Reinforcement Learning for Slicing in a 5G Flexible RAN

Citation for the original published paper (version of record):
Raza, M., Natalino Da Silva, C., Öhlen, P. et al (2019)
Reinforcement Learning for Slicing in a 5G Flexible RAN
Journal of Lightwave Technology, 37(20)
http://dx.doi.org/10.1109/JLT.2019.2924345

Reinforcement Learning for Slicing in a 5G Flexible RAN

Muhammad Rehan Raza, Carlos Natalino, *Member, IEEE*, Peter Öhlen, Lena Wosinska, *Senior Member, IEEE*, and Paolo Monti, *Senior Member, IEEE*

***Abstract*— Network slicing enables an infrastructure provider (InP) to support heterogeneous 5G services over a common platform (i.e., by creating a customized slice for each service). Once in operation, slices can be dynamically scaled up/down to match the variation of their service requirements. An InP generates revenue by accepting a slice request. If a slice cannot be scaled up when required, an InP has to also pay a penalty (proportional to the level of service degradation). It becomes then crucial for an InP to decide which slice requests should be accepted/rejected in order to increase its net profit.**

This paper presents a slice admission strategy based on reinforcement learning (RL) in the presence of services with different priorities. The use case considered is a 5G flexible radio access network (RAN), where slices of different mobile service providers are virtualized over the same RAN infrastructure. The proposed policy learns which are the services with the potential to bring high profit (i.e., high revenue with low degradation penalty), and hence should be accepted.

The performance of the RL-based admission policy is compared against two deterministic heuristics. Results show that in the considered scenario, the proposed strategy outperforms the benchmark heuristics by at least 55%. Moreover, this paper shows how the policy is able to adapt to different conditions in terms of: *(i)* slice degradation penalty vs. slice revenue factors, and *(ii)* proportion of high vs. low priority services.

***Index Terms*—5G, cloud RAN, dynamic slicing, flexible RAN, network function virtualization (NFV), optical networks, reinforcement learning, slice admission control, software defined networking (SDN).**

I. Introduction

THE 5th generation of mobile networks (5G) needs to support a wide variety of services over a shared network infrastructure, i.e., in order to improve the resource usage efficiency and to lower the infrastructure cost [2]. This can be enabled by *network slicing*, i.e., a key component of 5G systems [3]. Thanks to concepts such as software defined networking (SDN) and network function virtualization (NFV), an infrastructure provider (InP) can virtualize its resources (i.e., create slices), and share them among different tenants or service providers (SPs). Each SP then uses these slices to provision its services (i.e., usually one slice per service).

Slices are created according to the specific requirements (e.g., latency, capacity, reliability, etc.) of the corresponding services. In the presence of temporal and/or spatial variations of such requirements, an InP can improve its resource usage efficiency by dynamically scaling up/down the provisioned slices in order to match the variations of service requirements [2]. However, if a slice cannot be scaled up when needed (i.e., due to resource contention), an InP has to pay a penalty proportional to the degradation level experienced by the corresponding service. This aspect becomes crucial when the infrastructure resources are shared among services with different priorities. In this scenario, the revenue generated by an InP (i.e., by accepting a slice request) and the penalty incurred (i.e., due to degradation) are proportional to the service priority. This means that large revenues are generated by accepting slices of high priority services. However, if degradation is experienced at any point in time, an InP will also have to pay a very large penalty, which, in turn, will have an impact on the net profit of InP.

In order to maximize the profit on an InP, the challenge is two-fold. An InP needs to: *(i)* accept as many slice requests as possible (i.e., to increase revenue), while at the same time, *(ii)* match the variations of service requirements of the slices in operation as closely as possible (i.e., to limit the degradation penalties). In this respect, it becomes crucial to have an intelligent slice admission policy that accepts only those slice requests which generate high revenue and which, most likely, will experience (almost) no service degradation. One way of implementing such a policy is to apply machine-learning-based techniques, more specifically reinforcement learning (RL) [4]. RL-based algorithms learn about the association between actions taken in a given environment and the rewards associated to them. RL methods are particularly interesting because they can learn by interacting directly with the environment to which they are applied to, without the need of any prior knowledge or real-world dataset, which are not always easy to retrieve.

The application of RL-based algorithms for improving the performance of communication networks has recently gained

Manuscript received October 15, 2018. A preliminary version of this work was presented at ECOC 2018 [1]. This work was supported by the Kista 5G Transport Lab (K5) project funded by VINNOVA and Ericsson, and by the Celtic-Plus sub-project SENDATE EXTEND funded by Vinnova.

M. R. Raza, C. Natalino, L. Wosinska, and P. Monti were with the School of Electrical Engineering and Computer Science, KTH Royal Institute of Technology, Sweden. M. R. Raza is now with Ericsson, Stockholm, Sweden (e-mail: muhammad.rehan.raza@ericsson.com). C. Natalino, L. Wosinska and P. Monti are now with Chalmers University of Technology, Gothenburg, Sweden (e-mail: carlos.natalino@chalmers.se; wosinska@chalmers.se; mpaolo@chalmers.se).

P. Öhlen is with Ericsson Research, Stockholm, Sweden. (e-mail: peter.ohlen@ericsson.com)

interest from both academia and industry. Most of these works focus on resource scheduling [5, 6, 7] and/or assignment optimization problems [8, 2]. On the other hand, to the best of our knowledge, there are no works that apply RL to solve the slice admission problem.

This paper proposes an RL-based slice admission policy aimed at maximizing the profit of an InP in the presence of services with different priorities, i.e., a typical scenario in 5G networks. The proposed strategy decides which are the slices that should be accepted by learning how to maximize revenue (i.e., by accepting as many high revenue slice requests as possible) while minimizing penalty (i.e., by rejecting those slice requests for which the expected revenue is less than the penalty they would likely incur and/or those slice requests that would likely cause degradation for other running services).

The use case considered in the paper is a 5G flexible RAN [9], where services from different mobile SPs (MSPs) are virtualized over the same RAN infrastructure. Two classes of services are considered: high priority (HP) services (i.e., with strict latency constraints and high revenue/penalty) and low priority (LP) services (i.e., with non-strict latency constraints and low revenue/penalty). The paper presents a thorough analysis in terms of: *(i)* different values of slice degradation penalty vs. slice revenue factors, and *(ii)* different proportions of HP vs. LP services. The performance of the proposed RL-based slice admission policy is compared against a set of deterministic heuristics. Simulation results show that in the use case under exam, the proposed RL-based slice admission policy outperforms the benchmark heuristics by at least 55%.

