Learning the Optimal Controller Placement in Mobile Software-Defined Networks

Iordanis Koutsopoulos
Department of Informatics
Athens University of Economics and Business
Athens, Greece

***Abstract*—We formulate and study the problem of *online learning* of the optimal controller selection policy in mobile Software-Defined Networks, where the controller-switch round-trip-time (RTT) delays are unknown and time-varying. Static optimization approaches are not helpful, since delays vary significantly (and sometimes, arbitrarily) from one slot to another, and only RTT delays from the current active controller can be easily measured. First, we model the sequence of RTT delays across time as a stationary random process so that the value at each time slot is a sample from an unknown probability distribution with unknown mean. This approach is applicable in relatively static network settings, where stationarity can be assumed. We cast the problem as a stochastic multiarmed bandit, where the arms are the different controller choices, and we fit different bandit algorithms to that setting, such as: the Lowest Confidence Bound (LCB) algorithm by modifying the known Upper Confidence Bound (UCB) one, the LCB-tuned one, and the Boltzmann exploration one. The first two are known to achieve sublinear regret, while the last one turns out to be very efficient. In a second approach, the random process of RTTs is non-stationary and thus cannot be characterized statistically. This scenario is applicable in cases of arbitrary mobility and other dynamics that affect RTT delays in an unpredictable, adversarial manner. We pose the problem as an adversarial bandit that can be solved with the EXP3 algorithm which achieves sublinear regret. We argue that all approaches can be implemented in an SDN environment with lightweight messaging. We also compare the performance of these algorithms for different problem settings and hyper-parameters that reflect the efficiency of the learning process. Numerical evaluation shows that Boltzmann exploration achieves the best performance.**

I. Introduction

Software Defined Networking (SDN) has become a game-changer in network traffic engineering and management in the last decade. SDN separates the control and data plane functionalities so that the former are placed in dedicated nodes, the controllers, while data-plane forwarding is executed by SDN-enabled switch nodes according to their flow tables.

SDN has seen many success stories in wire-line core networks, and later it expanded to the wireless last mile of infrastructure-based networks, e.g. Long Term Evolution (LTE) cellular and WiFi [1]. The work [2] suggested the deployment of software-defined Radio Access Networks (RANs) through a global controller that centrally manages many base stations (BSs). This work suggested the separation of control decisions into those that require full state information and need to be taken at the controller, and those that need fast response and are assigned to BSs. However, steps towards applying SDN in mobile ad hoc networks (MANETs) have been at a much slower pace, mainly because of the high variability, mobility and link volatility in MANETs, and the decentralized nature of their basic algorithms which come in stark opposition with the centralized fashion of decisions in SDN.

Nevertheless, a number of proposals and research efforts in the last few years argue in favor of feasibility of SDN in MANET environments [3]-[5]. MANET traditional application scenarios such as tactical networks, disaster rescue, and sensor networks are still growing. MANETs are also at the core of modern settings such as device-to-device, unmanned air (or sea) vehicle (UAV, USV), and vehicular ad hoc networks (VANETs). Through its centralized nature and separation of control and data planes, SDN can offer its centralized point of view to network control. There are two main directions towards integrating SDN to MANETs. First, MANET distributed routing protocols can complement SDN-based control and may substitute it when conditions dictate it, e.g. when connection to a controller is lost. Second, the strict separation of control and data planes can be relaxed, and a portion of the centralized control exercised by a controller can be delegated to switches.

The basic functionality of an SDN controller is to configure the flow forwarding rules for SDN switches. An SDN switch queries the SDN controller using PACKET-IN messages for required forwarding actions to follow for new flows it encounters, for which the rules are not already included in its table. The controller responds back through the control channel, and a rule is installed in the switch table that provides a mapping between packet header fields and actions to execute upon packet match. Essentially, the controller runs centralized optimization algorithms whose output determines the flow (forwarding) tables of SDN switches or other rules about forwarding, blocking, modifying, and processing incoming packets. The SDN controller is also responsible for discovering and maintaining network topology information by sending PACKET-OUT messages. Further, the controller continuously gathers network state information and other statistics.

For these controller functionalities, the controller-to-switch *round-trip time* (C-S RTT) delay is crucial. RTT affects delays for flow setup, route updates and fault management, as well as the system abilities for fast topology discovery and for ensuring given quality-of-service (QoS) by adapting to different network variations. RTT in turn depends on *controller selection* or *placement*. The SDN controller placement problem has received a lot of attention in the last decade [6],

[7]. Different objectives have been studied, and control-plane latency (delay) is one of the most prominent ones.

In this work, we study the problem of *selecting a single node as controller* in a MANET environment. We are interested in finding the controller placement that minimizes the average RTT delay for generated requests. In MANETs, the controller selection problem becomes more crucial since nodes are highly volatile and oftentimes mobile, while wireless links have time-varying quality and lower transmission rates compared to wire-line ones. In addition, the controller-to-switch path via the control channel can be multi-hop, and this increases RTT [8].

Almost all approaches e.g. [8] tackle the *static* version of the problem. They assume that the delays between a controller and a switch are known, and they solve an integer programming optimization problem. If one controller is to be selected, this is the node with the smallest average delay to other nodes. If more than one controllers need to be selected, then, besides the controllers, the algorithm needs to find the set of switches associated with each controller.

However, in MANET settings, static optimization does not seem to be applicable or offer useful insights. First and foremost, it is not easy at any given time to measure the RTT delay between any pair of nodes in the network so as to take decisions. This is because, through the SDN protocol, it is possible to measure only the RTT delays between the switches and the *active* controller since only the active controller can send PACKET-OUT messages to other nodes, on which RTT can be measured. Second, even if such measurements were feasible through protocol modification so that all nodes could send PACKET-OUT messages, the information about RTT delays at a time slot t would not be useful in the controller selection decision at time slot $t+1$, since delays are likely to vary significantly (and sometimes, arbitrarily) from one slot to another in wireless settings.

Furthermore, RTT is hard to model, as it is the cumulative result of several factors such as the shared medium, interference, wireless channel propagation, mobility, and time-varying connectivity and path lengths, and it becomes hard to model each of each factors. Data-driven supervised learning approaches are also hard to realize, since they rely on availability of massive datasets to train Machine Learning models, which is hard to assume, especially in a tactical scenario.

