meter with a bandwidth of 10,000 kbps is allocated. The greedy heuristic again determines the bandwidth for each best-effort flow without impacting the current emergency flows. Finally, the batch calculations (visualized by the gray vertical line at time t3 in Figure 9) optimizes the flows of the whole network again.

It is clear that the online approach is guaranteeing the bandwidth of the emergency flows and creates a sub-optimal solution for the new incoming best-effort flows. The sub-optimal solution has 2.76% difference per flow compared to the result of the offline batches in the whole example scenario. In some cases, this difference is 100% because the online approach does not drop any new incoming flows, while the offline batches can decide to drop a flow much faster as explained in Section 5.3. Afterwards, the batch calculations optimize the best-effort flows over the remaining available bandwidth not used by emergency flows. The solving time of the batch calculations before and after adding 50 emergency flows is illustrated in Table 7 and shows that the proposed offline model can solve small-sized networks efficiently.

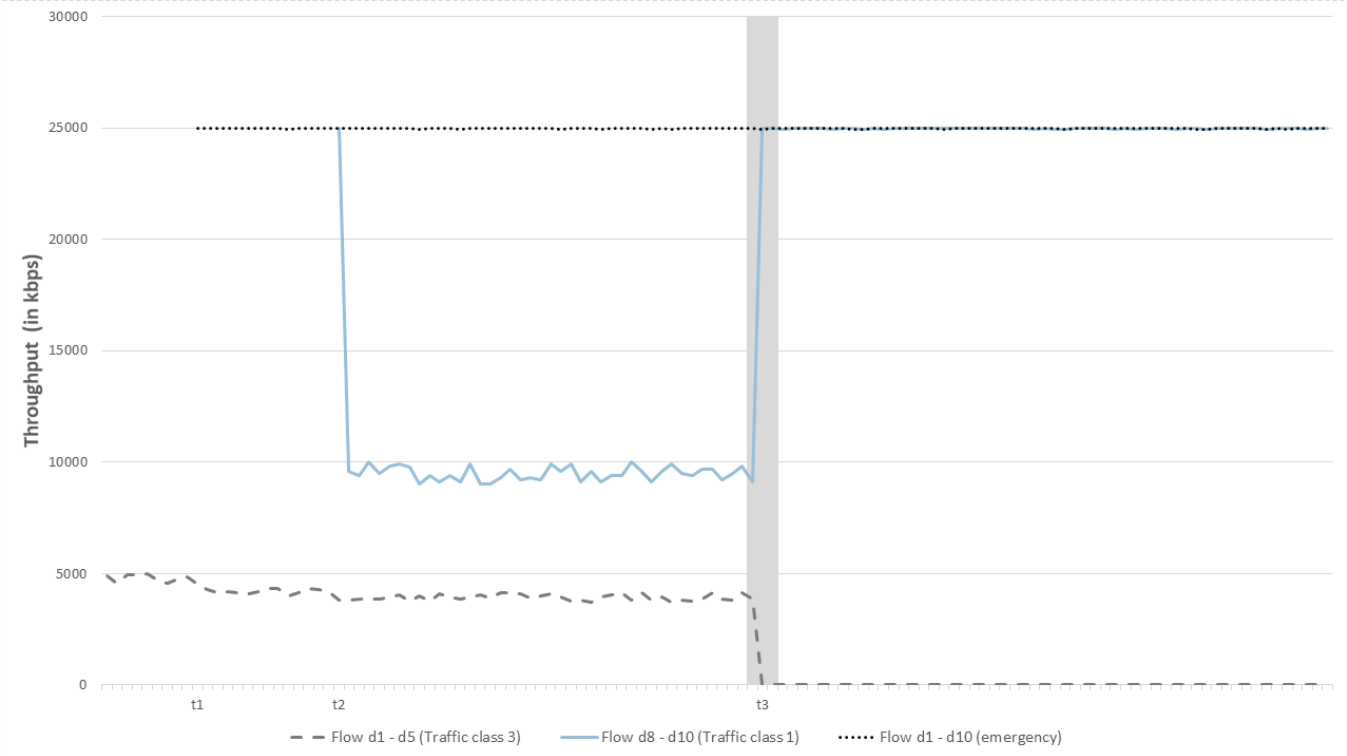

FIGURE 9 Throughput of 2 best-effort flows and 1 one emergency flow. At time t1, the emergency flow is added and assigned by the online approach. At time t2, another best-effort flow is added and assigned by the online approach. At time t3, the offline batch has calculated and applied the optimal solution.