The rest of paper is organized as follows. Sec. II presents a literature review. Sec. III presents the system architecture and more details about the use case under exam. Sec. IV presents how the RL agent has been designed for optimizing the slice admission decisions. Sec. V presents a number of performance evaluation results considering different scenarios. Finally, Sec. VI provides some concluding remarks.

II. Literature Review

Network slicing has received increasing attention due to its numerous benefits. Meanwhile, the use of RL-based network control and management strategies has gained interest recently due to their promising performance. This section first focuses on works tackling the network slicing problem (i.e., slice admission and scaling) using deterministic algorithms. Then, it reviews a number of works applying RL for resource scheduling/assignment problems.

The slice admission problem can be solved using deterministic algorithms. Some of the literature refers to approaches where incoming network slice requests are put into one or multiple queues when not enough resources are available in the network. For example, the authors in [7] consider a number of heterogeneous queues (i.e., different queues for different request priorities) and devise a multi-queuing controller for slice admission that maximizes the overall network utilization. The performance of the proposed controller is benchmarked against two simple strategies, i.e., first-come-first-served and slice-type-based approaches. The results show that the proposed controller outperforms the benchmarks, especially under heavy load conditions. The authors in [6] investigate the “impatient behavior” of tenants in a multi-queue slice admission control scenario. More precisely, a tenant may choose to cancel its slice request and ask the InP to remove it from the queue(s) if it has to wait more than a certain amount of time. Results highlight how making the information about the queue status fully available to the awaiting tenants creates benefits in terms of resource efficiency, waiting time and, in turn, overall revenue values. The authors in [5] derive similar conclusions as in [6] for the case of slice requests from Internet-of-Things (IoT) tenants, i.e., the InPs can have bi-directional negotiations with IoT tenants to allocate network resources efficiently.

Another way to address the slice admission problem (i.e., similar to the one considered in this paper) is to discard slice requests immediately if they cannot be accepted. This, on the other hand, leads to a loss of potential revenue for the InP. The authors in [8] propose a slice admission control algorithm which uses the information from a forecasting module (i.e., predicting future traffic levels) during the admission control phase. Results show significant gains in terms of network utilization as compared to a scenario when the forecasted information is not available. The work in [10] also presents a slice admission strategy based on traffic predictions, i.e., an incoming slice request is accepted only when it is estimated that no service degradation will take place for both the incoming slice request and the slices already in operation. Results show that the proposed strategy can increase the net profit of InPs by up to 50.7% as compared to a slice admission policy that does not use BDA predictions. The authors in [11] and [12] propose the concept of slice overbooking where more slice requests are admitted than the overall system capacity in order to maximize the profit of InPs. Results show that slice overbooking can provide up to 3-times higher profit compared to when overbooking schemes are not employed. The work in [12] presents an optimal slice admission algorithm maximizing the profit of InPs. However, the algorithm has a very high computational cost making it impractical for real scenarios. An adaptive algorithm for practical use based on Q-learning is also presented. It is shown that this algorithm achieves close to optimal performance. It is worth noting here that the work in [12] encourages the use of ML instead of an optimal algorithm in real scenarios. Furthermore, the use-case is significantly different from the one considered in this paper. For example, the footprint of slice requests is fixed (i.e., they cannot be scaled up/down), and an incoming slice request is always rejected if the requested resources are not available at the time of arrival. In contrast, the use-case considered in this paper involves dynamicity in the footprint of network slices, and the incoming slice requests with high profit can be admitted even when not enough resources are available (i.e., expecting that other existing slices will scale-down/depart in future).

Applying RL-based algorithms for improving the performance of communication networks has gained interest from both academia and industry. Most of these works focus on resource scheduling and/or routing optimization problems. For instance, the work in [14] presents an RL-based radio resource scheduling policy that maximizes the probability of meeting Quality-of-Service (QoS) requirements in a 5G radio access network (RAN). The authors in [15] present an RL-based framework for the power-efficient resource allocation in cloud RANs. The work in [16] proposes an RL-based strategy for scheduling resources in a multi-tenant network with mobile and cloud SPs. The authors in [17] present an RL agent that performs routing optimization by automatically adapting to current traffic conditions with the goal of minimizing the end-to-end latencies of all connections routed in the network. The authors in [18] present an RL agent for the cognitive and autonomous routing of lightpaths in elastic optical networks. The work in [19] proposes an RL-based routing policy for provisioning connectivity services with different QoS requirements. Finally, when looking specifically at use cases related to network slicing, the authors in [20] present a preliminary investigation of the benefits of using RL for intelligently scaling up/down slices according to traffic patterns of mobile users. On the other hand, to the best of our knowledge, there are no works that apply RL to solve the slice admission problem considering the dynamicity of slices, i.e., scaling up/down of slices.

III. System Architecture and Use Case Definition

This section describes the flexible RAN architecture as well as the use case considered in the paper.

A. *Flexible RAN System Architecture*

Figure 1 presents the system architecture considered in this work. The data plane comprises a flexible RAN, and the control plane is based on an orchestrator that performs cross-domain management of radio, transport, and cloud resources via different controllers [1].

The flexible RAN architecture includes two types of sites for running the radio functions, i.e., central offices (COs) and regional data centers (RDCs) [9]. The COs are located close to the mobile users, and the RDCs are deployed at distant locations. In order to be closer to the users, COs are usually deployed in more locations than the RDCs, and also have lower capacity. COs and RDCs are connected via an optical backhaul (OBH) network.

Macro and small cells are deployed according to the cloud radio access network (C-RAN) concept, where remote radio units (RRUs) and baseband processing functions (BPFs) are interconnected through a C1 interface [9]. Each BPF is connected to a virtualized packet processor (vPP) function, which carries the data to/from the packet gateway (PGW). The BPFs run on special purpose processors at the COs (i.e., close to RRUs) in order to meet the stringent latency requirements (i.e., 1 [ms]) of the C1 interface [9]. On the other hand, vPPs and PGWs are virtual network functions (VNFs) running on general purpose processors (GPPs), which can be instantiated either at the COs, or at the RDCs, depending on the service latency constraints. In the latter case, vPPs and PGWs are connected over the OBH network. The service latency constraint also governs whether service specific VNFs (i.e., referred to as generic application (APP) in Fig. 1) can be placed at the COs or at the RDCs. The fronthaul connections (i.e., RRU-BPF) are fixed. However, the backhaul connections and the VNFs (corresponding to vPP, PGW, APP) can be established on-the-fly as per service requirement.