In this work, we take a different approach to the controller selection problem by departing from the setting where delays are known at decision time at each time slot. We address the problem of *online learning* of the optimal controller selection policy in a network where RTT delays are unknown and varying with time. According to a first approach, we model the sequence of RTT delays across time as a *stationary* random process so that its value at each time slot is a sample from an unknown probability distribution with unknown mean. This approach is applicable in settings that are relatively static or have canonical mobility, in which stationarity can be assumed. Tactical networks are a primary example where such settings arise. At each time slot, only the RTT delays between the current active controller and the switches can be measured. We are interested in learning which node, if selected as controller, leads to an RTT probability distribution with the smallest mean. In a second approach, the random process of RTT delays is *non-stationary* and thus cannot be characterized statistically. This scenario is applicable in cases of arbitrary mobility and other dynamics that affect RTT delays in an unpredictable, adversarial manner. For both cases, we focus on minimizing regret, i.e. the difference between the average RTT delay achieved by our selection policy and the delay of the optimal fixed selection we would have made if we knew a priori the random process evolution.

This work contributes to the literature as follows:

- When RTT delays can be statistically characterized but have unknown means, we cast the problem of optimal controller selection as a stochastic multiarmed bandit, where arms are the different node choices as controller. We fit different bandit algorithms to the problem setting, such as the Lowest Confidence Bound (LCB) algorithm, the LCB-tuned one and the Boltzmann exploration one. The first two are known to achieve sublinear regret, while the last one turns out to be the most efficient.
- When RTT delays are arbitrary and cannot be characterized statistically, we pose the problem as an adversarial bandit that can be solved with the EXP3 algorithm that achieves sublinear regret.
- We argue that these approaches can be implemented in an SDN environment with lightweight messaging.
- We compare the performance of these algorithms for different settings in terms of learning fast the optimal controller selection. Boltzmann exploration is shown to achieve the best performance.

The rest of the paper is organized as follows. In section II we present an overview of relevant state of the art. In section III, we present the model, and in sections IV and V we present the formulations of the online learning problem as stochastic and adversarial bandit respectively. Section VI presents numerical results, and section VII concludes the paper.

II. Related work

Controller placement has attracted a lot of attention in the last decade due to the growing importance of SDN for wireline networks. A comprehensive survey is [6], where the works are categorized according to the sought objective of controller placement, such as control latency, reliability, network resilience against switch, link or controller failures, load balancing across controllers, and cost efficiency. We will discuss RTT delay, as this is the focus of our work.

Static controller placement. A vast body of literature is focused on static controller placement approaches, where the decision is taken either by taking a snapshot of the system or by studying the objective in an average sense. The seminal work under a latency objective is [9], which asks how many controllers to deploy in a network, and where to place them. It defines different objectives related to latency, such as: (*i*) minimizing the average C-S latency over all switches (assuming that each switch will connect to its associated controller through a shortest path), which is the k-median problem; (*ii*) minimizing the maximum C-S latency over all switches, which is the k-center problem; (*iii*) minimizing the number of switches with delays to the controller that are bigger than a value. The paper concludes that for medium-sized networks, a single controller is enough to achieve latency values that are useful to certain application scenarios. In [10], processing capacity constraints at the controllers are considered, and switch association to a controller is performed so as to balance processing load across controllers. The paper formulates the problem of finding the controller placement and switch association so as to minimize the maximum delay to the associated controller, subject to a maximum total load for each controller.

The work [11] studies the placement of the minimum number of controllers and switch assignment to controllers so as to achieve certain delay bounds in controller response time. The latter involves service time at controllers, which in turn depends on the workload assigned to a controller, assuming a First-Come-First-Serve packet service policy. The work [12] studies the controller placement and workload distribution problem to optimize a metric that is a tradeoff between controller response time and utilization. A heuristic methodology based on simulated annealing is proposed in [13] to achieve controller placements that are Pareto optimal with respect to different performance metrics, together with a practical user interface.

Clustering approaches. Another thread involves clustering techniques [14]-[16], where a cluster is a subset of switches assigned to the same controller. Different similarity metrics among nodes are used, such as the negative Euclidean distance and the shortest-path distance between node pairs. The work [16] proposes a network partition policy into sub-networks and controller placement in each subnetwork, with the aim to reduce the maximum end-to-end latency between controllers and their associated switches. Besides the clustering approach, the authors explicitly model the queuing latency of requests at the controllers through an $M/M/m$ queue, where m is the number of controllers that collaborate to serve switch requests.

Message overhead cost. Another line of works model the controller-to-controller (C-C) message overhead. The work [17] considers the problem of minimizing the C-C and C-S delay costs through a facility-location-based heuristic and a bargaining game-based approach. The work [18] formulates the problem of controller placement so as to minimize a weighted sum of delay and message overhead cost for C-C and C-S communication, where the overhead is the rate of exchange of control messages. An important contribution of this paper is that its models are inspired by real measurements of overhead costs on the OpenDaylight and ONOS SDN platforms. In ONOS, a leaderless approach is followed, whereby controllers exchange overhead messages among themselves, and the rate of such messages is linear with controller load. On the other hand, OpenDaylight uses a leader-based approach whereby controllers synchronize only with the master (leader) controller. These differences result in different type of objectives that are addressed respectively through a heuristic using the theory of submodular function optimization, and through a facility-location inspired algorithm.

Adaptive methods. A line of works pertains to adaptive methods that adjust the number of controllers, their placement, and the switches assigned to controllers in an adaptive fashion. These works are mostly focused on solving the switch migration problem from one controller to another. The work [19] formulates this problem as a Network Utility Maximization problem, where the objective is to maximize total utility of controllers, which is a concave function of their assigned load. The work [20] formulates the problem as a stable matching one, where the preference of a switch for a controller depends on response time, while the preference of a controller for a switch depends on message overhead among them. Since frequent reassignment of switches to controllers may be cumbersome to implement, some works propose "soft" association, whereby a switch can be associated with more than one controllers.