TABLE 7 The ILP model results

	Before	After
Solving Time (ms)	20520.234	17489.531

6 | CONCLUSION

Emergency network traffic needs to have priority over best-effort traffic during emergency situations. With the expected release of 5G, slicing concepts at network level will enable prioritization of the emergency network traffic over mobile connections. In addition, SDN principles allow to assign different QoS levels to different network slices.

In this paper, we therefore first propose two mathematical linear models that guarantee the requested rate of emergency flows and maximize the best-effort flows over the remaining available bandwidth. The LP model uses the principles of flow splitting, which is not supported by every OpenFlow-enabled switch. Therefore, a second linear model, ILP, is proposed that is supported by most of the OpenFlow-enabled switches running version 1.3 or higher. Afterwards, an online approach is explained, handling new incoming flows in between batches of the linear model. The shortest path, based on Dijkstra's algorithm, is calculated and a greedy heuristic is applied to obtain bandwidth from the best-effort flows. When the new incoming flow is an emergency flow, the requested bandwidth will be allocated by any means, a new incoming best effort flow will be allocated with the average bandwidth of all the active best-effort flows. Finally, a microservices-based framework is discussed and prototyped. This framework is able to run both in a centralized and a distributed manner, enabling scalability over larger network topologies. The distributed approach is necessary as different network operators can collaborate in managing cross-operator flows in the network topology or when the hardware resources are limited.

The two offline models are first evaluated by simulations and the results show that both the ILP and LP mathematical problems can be used with the ILP model exhibiting plus-second execution time while the LP model works 30 times faster for 500 best-effort flows and 50 emergency flows. Next, the distributed approach is evaluated by using an emulated network. Results show that the distributed architecture is a solution in case there is a lack of resources, allowing to split up the network topology in multiple parts in order to calculate and optimize the emergency and best-effort flows. When enough resources are available, a split up of the network in two parts results in speed up around 15%. When the results of the distributed architecture are compared with the centralized architecture, it shows that different paths are chosen for some flows, but the allocation of resources remain the same. Afterwards, the centralized framework is evaluated on an SDN network consisting of Zodiac Switches and Raspberry pi's. The Zodiac switches do not support flow splitting, so the use of the slower ILP model is obliged. The practical evaluation shows that the online problem efficiently handles new incoming flows while guaranteeing the bandwidth for all the emergency flows and providing a sub-optimal temporary solution for the best-effort flows.

Research concerning the distributed approach when three or more controllers are connected is envisaged as future work. This is because the overhead of adding a virtual switch that will connect the border switches of a specific part can be greater, possibly resulting in longer calculation times for the flow allocation. An improved network topology discovery service that is able to optimally divide the network into different parts along with better handling of inter-part flows can offer a solution.

References

1. Griffin A. Brussels attacks: Phone networks down and saturated after explosions at Zaventem airport and Metro station. *Independent* 2016. https://www.independent.co.uk/life-style/gadgets-and-tech/news/brussels-attacks-phone-networks-zaventem-airport-explosion-maelbeek-metro-live-updates-a6945571.html.

2. Belgium: ASTRID launches the next generation of its Blue Light Mobile service - Critical Communications Today. 2017. http://www.criticalcomms.com/news/belgium-astrid-launches-the-next-generation-of-its-blue-light-mobile-service.

3. Release 15 - 3GPP. https://www.3gpp.org/release-15.

4. O'Donnell B. The Evolution of 5G. *Forbes* 2019. https://www.forbes.com/sites/bobodonnell/2019/11/12/the-evolution-of-5g.

5. Foukas X, Patounas G, Elmokashfi A, Marina MK. Network Slicing in 5G: Survey and Challenges. *IEEE Communications Magazine* 2017; 55(5): 94–100. https://doi.org/10.1109/MCOM.2017.1600951doi: 10.1109/MCOM.2017.1600951

6. Rost P, Banchs A, Berberana I, et al. Mobile network architecture evolution toward 5G. *IEEE Communications Magazine* 2016; 54(5): 84–91.