B. *Use Case Description*

It is assumed that the MSPs request the orchestrator (Fig. 1) to provision RAN resources for different types of services (i.e., one slice per service). An RL agent inside the orchestrator is trained to decide about the admission of slice requests corresponding to different services. In the use case under exam, two types of services are considered, i.e., LP and HP. An LP service comes with non-strict latency constraints (e.g., on-demand media streaming, file transfer) and requires a slice of GPPs placed at either the CO or the RDC, as well as connectivity resources in the OBH network, i.e., the green service in Fig. 1. An HP service, on the other hand, comes with strict latency constraints (e.g., remote surgery [21]) and asks for a slice with GPPs placed only at the CO, i.e., the red service in Fig. 1. Sometimes, an HP service might also require a few GPPs in RDCs as well as connectivity resources in the OBH network, e.g., to fetch new content in the CO. Since a CO can host only a limited number of GPPs (i.e., as compared to a RDC), the GPPs at the COs are more precious resources and hence more costly to use compared to the ones at the RDCs. Consequently, an HP service generates more revenue than an LP one.

In the scenario under exam, the resource requirements of a service vary over time. Once a slice has been provisioned, it needs to be scaled up/down to match the temporal variation in the number of required GPPs (i.e., at the CO/RDC) and connectivity resources in the OBH network. The orchestrator is in charge of both the slice admission and the slice scaling processes. If the orchestrator is unable to scale up a slice when required, there is a penalty to be paid proportional to the amount of resources that cannot be provisioned multiplied by a penalty factor that depends on the service type (i.e., an HP service has a higher penalty factor than an LP one).

When the slice of an HP service is accepted, high revenue is

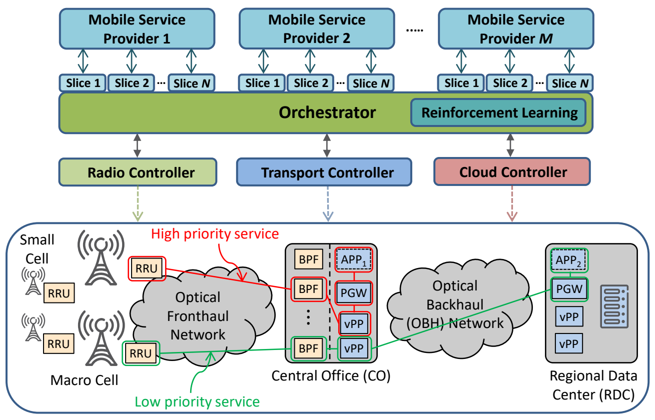


Fig. 1. Flexible RAN system architecture running both HP and LP services.

generated, but the orchestrator needs to make sure that this slice can be scaled up when required. This is to avoid the negative impact of the very high penalty on the generated revenue (where profit = revenue - penalty). Therefore, when deciding about slice admission, the orchestrator has to consider not only the potential revenue generated by provisioning a slice but also the penalty to be paid if the required resources are not available when needed. A way to solve this problem is to consider the possibility to reject the slice requests of some LP services in order to allow (more profitable) HP ones to be admitted in the future and to be scaled up when needed. This can be accomplished via an intelligent slice admission policy that accounts for all these aspects.

Figure 2 depicts the intuition behind the proposed intelligent slice admission policy. We use, for comparison, a simple policy that aims only at accepting as many slices as possible, without taking into account any implications on the service degradation. In both cases, it is assumed that the orchestrator has no prior knowledge of the temporal variations in resource profiles of the slices to be provisioned. It is also assumed to have three resource pools in the network (i.e., one CO, one OBH, and one RDC), with three resource units each.

The figure 2 shows an HP slice request requiring two resource units in the CO at the time of arrival (red profile in the figure), which comes one time unit after an LP slice request that requires one resource unit in each of the resource pools at the time of arrival (green profile in the figure). The simple policy always accepts a slice request if the resources required at that point in time are available, and hence both slice requests are accepted. As a result, when the HP slice needs to be scaled up (i.e., three resource units needed in the CO one time unit after the slice is accepted), enough resources are not available and the service is degraded. This leads to a high penalty to be paid due to the fact that this is an HP service. On the other hand, an intelligent policy might be able to understand that, in this particular instance, the overall profit could be maximized by proactively rejecting the slice of LP service. This leaves more resources free for provisioning and scaling the slice of the HP service without having to pay any degradation penalty.

The next section describes how RL can bring such intelligence into the slice admission process.

IV. Reinforcement Learning Based Slice Admission Policy

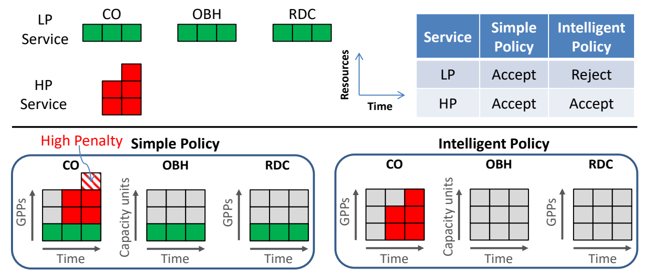
An RL agent is trained to decide whether or not a new slice request should be accepted. The agent is embedded into a slice management loop running at the orchestrator, which is composed of two parts illustrated in Fig. 3. The outer loop (i.e., solid line) includes: slice admission, setup, scaling, tear-down, and reward computation (i.e., for the RL agent). The inner loop (i.e., dashed line) describes the actions taken during the slice scaling process, i.e., during the holding time of a slice.

A slice request is specified in terms of the following parameters: holding time, service priority, number of resources required (i.e., in the CO, OBH, and RDC) at the time the slice is requested, and the location of CO (i.e., corresponding to the fixed RRU-BPF fronthaul connections of the slice). After receiving a slice request, the RL agent makes its decision (i.e., yes/no) about the slice admission. If the slice request is accepted, the orchestrator proceeds with the setup, i.e., it reserves the current resources required by the slice for a duration equal to the slice holding time. The selection of the RDC as well as the path from the CO to the chosen RDC over the OBH network is done by a heuristic algorithm.