While almost all adaptive methods focus on the solution of an optimization problem periodically and in a myopic manner, the work [21] takes a different approach. It formulates the problem of controller placement and switch association as an online optimization problem, based on instantaneous information about request arrivals at each time slot. The paper uses as additional controls the adjustment of controller computational speed and the routing of some requests to other controllers. The objective is to minimize the time-average energy cost due to processing and due to request routing to other controllers, subject to maintaining a stable queue of backlogged requests at each controller. The authors use Lyapunov optimization methods to solve this problem. A Lyapunov optimization method is also used in [22] for a similar cost minimization problem, where the idea is to allow control devolution, i.e. the selective transfer of a portion of the request load from controllers to switches.

Machine Learning approaches. Although a number of Machine Learning (ML) approaches have been applied in the data-plane of SDN [23]-[25], ML-based approaches for controller placement are scarce. An approach based on deep reinforcement learning is proposed in [26] for placement of a given number of controllers. The state of each switch is modeled as the set of flows that are addressed to each controller, the action is the tentative assignment of each switch to a controller, and the reward is a weighted sum of average latency and load variance at each controller. The work [27] addresses the joint control and learning problem for control devolution and switch-controller association for a given set of controllers, subject to maintaining request queue stability. For the control problem, Lyapunov optimization is used, while the learning problem is modeled as a combinatorial multi-armed bandit, where an arm corresponds to a switch-controller pair.

To the best of our knowledge, *our work is the first one to apply the multi-armed bandit online learning approach* to controller placement.

III. Model

We consider a wireless network with N nodes. We assume that the topology size is such that a single controller suffices to coordinate the network. Time is slotted, and at each time t, a single node needs to be selected as controller, and the rest of the nodes act as switches.

At each time slot t, let the decision variable $x_i(t) = 1$ if node i is selected as the controller, and $x_i(t) = 0$, if it is not. The sequence of events is as follows. At the beginning of each time slot t, a node i is selected as the controller, and all other nodes are associated with it. Next, requests from the controller and from the switches arise at the beginning of the time slot. Requests from the controller to switches may be for topology discovery, switch monitoring, or network state and statistics collection. Requests from the switches to the controller may be about flow forwarding rule setup. Without loss of generality, we assume that at each slot, one request is generated from the controller to each switch, and one request is generated from each switch to the controller. Requests are satisfied and the RTT delay is measured within time slot t.

Let $W_{i,j}(t)$ denote the round-trip-time (RTT) delay between nodes i and j at time t, when i is the request initiator and j is the recipient. Delay is the superposition of factors such as C-S propagation delay which depends on the number of hops between them, intermediate link transmission delays that depend on wireless transmission rates, and queuing delay at the node that serves the request. We do not isolate these factors but consider their cumulative effect as a single number. For requests initiated by the controller, RTT delay is measured at the controller upon reception of request response. For requests initiated by the switch, RTT is measured at the switch, and its value is passed to the controller at the end of the time slot. The slot duration is assumed to be long enough, e.g. one order of magnitude larger that typical RTT values. The total measured RTT delay at controller node i for controller- and switch-initiated requests is:

$$W_i(t) = \sum_{j\neq i} W_{i,j}(t) + \sum_{j\neq i} W_{j,i}(t)\,, \tag{1}$$

At the next slot, the controller selection decision is triggered again, and the procedure above repeats itself.

If each RTT delay process $W_{i,j}(t)$ is stationary with unknown distribution and unknown mean $\mathbb{E}[W_{i,j}(t)] = \overline{W}_{i,j}$, then process $W_i(t)$, $i = 1, \dots, N$ is stationary with unknown mean $\mathbb{E}[W_i(t)] = \overline{W}_i = \sum_{j\neq i} \overline{W}_{i,j} + \sum_{j\neq i} \overline{W}_{j,i}$.

Consider a time horizon T. A controller placement policy is $\mathbf{x} = (\mathbf{x}(1), \dots, \mathbf{x}(T))$, where $\mathbf{x}(t) = (x_1(t), \dots, x_N(t))$ is a binary vector whose i-th component $x_i(t) = 1$ if node i is selected as controller at time t, and 0 otherwise. It is $\sum_{i=1}^N x_i(t) = 1$ for each t, since one node is the controller.

We are interested in finding the controller selection policy that minimizes the total average delay across all requests arising in the time horizon, i.e we would like to solve:

$$\min_{\mathbf{x}} \frac{1}{T} \sum_{t=1}^{T} \sum_{i=1}^{N} \mathbb{E}[W_i(t)]\, x_i(t) \tag{2}$$

subject to $\sum_{i=1}^N x_i(t) = 1$, for all t, where the expectation in $\mathbb{E}[W_i(t)]$ is with respect to RTT randomness.

Remark: If we relax the assumption of one request from the controller and each switch at each slot, we need to account for the different number of requests. In this case, there will exist two sources of randomness, i.e. the number of requests and the RTT delays. Let $\lambda_i(t)$, $\lambda'_j(t)$ be the number of requests initiated by the controller (say, node i) and by switch node j at slot t. We assume that all requests arise at the beginning of the slot. Let $W^s_{i,j}(t)$ be the measured RTT delay for request $s = 1, \dots, \lambda_i(t)$ initiated by the controller. Let $W^s_{j,i}(t)$ be the RTT delay for request $s = 1, \dots, \lambda'_j(t)$ initiated by switch j. The total measured RTT delay is:

$$W_i(t) = \sum_{j\neq i} \left(\sum_{s=1}^{\lambda_i(t)} W^s_{i,j}(t) + \sum_{s=1}^{\lambda'_j(t)} W^s_{j,i}(t) \right). \tag{3}$$

If processes $\lambda_i(t)$ and $\lambda'_i(t)$ for each i are stationary with mean $\overline{\lambda}_i$ and $\overline{\lambda}'_i$, and if the random processes of the number of requests and RTT delays are independent, then

$$\mathbb{E}[W_i(t)] = \overline{\lambda}_i \sum_{j\neq i} \overline{W}_{i,j} + \sum_{j\neq i} \overline{\lambda}'_j \overline{W}_{j,i}\,. \tag{4}$$

In the sequel, we stick to the RTT delay expressions as in (1).