7. Zhou X, Li R, Chen T, Zhang H. Network slicing as a service: enabling enterprises' own software-defined cellular networks. *IEEE Communications Magazine* 2016; 54(7): 146–153.

8. Banchs A, Breitbach M, Costa X, et al. A Novel Radio Multiservice adaptive network Architecture for 5G networks. *2015 IEEE 81st Vehicular Technology Conference (VTC Spring)* 2015: 1–5.

9. Hawilo H, Shami A, Mirahmadi M, Asal R. NFV: State of the art, challenges, and implementation in next generation mobile networks (vEPC). *IEEE Network* 2014; 28(6): 18–26. doi: 10.1109/MNET.2014.6963800

10. Kim H, Feamster N. Improving network management with software defined networking. *IEEE Communications Magazine* 2013; 51(2): 114–119. doi: 10.1109/MCOM.2013.6461195

11. Stallings W. The Internet Protocol Journal Software-Defined Networks and OpenFlow. *The Internet Protocol Journal* 2013; 16(1): 1–40. doi: 10.1097/00004583-201007000-00001

12. Santos J, Wauters T, Volckaert B, De Turck F. Fog computing: Enabling the management and orchestration of smart city applications in 5g networks. *Entropy* 2018; 20(1): 4.

13. Karakus M, Durresi A. Quality of Service (QoS) in Software Defined Networking (SDN): A survey. *Journal of Network and Computer Applications* 2017; 80: 200–218.

14. Oh B, Vural S, Wang N, Tafazolli R. Priority-Based Flow Control for Dynamic and Reliable Flow Management in SDN. *IEEE Transactions on Network and Service Management* 2018; 15(4): 1720-1732. doi: 10.1109/TNSM.2018.2880517

15. Yan J, Zhang H, Shuai Q, Liu B, Guo X. HiQoS: An SDN-based multipath QoS solution. *China Communications* 2015; 12(5): 123–133.

16. Zhang Y, Tang Y, Tang D, Wang W. QOF: QoS Framework Based on OpenFlow. *2015 2nd International Conference on Information Science and Control Engineering* 2015: 380–387.

17. Dijkstra EW. A Note on Two Probles in Connexion with Graphs. *Numerische Mathematik* 1959; 1(1): 269–271. doi: 10.1007/BF01386390

18. Akella AV, Xiong K. Quality of Service (QoS)-Guaranteed Network Resource Allocation via Software Defined Networking (SDN). *2014 IEEE 12th International Conference on Dependable, Autonomic and Secure Computing* 2014: 7–13.

19. Cao S, Tong M, Lv Z, Jiang D. A Study on Application-Towards Bandwidth Guarantee Based on SDN. *2016 IEEE Globecom Workshops (GC Wkshps)* 2016: 1–6.

20. Yen JY. Finding the K Shortest Loopless Paths in a Network. *Management Science* 1971; 17(11): 712–716.

21. Tomovic S, Radusinovic I. Fast and efficient bandwidth-delay constrained routing algorithm for SDN networks. *2016 IEEE NetSoft Conference and Workshops (NetSoft)* 2016: 303–311.

22. Pinto P, Cardoso R, Amaral P, Bernardo L. Lightweight admission control and traffic management with SDN. *2016 IEEE International Conference on Communications (ICC)* 2016: 1–7.

23. Krishna H, Adrichem NLMv, Kuipers FA. Providing bandwidth guarantees with OpenFlow. *2016 Symposium on Communications and Vehicular Technologies (SCVT)* 2016: 1–6.

24. Morin C, Texier G, Phan C. On demand QoS with a SDN traffic engineering management (STEM) module. *2017 13th International Conference on Network and Service Management (CNSM)* 2017: 1–6.

25. Lu Y, Fu B, Xi X, Zhang Z, Wu H. An SDN-Based Flow Control Mechanism for Guaranteeing QoS and Maximizing Throughput. *Wireless Pers Commun* 2017; 97(1): 417–442.

26. Samani A, Wang M. MaxStream: SDN-based Flow Maximization for Video Streaming with QoS Enhancement. *2018 IEEE 43rd Conference on Local Computer Networks (LCN)* 2018: 287–290.