After a slice is set up, the orchestrator monitors its resource requirements and decides for a scale up/down when they exceed/fall-below a given threshold γ. During the holding time of a slice, the location of the RDC and the path from the CO to the RDC over the OBH network remain fixed. On the other hand, the amount of resources allocated to them may vary during the slice scaling process. The scaling policy is based on a heuristic algorithm, described in the next section. At each time instance, the orchestrator computes the net profit associated with operating a slice (i.e., sum of the revenue generated by accepting the slice request and the penalty incurred by not being able to scale up the slice when needed).

After the holding time of a slice expires, it is torn down, i.e., all the resources currently allocated to the slice are released. Finally, the total net profit obtained by operating the slice during its holding time is computed, i.e., fed back as reward to RL agent. A high reward makes the agent learn to accept more slice requests of similar type and in similar conditions in the future. On the contrary, a low reward may lead to the rejection of similar slice requests in the future.

As mentioned earlier, the overall objective of the admission



Service	Simple Policy	Intelligent Policy
LP	Accept	Reject
HP	Accept	Accept

$Profit_{simple} = Rev_{LP} + Rev_{HP} - Pen_{HP}$ $Profit_{intelligent} = Rev_{HP} > Profit_{simple}$

Fig. 2. An example of how the proactive rejection of an LP service can help to maximize the profit of an InP.

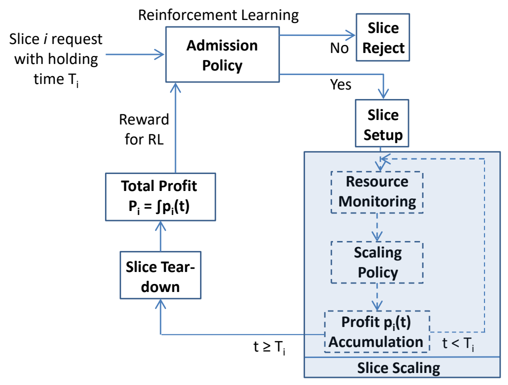


Fig. 3. Slice admission and management loop running at the orchestrator.

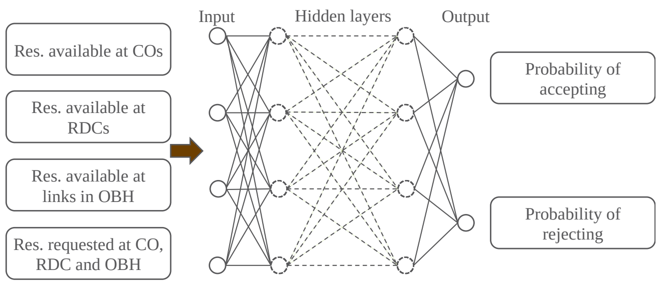


Fig. 4. Stochastic policy network architecture.

policy is to maximize the total profit of the InP. This problem is equivalent to a loss minimization problem, where the loss has two components: *(i)* loss of revenue derived by rejecting the slice requests, and *(ii)* loss derived by not being able to scale up the slices in operation when needed (i.e., the degradation penalty).

Following the above rationale, the objective of the ANN is to minimize the total loss experienced by the InP. The RL agent considered in this work is illustrated in Fig. 4. The RL agent is modeled as a stochastic policy network (PN) [Mao16], which, in turn, uses an artificial neural network (ANN) to represent its stochastic policy. The ANN receives as input: *(i)* an array describing the resources currently available at the COs, the RDCs, and links in the OBH network, and *(ii)* the specific slice request parameters described at the beginning of the section. The ANN comprises a number of fully-connected, hidden layers of neurons. The ANN has two outputs, representing the probability of accepting or rejecting the slice request. The output neurons in the PN use the soft-max activation function, which outputs probabilities in the range [0-1]. The probabilities output by the ANN dictate the action taken by the orchestrator.

The ANN is trained in an episodic manner, where a fixed number of slice requests arrive in each episode. The reward ω for each action is computed as:

$$\omega = \sum_{a \in S} \frac{-l_a}{W}, \qquad (1)$$

where S is the set of all slice requests arrived up to the current time, l_a is the loss incurred by slice request a, and W is the maximum potential revenue that could be generated by a slice request. In summary, ω is the sum of the rewards obtained for all the slice requests (i.e., with each reward the in range of [-1, 0]) up to the current point in time. At the end of an episode, the cumulative reward for all the actions is computed [22]. This is done to ensure that the effect of all the actions taken during an episode has an impact on future decisions made by the ANN. After collecting the set of observations, actions, and rewards from an episode, a training iteration is performed, where the PN is optimized by applying the gradient descent method [16] with the objective of maximizing the reward function (1) (i.e., minimizing the total loss). The gradients are used to update the weights of the ANN, which helps it to take better decisions in the next episode. By gradually increasing the cumulative reward in each episode, the ANN converges to a policy which minimizes the total loss.

V. Performance Evaluation

This section describes the scenario used for the performance evaluation and discusses the results for different cases.

A. *Scenario Description*

The performance of the proposed RL-based slice admission policy is evaluated using a custom-built Python-based event-driven simulator. The simulator uses the NetworkX library [23] for the graph representation and manipulation of network resources, and Keras [24] as the machine learning library for implementing the PN. Three types of events are modeled in the simulator: *arrival*, *departure*, and *scaling*. The inter-arrival time and holding time of slice requests are exponentially distributed. The mean holding time is 24 hours, while the mean inter-arrival time is varied according to different load conditions. The scaling events, i.e., when the slices might need to be scaled up/down, occur periodically with a fixed interval of one hour.

Results are obtained using a 12-node network topology [1] depicted in Fig. 5. Five COs and two RDCs are placed at high degree nodes. It is assumed that the fixed fronthaul connections between RRUs and BPFs in the COs are already established. Hence, a slice may ask for only GPPs in a CO and an RDC, as well as connectivity resources over the OBH network. Moreover, this work assumes that only one MSP generates slice requests for HP and LP services, although this can be generalized to more SPs. The proportion of HP services over the total number of services (i.e., LP and HP) is denoted as pr_{HP}. The number of resources in each CO, in each link in the OBH network, and in each RDC is assumed to be 50 GPPs, 50 capacity units, and 80 GPPs, respectively. The per-hour price paid by an LP service using a resource unit at the CO, and the RDC is $p_{CO} = 4$ cost-units (CUs), and $p_{RDC} = 1$ CU, respectively. The per-hour and per-capacity-unit price for using a path in the OBH network is $p_{OBH} = 2$ CUs. It is assumed that the price of a path is independent of number of hops. The price paid by an HP service s using the same resources as an LP service is higher by an amount proportional to its revenue factor Rf_s. Moreover, an HP service s also incurs a penalty (i.e., in case of degradation) higher than an LP service that is proportional to its penalty factor Pf_s. The values of Rf_s and Pf_s are assumed to be 1 for an LP service and 5 for an HP service. Regardless of the priority, the degradation of an accepted service requiring one resource unit for one hour results in a penalty r times higher than the generated revenue.