IV. Controller placement as a stochastic Multi-armed Bandit problem

We now study the online learning version of the problem. For the RTT delay processes $\{W_{i,j}(t)\}$, we consider two alternatives. In this section, we assume that the processes are stationary with unknown means, and we model the problem as a stochastic multi-armed bandit (MAB). Next, in section V, the RTT delay processes are assumed to be non-stationary, and the problem is modeled as an adversarial MAB.

First, we assume that the RTT delay processes $W_{i,j}(t)$ are stationary, each with unknown mean. This assumption is suitable for scenarios with static or low-mobility nodes. In this case, the processes $W_i(t)$ of RTT delays when node $i = 1, \dots, N$ is selected as controller are also stationary, but with unknown mean, $\overline{W}_i$. If we knew $\overline{W}_i$, we would always select as controller the node $i^* = \arg\min_{i=1,\dots,N} \overline{W}_i$.

We are interested in learning the unknown means over time with an appropriate controller selection strategy $\mathbf{x} = (\mathbf{x}(1), \dots, \mathbf{x}(T))$. The goal is that in the long run, the average RTT delay incurred by our strategy asymptotically approaches the RTT delay achieved by the optimal static policy in which the best controller i^* is selected for the entire horizon. This goal is captured by minimizing the time-average expected *regret* associated with policy $\mathbf{x}$ over horizon T,

$$\overline{R}_{\mathbf{x}}(T) = \frac{1}{T} R_{\mathbf{x}}(T) = \frac{1}{T}\left[\mathbb{E} \sum_{t=0}^{T-1} \sum_{i=1}^{N} x_i(t) W_i(t) - T\,\overline{W}_{i^*} \right], \tag{5}$$

where $R_{\mathbf{x}}(T)$ is the total expected regret, and the expectation is taken with respect to the randomness of RTT delay processes. Our policy $\mathbf{x}$ should be such that $\overline{R}_{\mathbf{x}}(T) \to 0$ as $T \to \infty$, or equivalently $R_{\mathbf{x}}(T)$ is a sub-linear function of T.

A. Stochastic multi-armed bandit formulation

The stochastic multi-armed bandit (MAB) is a way to model sequential decision-making problems under uncertainty [28]-[30]. At each time over a finite time horizon, a player selects an arm (choice) out of a number of alternatives. Each arm is associated with a reward with unknown probability distribution, with unknown mean and variance. After the choice, the player receives a reward that is a sample from the corresponding probability distribution. The goal of the player is to to find out through an online learning policy which distribution (arm) has the highest mean reward, and at the same time maximize its total reward while playing over the time horizon. A bandit algorithm specifies a way for selecting an arm at each time. The idea of bandit algorithms is to balance exploitation with exploration so as to exploit the arm that currently appears to offer the best reward, while exploring other arms that have not been played much up to that time.

In the controller placement problem, the player is the decision maker that picks a node as a controller at each time slot. An arm corresponds to one of the N possible choices for a node to become controller, while the random reward obtained after making a choice corresponds to the negative RTT delay, $-W_i(t)$ that is measured when i is selected as controller. Each choice of node i as controller is characterized by an unknown probability distribution of RTT delay, which is in general different for different choices but invariant with time. This makes sense because of the assumed static network setting that allows statistical characterization of randomness.

B. Stochastic bandit algorithms for controller placement

The stochastic MAB problem has been studied in the seminal paper [31] where the Upper Confidence Bound (UCB) algorithm and some variants of it were proposed, together with results on regret upper bounds. The work [32] performed experimental comparison of various bandit algorithms with respect to the following performance indicators: total regret accumulated over a certain horizon, regret as a function of time, and percentage of time when the optimal arm is selected. The paper argues that the only parameters that affect the performance of bandit algorithms are the number of arms and the variance of arm rewards. In the sequel, we present three bandit algorithms that stand out, based on empirical evidence from the experiments in [31], [32]. For each algorithm, we discuss results on regret guarantees (if any), and we propose guidelines for its practical implementation.

1) UCB algorithm: In the Upper Confidence Bound (UCB) algorithm, the idea is to maintain at each time t a reward estimate for each arm i, which consists of the empirical average reward $\widehat{\mu}_i(t)$ up to time t, plus a term that accounts for the additional *potential* reward that might be achievable by the arm due to the fact that it has not been selected many times. The sum of these terms forms an upper confidence bound for achievable reward. The reward estimate for arm i at time t is

$$\mathrm{UCB}_i(t) = \widehat{\mu}_i(t) + \sqrt{\frac{2\ln t}{n_i(t)}}\,, \tag{6}$$

where $n_i(t)$ is the number of times that arm i is played up to time t. The algorithm starts by first picking each arm once. Then, at each time t, it chooses arm $i^*(t) = \arg\max_i \mathrm{UCB}_i(t)$. The UCB algorithm achieves a bounded total regret R_T,

$$R_T \le 8\log T \sum_{i:\mu_i<\mu^*} \frac{1}{\Delta_i} + \left(1+\frac{\pi^2}{3}\right)\sum_{i=1}^{N}\Delta_i\,, \tag{7}$$

where $\Delta_i = \mu^* - \mu_i$, where μ_i is the real average reward of arm i, and $\mu^* = \max_i \mu_i$. This bound depends on factors Δ_i which depend on the arm reward distributions. A distribution-independent bound for the regret is [33],

$$R_T \le 5\sqrt{NT\log T} + 8N\,. \tag{8}$$

In the controller placement problem, the UCB algorithm needs to be modified as follows: instead of the upper confidence bound, at each time t and for each arm i, the *Lower Confidence Bound* is computed,

$$\mathrm{LCB}_i(t) = \max\{\widehat{w}_i(t) - \sqrt{\frac{2\ln t}{n_i(t)}}, 0\}\,, \tag{9}$$

where the empirical mean RTT delay is

$$\widehat{w}_i(t) = \frac{\sum_{\tau=1}^{t} W_i(\tau)x_i(\tau)}{n_i(t)}\,, \tag{10}$$

and the operator $\max\{\cdot, 0\}$ ensures non-negativity. Factor $\mathrm{LCB}_i(t)$ thus presents an optimistic estimate for the RTT delay of arm i at time t. The algorithm starts with first selecting each node as a controller once. Then, at each time t, it chooses as controller the node $i^*(t) = \arg\min_i \mathrm{LCB}_i(t)$.