27. D'Oro S, Galluccio L, Mertikopoulos P, Morabito G, Palazzo S. Auction-based resource allocation in OpenFlow multi-tenant networks. *Computer Networks* 2017; 115(318306): 29–41. http://dx.doi.org/10.1016/j.comnet.2017.01.010doi: 10.1016/j.comnet.2017.01.010

28. Oliveira AT, Martins BJC, Moreno MF, Gomes ATA, Ziviani A, Borges Vieira A. SDN-based architecture for providing quality of service to high-performance distributed applications. *International Journal of Network Management* 2019(March): 1–21. doi: 10.1002/nem.2078

29. Moeyersons J, Farkiani B, Bakhshi B, et al. Enabling emergency flow prioritization in SDN networks. *CNSM2019, the 15th International Conference on Network and Service Management* 2019: 1–8.

30. Specifications OS. 1.5. 1. *Open Networking Foundation* 2015; 3.

31. Understanding CoS Two-Color Marking - TechLibrary - Juniper Networks. 2019. https://www.juniper.net/documentation/en_US/junos/topics/concept/cos-ex-series-two-color-marking-understanding.html.

32. Haddock S. Frame Metering in 802.1 Q. 2013.

33. GENI. 2019. https://www.geni.net/.

34. Mininet Overview - Mininet. http://mininet.org/overview/.

35. Open vSwitch. https://www.openvswitch.org/.

36. Mieghem PV, Kuipers FA. Concepts of exact QoS routing algorithms. *IEEE/ACM Transactions on Networking* 2004; 12(5): 851–864.

37. Tuncer D, Charalambides M, Clayman S, Pavlou G. Flexible Traffic Splitting in OpenFlow Networks. *IEEE Transactions on Network and Service Management* 2016; 13(3): 407–420. doi: 10.1109/TNSM.2016.2580666

38. Yu M, Yi Y, Rexford J, Chiang M. Rethinking virtual network embedding: substrate support for path splitting and migration. *ACM SIGCOMM Computer Communication Review* 2008; 38(2): 17–29.

39. Khachiyan LG. Polynomial algorithms in linear programming. *USSR Computational Mathematics and Mathematical Physics* 1980; 20(1): 53–72. doi: 10.1016/0041-5553(80)90061-0

40. Cormen TH, Leiserson CE, Rivest RL, Stein C. *Introduction to algorithms*. MIT press . 2009.

41. Merkel D. Docker: lightweight linux containers for consistent development and deployment. *Linux journal* 2014; 2014(239): 2.

42. Tomonori F. Introduction to ryu sdn framework. *Open Networking Summit* 2013.

43. Grinberg M. *Flask web development: developing web applications with python*. O'Reilly Media, Inc. . 2018.

44. OpenJDK. https://openjdk.java.net/.

45. IBM ILOG CPLEX Optimization Studio. https://www.ibm.com/be-en/marketplace/ibm-ilog-cplex.

46. DuBois P. *MySQL*. New riders publishing . 1999.

47. Internet2 Network Infrastructure Topology. 2018. https://www.internet2.edu/media/medialibrary/2019/04/10/I2-Network-Infrastructure-Topology-All-legendtitle.pdf.

48. Farkiani B, Bakhshi B, Ali MirHassani S. Stochastic virtual network embedding via accelerated Benders decomposition. *Future Generation Computer Systems* 2019; 94: 199–213.

49. Zanna P. Zodiac FX. 2019. https://northboundnetworks.com/collections/zodiac-fx/products/zodiac-fx.

50. Zanna P. Zodiac GX. 2019. https://northboundnetworks.com/collections/zodiac-gx.

51. De Oliveira RLS, Schweitzer CM, Shinoda AA, Prete LR. Using mininet for emulation and prototyping software-defined networks. In: IEEE. ; 2014: 1–6.

52. Fernandes EL, Rojas E, Alvarez-Horcajo J, et al. The Road to BOFUSS: The Basic OpenFlow User-space Software Switch. *CoR* 2019; abs/1901.06699. http://arxiv.org/abs/1901.06699.

How to cite this article: Moeyersons J, Behrooz F, Wauters T Volckaert B, and De Turck F (2020), Towards Distributed Emergency Flow Prioritization in SDN Networks, *Int J Network Mgmt.*, *2020;e2127*. https://doi.org/10.1002/nem.2127