The temporal variations of the resource requirements of HP and LP services are modeled using the profiles reported in [10]. The value of γ is set to 60% of the peak value. When a slice is scaled up (i.e. γ is exceeded), an HP service requires

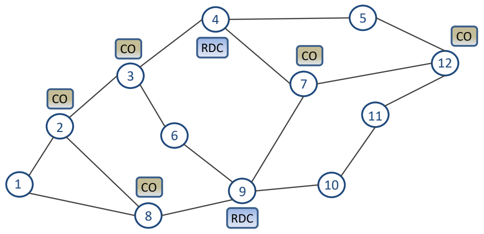
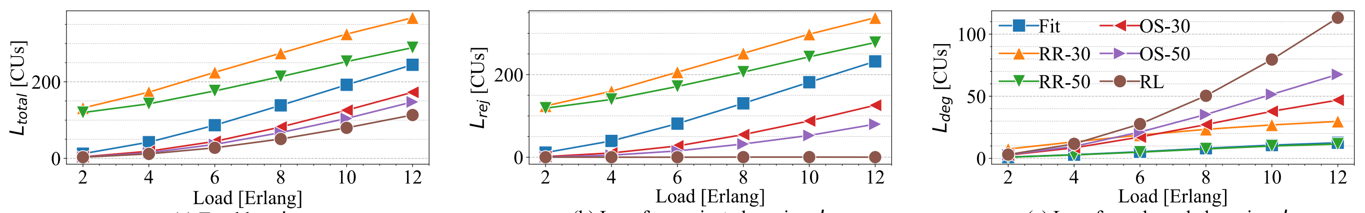


Fig. 5. 12-node network topology with 5 COs and 2 RDCs.



(a) Total loss L_{total} (b) Loss from rejected services L_{rej} (c) Loss from degraded services L_{deg}

Fig. 6. Test results for the baseline scenario (i.e., $r = 1.5$, $pr_{HP} = 50\%$). The total loss L_{total} (a) is the sum of L_{rej} (b) and L_{deg} (c).

20 GPPs in the CO, 5 GPPs in the RDC, and 5 capacity units in the OBH, while an LP service requires 10 GPPs in both the CO and in the RDC, plus 10 capacity units in the OBH. When a slice is scaled down (i.e. requirements go below γ), the number of required resources (for both HP and LP services) is decreased by 5 everywhere. As mentioned earlier, scaling is done using a heuristic algorithm, which adopts an HP first (HPF) policy, i.e., all the HP services are scaled before the LP ones. This policy is used by all the tested admission policy algorithms in order to ensure that the degradation of HP services is minimal.

The performance evaluation metric is the total loss L_{total} experienced by an InP, i.e., the sum of loss derived from rejected services (L_{rej}) and loss from degraded services (L_{deg}), with:

$$L_{rej} = \sum_{s\in R} Rf_s \times ht_s \times \left[\left(Q_{CO,s}(at_s) \times p_{CO}\right) + \left(Q_{RDC,s}(at_s) \times p_{RDC}\right) + \left(Q_{OBH,s}(at_s) \times p_{OBH}\right)\right], \quad (2)$$

$$L_{deg} = \sum_{s\in A} Pf_s \times r \times \int_{t=at_s}^{at_s+ht_s} \left[\left(N_{CO,s}(t) \times p_{CO}\right) + \left(N_{RDC,s}(t) \times p_{RDC}\right) + \left(N_{OBH,s}(t) \times p_{OBH}\right)\right], \quad (3)$$

where R and A denote the set of rejected and accepted services respectively; ht_s represents the holding time of slice s; $Q_{CO,s}(at_s)$, $Q_{RDC,s}(at_s)$, $Q_{OBH,s}(at_s)$ denote the resources required by slice s at the arrival time at_s in CO, RDC, and OBH respectively; $N_{CO,s}(t)$, $N_{RDC,s}(t)$, $N_{OBH,s}(t)$ represent the resources not provisioned to slice s due to the degradation at time t in CO, RDC, and OBH respectively.

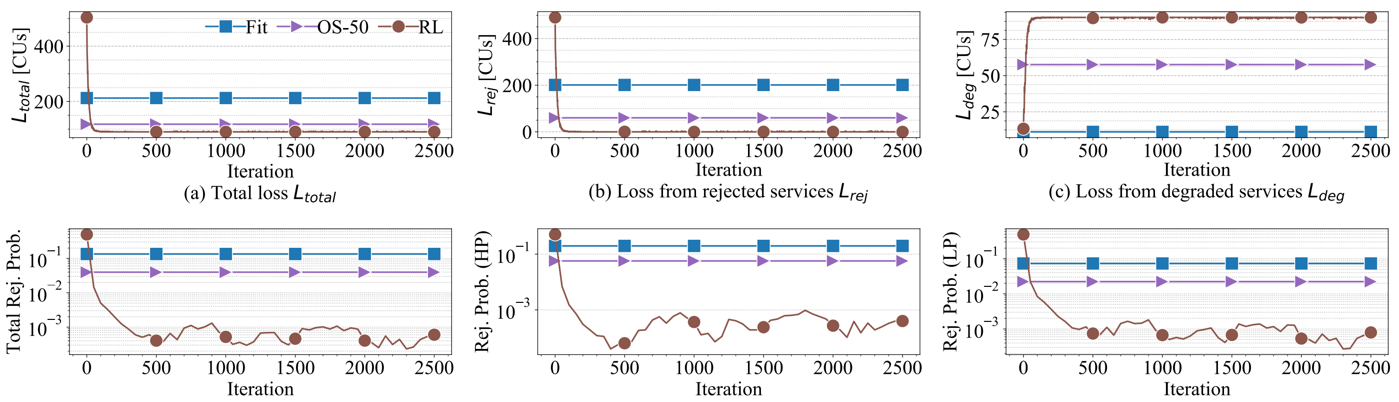
The performance of the proposed RL-based slice admission policy is compared against three benchmark strategies: *fit*, *oversubscription (OS)* and *resource reservation (RR)*. The Fit strategy is a static heuristic that follows a conservative approach and accepts a slice request only if: *(i)* the resources required (i.e., at arrival time) are available at the CO, *(ii)* an RDC with enough resources as well as a path connecting the CO and the RDC with enough capacity are available. The OS strategy inspired by [11, 12] considers that it is possible to multiplex the use of resources over time, assuming the time-varying requirements of slices. The OS strategy is a threshold-based heuristic that considers an overbooking of up to a certain percentage of resources, i.e., when computing available resources an amount higher than 100% is considered. In our work, we consider the cases where the OS allows 30% (OS-30) and 50% (OS-50) overbooking. Finally, the RR strategy is also a threshold-based heuristic that assumes that a percentage of the resources is reserved for the HP services. This strategy is inspired by the fact that, in multi-priority networks, reserving a percentage of the resources for HP services potentially reduces rejection losses from these services. In our work, we consider the reservation of 30% (RR-30) and 50% (RR-50) of the resources for HP services. For all the admission policies, when a slice request is accepted, the closest available RDC (with the shortest available path) is chosen.