2) UCB-tuned algorithm: UCB-tuned is a variant of UCB that was proposed in [31]. The idea is to include in the UCB factor the empirical variance of the reward of each arm so as to favor selection of arms with high variance of rewards and therefore spot suboptimal arms faster. In the controller placement problem, the algorithm becomes *LCB-tuned*. At each time t, it computes for each arm i the factor

$$\text{LCB-tuned}_i(t) = \max\{0, \widehat{w}_i(t) - \sqrt{\frac{2\ln t}{n_i(t)}\cdot\min\{1/4, \widehat{V}_i(t)\}}\} \tag{11}$$

where

$$\widehat{V}_i(t) = \widehat{\sigma}_i^2(t) + \sqrt{\frac{2\ln t}{n_i(t)}} \tag{12}$$

is an optimistic estimate of the variance of RTT delays of node i as controller at time t, $\widehat{\sigma}_i$ is the empirical variance of RTT delay of node i whenever it is selected as controller, and $1/4$ is the maximum value of variance of a Bernoulli random variable over different success probabilities, which is inserted to regulate the impact of arm variance. LCB-tuned chooses at each time t as controller the node $i^*(t) = \arg\min_i \text{LCB-tuned}_i(t)$

The algorithm starts by selecting each node as controller once. Then, at each time t, it chooses as controller the node with the lowest value LCB-tuned(t). There does not exist a regret performance guarantee for UCB-tuned (and thus for LCB-tuned). However, the work [34] proves logarithmic regret bounds for a family of algorithms that use functions of empirical variance in the UCB factors.

3) Boltzmann exploration: Boltzmann exploration belongs to the class of methods in which, at each time an arm is chosen with probability that is proportional to its empirical mean reward up to that time so that arms with higher empirical mean rewards are selected with higher probability. The exponential weighting of rewards applied by the softmax function has proved to be effective in weighting the relative importance of arms and their selection probabilities [35]. At each time t, arm i is chosen with probability

$$p_i(t) = \frac{\exp\left(\eta \widehat{\mu}_i(t)\right)}{\sum_{j=1}^{N} \exp\left(\eta \widehat{\mu}_j(t)\right)}, \tag{13}$$

where $\widehat{\mu}_i(t)$ is the empirical mean reward of arm i up to time t, and η is a learning rate that calibrates the randomness of choice. Smaller values of η cause actions to be selected with almost equal probability, while as η increases, there are larger differences in selection probability for actions that differ in their empirical rewards. For $\eta \to \infty$, the arm with the highest empirical average reward is selected with probability one. For $t = 0$, $p_i(0) = 1/N$ for all i.

In the controller placement problem, for each node i selected as controller, let $\widehat{w}_i(t)$ be the empirical mean RTT delay up to time t in (10). Boltzmann exploration intuitively chooses a node as controller with higher probability if its empirical mean RTT delay is smaller. Thus, at each time t, a node i is selected to be the next controller with probability

$$p_i(t) = \frac{\exp\left(-\eta \widehat{w}_i(t)\right)}{\sum_{j=1}^{N} \exp\left(-\eta \widehat{w}_j(t)\right)}. \tag{14}$$

Although Boltzmann exploration is shown in practice to be very effective in exploration, it comes with no regret performance guarantees. In [36], the authors propose a way to vary the learning rate so as to achieve logarithmic regret, albeit under the unrealistic assumption of knowing the suboptimality gap, i.e. the smallest difference between the mean reward of the optimal arm and the mean reward of any other arm. The paper proposes a way to modify Boltzmann exploration to achieve logarithmic regret through the Gumbel distribution. The idea is to view the probabilistic selection rule $p_i(t) \propto \exp\left(\eta \widehat{\mu}_i(t)\right)$ equivalently as the deterministic rule, $I_t = \arg\max_i\{\widehat{\mu}_i(t) + \theta_i(t) Z_i(t)\}$, where $\theta_i(t) = \sqrt{C^2/n_i(t)}$, C is a constant, and $Z_i(t)$ is a random sample from the Gumbel probability distribution, drawn independently for each arm i. This method involves several parameters that require tuning, hence in the sequel we adopt the simple Boltzmann exploration rule (14).

C. Implementation in an SDN environment

1) LCB algorithm: The LCB algorithm can be implemented in an SDN environment as follows. At the beginning of time slot t, assume node $i(t) \in \{1, \ldots, N\}$ is selected as controller. This node has just received from the previously selected controller $i(t-1)$ the vector of current empirical mean RTTs up to time $t-1$, $\widehat{\mathbf{w}}(t-1) = (\hat{w}_1(t-1), \ldots, \hat{w}_N(t-1))$ and the vector of number of times that each node has been selected as controller, $\mathbf{n}(t-1) = (n_1(t), \ldots, n_N(t-1))$. At time t, the current controller $i(t)$ (say i) performs the following steps:

- **STEP 1**: It measures total incurred RTT delay $W_i(t)$.
- **STEP 2**: It updates its own empirical mean RTT delay, $\widehat{w}_i(t)$. The vector of RTT empirical estimates up to time t, $\widehat{\mathbf{w}}(t)$ is then formed, whose i-th component is the computed $\widehat{w}_i(t)$, while all other components remain the same, i.e. $\widehat{w}_j(t) = \widehat{w}_j(t-1)$ for $j \neq i$.
- **STEP 3**: Vector $\mathbf{n}(t)$ is formed, where $n_i(t) = n_i(t-1) + 1$, and $n_j(t) = n_j(t-1)$ for $j \neq i$.
- **STEP 4**: LCB indices $\text{LCB}_1(t), \ldots, \text{LCB}_N(t)$ of all nodes are updated.
- **STEP 5**: The current controller $i(t)$ finds the node $i^*(t) = \arg\min \text{LCB}_i(t)$ with smallest LCB index. If $i^*(t) \neq i(t)$, then $i(t)$ sends a notification to $i^*(t)$ so that it becomes the controller at the next time slot, i.e. $i(t+1) = i^*(t)$. It also passes vectors $\widehat{\mathbf{w}}(t)$ and $\mathbf{n}(t)$ to the next controller, $i(t+1)$ to take over.

2) LCB-tuned algorithm: A similar process as the one above is followed for the LCB-tuned algorithm. The difference is that in Step 2, the empirical variance of RTT delays needs to be updated, and this empirical variance needs to be communicated to the next controller in Step 5.

3) Boltzmann exploration: At the beginning of time slot t, the current controller $i(t)$ has just received from the previous controller the vector $\widehat{\mathbf{w}}(t-1)$ and has become active controller. The steps are as follows.

- **STEP 1** Node i measures total RTT delay $W_i(t)$ and updates its own empirical RTT delay $\widehat{w}_i(t)$.
- **STEP 2**: The vector of RTT empirical means up to time t, $\widehat{\mathbf{w}}(t)$ is formed, where only the i-th component changes.
- **STEP 3** Probabilities $\{p_i(t)\}_{i=1,\ldots,N}$ are found by (14).
- **STEP 4**: Node $i(t)$ generates a random sample from an N-dimensional categorical distribution that takes as input these probabilities. The sample takes values in $\{1, \ldots, N\}$ and specifies the next node $i(t+1)$ to become controller at time $t+1$. If $i(t+1) \neq i(t)$, $i(t)$ sends to $i(t+1)$ vector $\widehat{\mathbf{w}}(t)$, and the process continues.

V. Controller placement as an adversarial Multi-armed bandit problem

In section IV, we assumed that the incurred RTT delay if a node is selected as controller can be characterized through a probability distribution, and the random processes of RTT delays for each controller selection are stationary. Here, we relax the assumption of existence of stationary distributions of RTT delays. We assume that for each node selected as

Algorithm 1 EXP3 Algorithm for controller selection among N nodes.

```
1:  input: Parameter $\beta > 0$, the learning rate.
2:  output: A sequence of vectors $\mathbf{x}(1), \dots, \mathbf{x}(T)$.
3:  Initialization: $\mathbf{y}(1) = \mathbf{1}$, $\mathbf{x}(1) = \frac{\mathbf{y}(1)}{\|\mathbf{y}(1)\|_1} = \frac{1}{N}\mathbf{1}$.
4:  for $t = 1, \dots, T$
5:      Variant (i): Choose node $a(t) = i$ as the controller with prob. $x_i(t+1)$.
6:      Variant (ii): Choose node $a(t) = i$ as the controller with prob. $x'_i(t+1) = (1-\gamma)x_i(t+1) + \gamma\frac{1}{N}$ $(0 < \gamma < 1)$.
7:      Observe RTT delay $W_{a(t)}(t)$, and estimate the gradient of the cost function with (16).
8:      if $a(t) = i$ then $y_i(t+1) = y_i(t)\exp\left(-\beta\frac{W_i(t)}{x_i(t)}\right)$.
9:      else if $a(t) \neq i$, then $y_i(t+1) = y_i(t)$.
10:     Project $\mathbf{x}(t+1) = \frac{\mathbf{y}(t+1)}{\|\mathbf{y}(t+1)\|}$.
11: end for
```

controller, the RTT delay is a non-stationary process that cannot be characterized statistically through a probability distribution. This scenario captures the worst-case scenario that RTT delays are shaped in an arbitrary manner through a malicious entity that seeks to disrupt learning. The scenario models network settings with arbitrary node mobility, topology changes or traffic dynamics that change RTT delay in an unpredictable manner from one slot to the other. We assume RTT delays take values in a bounded interval.

A. Adversarial multi-armed bandit formulation

In the presence of arbitrarily varying RTT delays, the controller placement problem can be modeled as an adversarial MAB [37]. While in stationary stochastic settings deterministic online policies suffice to achieve sublinear regret, this is not the case in adversarial settings, where only randomized algorithms are effective for combating the hypothetical adversary that may cause arbitrary changes in RTT delays.

The setting remains the same. Each of the N arms stands for a node selection as a controller. At each time t, the learner needs to select a controller $a(t)$. Let Pr$[a(t) = i] = x_i(t)$ be the probability of choosing node i as controller at time t. We define a linear cost function $f(t) = \sum_{i=1}^N W_i(t)x_i(t)$, where $W_i(t)$ is the RTT delay if node i is selected as the controller at time t. Once the learner picks a controller $a(t)$ at time t, it gets to see *only* the cost $W_{a(t)}(t)$ associated with that node, and it cannot observe the entire vector $\mathbf{W}(t) = (W_1(t), \dots, W_N(t))$, as *only* the RTT delay for the selected controller is measured.

Given a time horizon T, the goal of the learner is to find a controller selection policy $\mathbf{x} = (\mathbf{x}(1), \dots, \mathbf{x}(T))$ that minimizes time-average total regret,

$$\overline{R}_{\mathbf{x}}(T) = \frac{1}{T}R_{\mathbf{x}}(T) = \frac{1}{T}\left[\mathbb{E}[\sum_{t=1}^T \mathbf{W}^T(t)\mathbf{x}(t)] - \min_i \sum_{t=1}^T W_i(t)\right], \tag{15}$$

where expectation is with respect to the randomness of the distribution of selecting a controller at time t.

B. The EXP3 algorithm for controller placement

First, note that $\nabla_{\mathbf{x}(t)} f(t) = \mathbf{W}(t)$. If the learner had access to the entire gradient vector (thus, vector $\mathbf{W}(t)$) after decision, an efficient learning algorithm would be Exponentiated Gradient (EG) [38, p.80]. At each time t, EG would choose node i as controller with probability $x_i(t) = \frac{z_i(t)}{\sum_{j=1}^N z_j(t)}$ where $z_i(t) = \exp\left(-\beta\sum_{\tau=1}^t W_i(\tau)\right)$ i.e. the probability of each choice would depend on its accumulated RTT delay up to time t. However, the learner receives *limited* feedback, namely it measures only cost $W_{a(t)}(t)$ for the selected controller $a(t)$ at time t, i.e. only the $a(t)$-th component of $\nabla_{\mathbf{x}(t)} f(t)$. Thus, the whole gradient vector is not available, and an estimate is needed. We define the random vector $\widehat{\mathbf{W}}(t)$, with

$$\widehat{W}_j(t) = \begin{cases} W_j(t)/x_j(t), & \text{if } j = a(t), \\ 0, & \text{else }, \end{cases} \tag{16}$$

which is an unbiased estimate of the gradient (and the RTT delay) at each time t, since $\mathbb{E}[\widehat{W}_i(t)\,|\,\mathbf{x}(t)] = x_i(t)\cdot\frac{W_i(t)}{x_i(t)} + (1 - x_i(t))\cdot 0 = W_i(t)$. In the presence of limited feedback, an algorithm that solves the adversarial bandit problem is the Exponential-weight algorithm for Exploitation and Exploration (EXP3) [39], which is based on EG, but with the gradient estimate plugged into the vector update of EG.