For each value of the load, the RL agent is trained for 2500 iterations using 25 different sets, each one comprising 200 slice requests generated synthetically. The test results are obtained by averaging the results from 25 different sets of 3000 slice requests, which are different from the ones seen by the ANN during the training phase. The confidence interval of the test results (i.e., calculated with a 95% confidence level) represents 3% of the value of the total loss for the Fit strategy at 12 Erlangs. The designed ANN contains four hidden layers with 40 neurons each, with ReLU as the activation function. The ANN is trained with a learning rate of 0.0001.

The next sub-sections presents an analysis of the simulation results under a number of representative scenarios.

B. *Results for Baseline Scenario*

For the baseline scenario, the simulation parameters are set to $r = 1.5$, $pr_{HP} = 50\%$, i.e., the degradation penalty is 1.5



(a) Total loss L_{total} (b) Loss from rejected services L_{rej} (c) Loss from degraded services L_{deg}

(d) Total rejection probability (e) Rejection probability for HP (f) Rejection probability for LP

Fig. 7. Training results for the baseline scenario (i.e., $r = 1.5$, $pr_{HP} = 50\%$) at 12 Erlangs. Results are averaged over 25 different sets of 200 slice requests.

times higher than the revenue factor, and the proportion of HP and LP services is the same in each set of slice requests. Figure 6 presents the test results comparing the value of the average total loss for different values of the load. Fit presents loss values that are between RR and OS (Fig. 6(a)). This value is mainly dominated by the rejection loss (Fig. 6(b)), which is high due to the conservative approach taken by Fit at the admission control, i.e., it only accepts if current resource requirements can be met. As a result, the scaling loss is very low (Fig. 6(c)), but not enough to compensate for the high rejection loss. It can be observed that by reserving 30% of the resources for HP services, the RR-30 strategy leads to the highest value of L_{total}. This is caused by an increase in the rejection loss (Fig. 6(b)) that is not sufficiently compensated by the decrease in scaling loss (Fig. 6(c)). RR-50 has better performance in terms of both lower rejection and scaling losses in comparison to RR-30. At low load, RR-50 performs similar to RR-30. As the load increases, RR-50 steadily increases difference from RR-30 and approaches Fit. The reason is twofold: while RR-50 reserves more resources for HP services than RR-30, there are more HP services to use the reserved resources at high load. Meanwhile, less LP services are accepted (i.e., only 50% of resources are available for LP services, which reduces the competition for resources during scaling, causing the scaling loss reduction). The OS admission policy presents better performance than Fit. As expected, compared with Fit, OS trades a lower rejection loss for a (possibly) higher scaling loss. In this scenario, OS-50 presents a lower total loss because of a slightly higher scaling loss that is compensated by a lower rejection loss. On the other hand, at all load values, RL performs better than the heuristics in terms of L_{total}, with 53% improvement over Fit, 60% over RR-50, and 23% over OS-50, at high load conditions. This is because RL learns that it can accept all the slice requests by trading a relatively small increase in L_{deg} with a significant decrease of L_{rej}.

Figure 7 depicts how the RL agent learns over the training iterations, for a load value of 12 Erlangs. In the figure, the rejection probability for HP/LP services is averaged over the corresponding number of HP/LP slice requests. The total rejection probability is averaged over the total number of slice requests. At the beginning, i.e., iteration 1, RL behaves similarly to a random policy, i.e., no knowledge about the system dynamics. After around 100 training iterations, RL learns that L_{total} can be decreased by accepting more HP services, i.e., the rejection probability of HP services (Fig. 7(e)) drops to 10^{-3} at around 100 iterations. Afterward, RL keeps on trying to decrease L_{total} by accepting some of the LP services until it learns, after 400 iterations, that almost all the LP services can be accepted (even if this has a minor impact on the L_{total}). After 500 iterations, L_{total} converges to a minimum, although RL keeps on trying to further improve the value of L_{total} by slightly adjusting the rejection probabilities (Figs. 6(e) and 6(f)), but without any significant improvement.

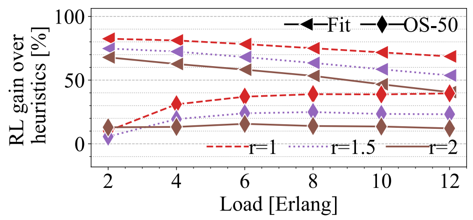


Fig. 8. Gain in terms of L_{total} when RL is compared to Fit and OS-50 for different values of r and with $pr_{HP} = 50\%$.

C. *Results for Varying the Value of r*

Figure 8 presents the gain (in percentage) in terms of L_{total} achieved by RL when compared to Fit and OS-50, for different values of r. In the figure, $r = 1.5$ refers to the baseline scenario (Figs. 6 and 7). With $r < 1.5$, RL achieves higher gains over Fit and OS-50 because of the degradation of an accepted service results in a lower penalty than the baseline scenario. On the other hand, when $r > 1.5$ the gain of RL over Fit and OS-50 decreases. In this case, L_{deg} has a higher contribution to L_{total}, and more careful acceptance decisions need to be taken. Still, RL reduces the L_{total} by at least 12% for $r = 2$.