At each slot t, the EXP3 algorithm outputs a probability vector $\mathbf{x}(t)$ that specifies the probabilities with which each node will be selected as controller at the next time slot. At initialization, these probabilities are all set to $1/N$. At each slot, EXP3 constructs the unbiased estimate of the gradient of the cost function based on the observed RTT delay, and it uses this to update the probabilities of selecting a node as controller in the next slot. Intuitively, a node for which large RTT delay is observed up to slot t has fewer chances of being selected as controller in the next slot $t + 1$.

The EXP3 algorithm steps are shown under Algorithm 1. There are two variants: (*i*) a node is selected as controller based on distribution $\mathbf{x}(t)$; (*ii*) controller selection occurs through a mix of distribution $\mathbf{x}(t)$ and the uniform distribution over all nodes, with a mixing factor γ. With mixing, we aim to increase the randomness of the selection by doing more exploration than exploitation, and thus we try all N choices and estimate RTT delays for them. Otherwise, we might miss a good node because of initially large RTT delays for this node, which would lead us to not select that node, while the possibly smaller RTT delays that could occur later would not be observed. For $\beta = \sqrt{\log N/TN}$, EXP3 achieves a regret upper bound of $O(\sqrt{2TN\log N})$ [38, p.104].

C. Implementation

EXP3 runs at each slot t at node $a(t)$ that is selected as the controller at t. Node $a(t)$ receives from the previously selected controller $a(t-1)$ the vector of weights $\mathbf{y}(t-1)$. Then, node $a(t)$ observes RTT delay $W_{a(t)}(t)$ and updates the weight vector $\mathbf{y}(t)$ based on Steps 7,8 of Algorithm 1. It then computes the probability vector $\mathbf{x}(t)$ or $\mathbf{x}'(t)$, depending on the variant, and it chooses the next controller according to

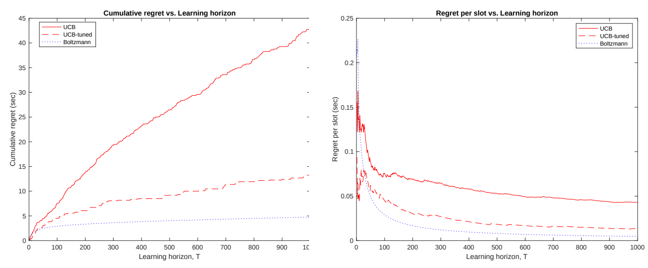


Figure 1. Regret comparison for the UCB, UCB-tuned and Boltzmann exploration algorithms, for $N = 5$ arms and $\sigma = 0.1$. Left: Cumulative regret; right: Regret per slot.

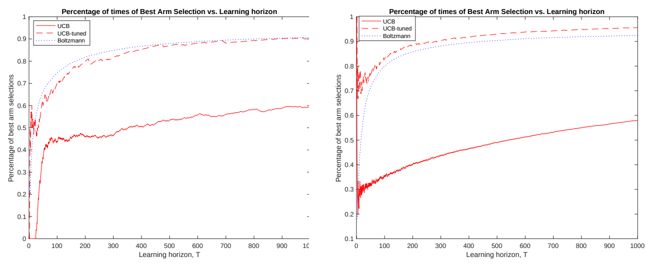


Figure 2. Percentage of times of best arm selection for the UCB, UCB-tuned and Boltzmann exploration algorithms for $N = 5$ arms and for different values of σ of the RTT normal distribution. Left: $\sigma = 0.1$; right: $\sigma = 0.01$.

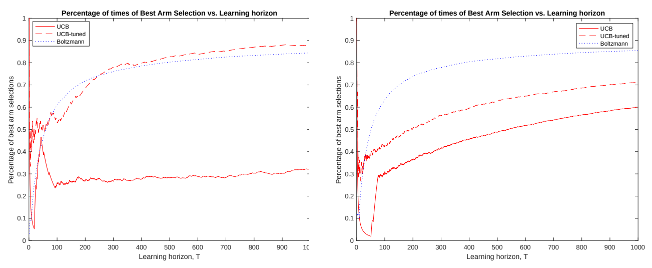


Figure 3. Percentage of times of best arm selection for the UCB, UCB-tuned and Boltzmann exploration algorithms for $N = 10$ arms and for different values of σ of the RTT normal distribution. Left: $\sigma = 0.1$; right: $\sigma = 0.01$.

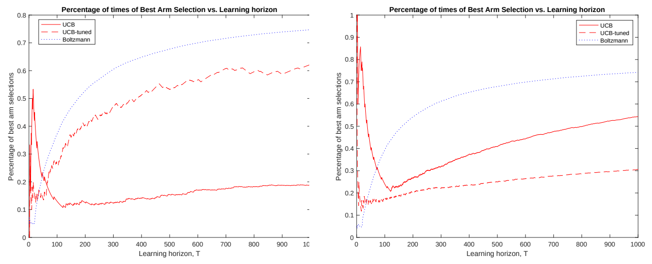


Figure 4. Percentage of times of best arm selection for the UCB, UCB-tuned and Boltzmann exploration algorithms for $N = 20$ arms and for different values of σ of the RTT normal distribution. Left: $\sigma = 0.1$; right: $\sigma = 0.01$.

the output of a random sample from a categorical distribution, If $a(t+1) \neq a(t)$, node $a(t)$ sends a message to $a(t+1)$ and notifies it to take over. It will also pass to $a(t+1)$ the current weight vector $\mathbf{y}(t)$, and the process continues at $a(t+1)$.