Figure 9 presents the training results at a load of 12 Erlangs with $r = 2$. Compared to the training results in Fig. 7 (i.e., the baseline scenario), RL still learns first to accept more HP slice requests in order to avoid high rejection losses, reaching 10^{-3} in around 100 iterations (Fig. 9(e)). At the end of the training, RL learns that all HP slice requests can be accepted. On the other hand, RL is more conservative while accepting LP services, reaching 10^{-3} in around 500 iterations (Fig. 9(f)). This is because of the rejection of LP slice requests does not have a significant impact on L_{rej}, but it potentially reduces

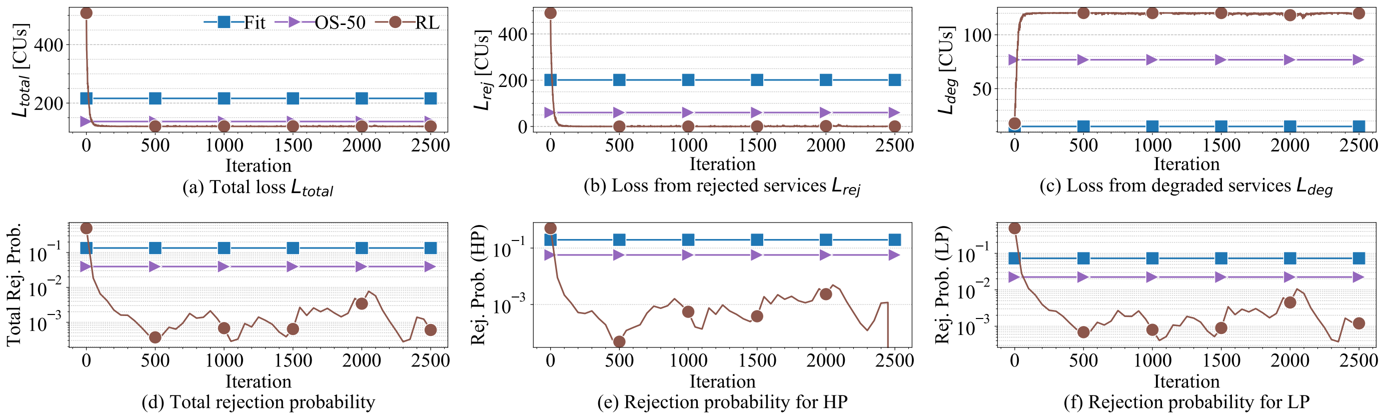


Fig. 9: Training results at 12 Erlangs with $r = 2$ and $pr_{HP} = 50\%$.

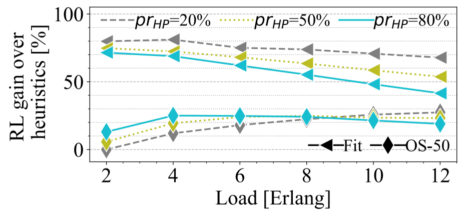


Fig. 10. Gain in terms of L_{total} when RL is compared to Fit for different values of pr_{HP} and with $r = 1.5$.

L_{deg}, i.e., mainly driven by the degradation of HP services.

D. *Results for Varying the Value of pr_{HP}*

Figure 10 presents the gains (in percentage) in terms of L_{total} when RL is compared to Fit and OS-50 for different values of pr_{HP} (i.e., the proportion of HP services in each set of slice requests), where $pr_{HP} = 50\%$ refers to the baseline scenario. When the value of $pr_{HP} < 50\%$, the gain of RL over Fit becomes higher. This is because the RL is able to achieve lower L_{deg}, as less HP services (having higher Pf_s than LP services) are competing for resources during scaling. On the other hand, when $pr_{HP} > 50\%$, there is more competition for resources. In this scenario, degradation is likely to happen more often leaving fewer opportunities for significant improvements over Fit. This can be attributed to: *(i)* an increase in the value of L_{deg} as more HP services experience degradation, and *(ii)* an increase in the value of L_{rej} as more LP services are rejected in order to create space for a higher number of HP services. Still, RL achieves at least 48% lower L_{total} compared to Fit. When comparing OS-50 with RL, there is a shift in trends when load increases. At low load and with a low pr_{HP}, OS-50 performs closer to RL, as the lower competition for resources and a lower percentage of HP makes it for an easy to solve simple scenario. However, at high load, OS-50 faces higher competition of resources, and accepting HP services becomes more problematic. At high load, OS-50 benefits more from higher pr_{HP}. RL is able to adapt to the different load conditions, with gains over OS-50 reaching at least 18% lower L_{total} at high load, and up to 25% at medium load.

Figure 11 presents the training results at a load of 12 Erlangs with $pr_{HP} = 80\%$. Compared to the training results in Fig. 7, the losses are much higher due to the higher number of HP service requests. Moreover, the RL agent learns to reject more LP services (Fig. 11(f)), i.e., nearly 10^{-3} for the baseline scenario and 10^{-2} for $pr_{HP} = 80\%$. This is because the RL tries to free up some space for the HP services such that most of them can be accepted (Fig. 11(e)) to minimize the value of L_{total}.

VI. Conclusion

This paper presents an RL-based admission policy that can be used in a 5G flexible RAN where multiple MSPs ask for resource slices to accommodate services with different QoS constraints (i.e., LP services with non-strict latency and HP services with strict latency requirements). The proposed policy aims at maximizing the profit of an InP. In the study presented in the paper, the profit maximization objective has been converted in the equivalent loss minimization counter-part, where the loss experienced by an InP is defined as the sum of the loss from rejected services (i.e., potential revenue loss) and the loss from degraded services (i.e., when a resource slice cannot be scaled up when needed).

Simulation results show that, in the use case under exam, the proposed RL-based policy achieves at least 50% lower loss when compared to static heuristics, and at least 23% lower loss when compared to threshold-based heuristics. This is because the RL learns to selectively reject some of the services that generate low revenues (i.e., LP) in favor of the ones that are more profitable (i.e., HP). Moreover, the RL is able to adapt its behavior when the penalty due to service degradation becomes higher than the loss of revenue due to slice rejection, or when the proportion of HP and LP services changes. This highlights that the RL interacts with the system to understand different parameters, and learns to adapt its policy accordingly.

References

[1] M. R. Raza, *et al.*, “A Slice Admission Policy Based on Reinforcement Learning for a 5G Flexible RAN,” in *Proc. Europ. Conf. Opt. Commun.*, Sep. 2018.
[2] M. R. Raza, *et al.*, “Dynamic slicing approach for multi-tenant 5G transport networks,” *IEEE/OSA J. Opt. Commun. Netw.*, vol. 10, no. 1, pp. A77-A90, Jan. 2018.
[3] R. Vilalta, *et al.*, “Optical networks virtualization and slicing in the 5G era,” in *Proc. Opt. Fiber Commun. Conf.*, Mar. 2018.
[4] R. S. Sutton, *et al.*, “Reinforcement learning: An introduction,” *MIT Press Cambridge*, vol. 1, no. 1, 1998.
[5] V. Sciancalepore, et al., “Slice as a Service (SlaaS) Optimal IoT Slice Resources Orchestration,” in Proc. IEEE Global Commun. Conf., Dec. 2017, pp. 1-7.
[6] B. Han, et al., “Rational Impatience Admission Control in 5G-sliced Networks: Shall I Bide my Slice Opportunity?,” arXiv:1809.06815v3,

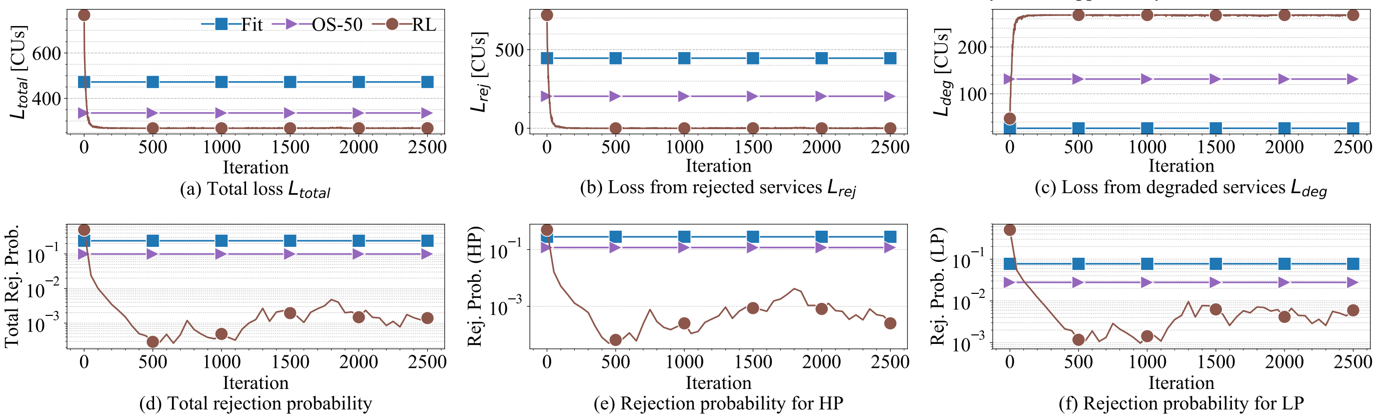


Fig. 11. Training results at 12 Erlangs with $r = 1.5$ and $pr_{HP} = 80\%$.

Oct. 2018.

[7] B. Han, et al., "A Utility-Driven Multi-Queue Admission Control Solution for Network Slicing", arXiv:1901.06399v1, 2019.

[8] V. Sciancalepore, et al., "Mobile traffic forecasting for maximizing 5G network slicing resource utilization," in Proc. IEEE INFOCOM, May 2017, pp. 1-9.

[9] E. Westerberg, "4G/5G RAN architecture: how a split can make the difference," *Ericsson Tech. Review*, 2016.

[10] M. R. Raza et al., "A Slice Admission Policy Based on Big Data Analytics for Multi-Tenant 5G Networks," J. Lightw. Technol., vol. 37, no. 7, pp. 1690-1697, Apr. 2019.

[11] J. X. Salvat, et al., "Overbooking network slices through yield-driven end-to-end orchestration," in Proc. ACM 14th International Conference on emerging Networking EXperiments and Technologies (CoNEXT), Dec. 2018, pp. 353-365.

[12] L. Zanzi et al., "OVNES: Demonstrating 5G network slicing overbooking on real deployments," in Proc. IEEE INFOCOM Workshops, Apr. 2018, pp. 1-2.

[13] D. Bega et al., "Optimising 5G infrastructure markets: The business of network slicing," in Proc. IEEE INFOCOM, May 2017, pp. 1-9.

[14] I. S. Comsa, *et al.*, "QoS-Driven Scheduling in 5G Radio Access Networks - A Reinforcement Learning Approach," in *Proc. IEEE Global Commun. Conf.*, 2017.

[15] Z. Xu, *et al.*, "A deep reinforcement learning based framework for power-efficient resource allocation in cloud RANs," in *Proc. IEEE Internat. Conf. on Commun.*, 2017.

[16] C. Natalino, *et al.*, "Machine Learning Aided Resource Orchestration in Multi-Tenant Networks", in *Proc. IEEE Summer Topicals*, Jul. 2018.

[17] G. Stampa, *et al.*, "A Deep-Reinforcement Learning Approach for SDN Routing Optimization," *arXiv:1709.07080v1*, 2017.

[18] X. Chen, *et al.*, "Deep-RMSA: A Deep-Reinforcement-Learning Routing, Modulation and Spectrum Assignment Agent for Elastic Optical Networks," in *Proc. Opt. Fiber Commun. Conf.*, Mar. 2018.

[19] C. Natalino, *et al.*, "Machine-Learning-Based Routing of QoS-Constrained Connectivity Services in Optical Transport Networks", in *Proc. OSA Adv. Photon. Congr., NETWORKS*, Jul. 2018.

[20] Z. Zhao, *et al.*, "Deep Reinforcement Learning for Network Slicing," *arXiv:1805.06591v2*, 2018.

[21] O. Yilmaz, "5G Radio Access for Ultra-Reliable and Low-Latency Communications," *Ericsson Research Blog*, 2015.

[22] H. Mao, *et al.*, "Resource Management with Deep Reinforcement Learning," in *Proc. ACM Workshop on Hot Topics in Networks*, 2016.

[23] Aric A. Hagberg, *et al.*, "Exploring network structure, dynamics, and function using NetworkX," in *Proc. Python in Science Conference (SciPy2008)*, pp. 11–15, Aug. 2008.

[24] François Chollet, *et al.*, "Keras," https://keras.io.