VI. Numerical Results

We simulate different bandit algorithms and compare their performance. We consider a network of N nodes, each of which may become a controller and is an arm in the bandit. We compare bandit algorithms with respect to the following performance metrics to capture different angles of learning:

- Regret per time slot (or time-average regret) over time horizon T, $\overline{R}(T)$. This is given by (5) and (15) respectively for the stochastic and the adversarial bandit.
- Cumulative regret over horizon T, $R(T)$, given by (5) and (15) for the stochastic and adversarial bandit.
- Percentage of slots in which the best arm is selected over time horizon T, P_T. If K_T is the number of slots over horizon T in which the best arm is selected, then $P_T = K_T/T$. The best arm for the stochastic bandit is $i^* = \arg\min_i \mu_i$, and for the adversarial bandit it is $i^* = \arg\min_i \sum_{t=1}^{T} W_i(t)$.
- Time when the percentage of best arm selection reaches a certain value $q\%$, denoted as $t_q = \min\{t : P_t = q\}$.

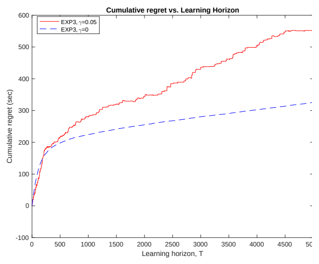
Results are averaged over 100 experiments. Algorithms LCB and LCB-tuned are referred to as UCB and UCB-tuned.

A. Stochastic bandit numerical experiments

For each experiment in the stochastic bandit case, we assume that for arm $j = 1, \ldots, N$, the expected delay is $\mu_j = 0.3 + 0.7 d_j$ where d_j is uniformly distributed in $[0, 1]$. The delays of different arms at different slots are generated through a normal distribution $N(\mu_j, \sigma^2)$, where σ takes different values. The learning parameter in Boltzmann exploration is $\eta = 100$.

In Figure 1, we depict the performance of UCB, UCB-tuned and Boltzmann exploration algorithms for $N = 5$ arms and $\sigma = 0.1$. We show the cumulative regret and the regret per slot. Boltzmann and UCB-tuned achieve the best performance, and their superiority over UCB is clear. In Figure 2, we depict the performance of the three algorithms in terms of percentage of times when the best arm is chosen, for the cases of high and low RTT delay variability ($\sigma = 0.1$ and $\sigma = 0.01$, respectively). For $\sigma = 0.1$, Boltzmann and UCB-tuned select the best arm at 91% and 90% of times, while UCB's corresponding percentage is around 60%. An even better performance is observed for Boltzmann and UCB-tuned for $\sigma = 0.01$, where these two methods select the best arm at 93% and 95% of times in the long run, while UCB's percentage is below 60%.

In Figures 3 and 4, the percentages of best arm selection for the three methods are shown for $N = 10$ and $N = 20$ nodes (arms). For $N = 10$ arms and $\sigma = 0.1$, UCB-tuned outperforms Boltzmann, with 88% over 84%, while UCB achieves only 32%. For $\sigma = 0.01$, RTT delays are more distinguishable and thus Boltzmann goes up to 85% and UCB-tuned goes to 71%, while UCB reaches 60%. Similar trends are observed for $N = 20$ arms, but with lower percentages for all algorithms. For $\sigma = 0.01$, UCB outperforms UCB-tuned with



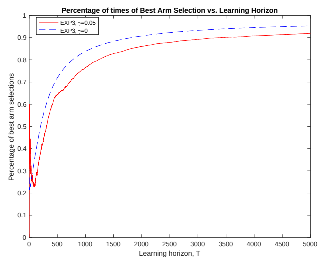
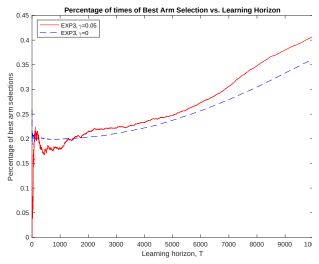


Figure 5. Performance of the EXP3 algorithm with mixing parameter $\gamma = 0$ and $\gamma = 0.05$, for $N = 5$ arms and for the delay process of Case A. Left: Cumulative regret; right: Percentage of times of best arm selection.



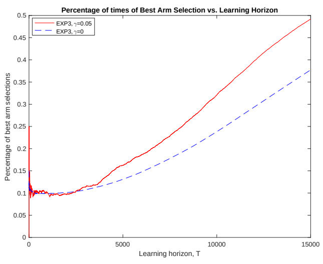


Figure 6. Performance of the EXP3 algorithm with mixing parameter $\gamma = 0$ and $\gamma = 0.05$, for the delay process of Case B. Left: $N = 5$ arms; right: $N = 10$ arms.

Table I
TIMES t_q (IN NUMBER OF SLOTS) WHEN THE PERCENTAGE OF BEST ARM SELECTION REACHES VALUE $q\%$.

Experiment	**Boltzmann**	**UCB-tuned**	**UCB**
$N = 5$, $\sigma = 0.1$	$t_{70} = 50$ $t_{80} = 180$ $t_{90} = 600$	$t_{70} = 110$ $t_{80} = 210$ $t_{90} = 680$	- - -
$N = 5$, $\sigma = 0.01$	$t_{70} = 50$ $t_{80} = 100$ $t_{90} = 420$	$t_{70} = 30$ $t_{80} = 80$ $t_{90} = 250$	- - -
$N = 10$, $\sigma = 0.1$	$t_{70} = 170$ $t_{80} = 380$	$t_{70} = 200$ $t_{80} = 420$	- -
$N = 10$, $\sigma = 0.01$	$t_{60} = 80$ $t_{70} = 140$ $t_{80} = 380$	$t_{60} = 400$ $t_{70} = 950$ -	$t_{60} = 1,000$ - -
$N = 20$, $\sigma = 0.1$	$t_{50} = 160$ $t_{60} = 270$ $t_{70} = 560$	$t_{50} = 360$ $t_{50} = 670$ -	- - -
$N = 20$, $\sigma = 0.01$	$t_{50} = 140$ $t_{60} = 260$ $t_{70} = 550$	- - -	$t_{50} = 770$ - -

55% vs. 30% percentage of times of best arm selection. Note that in all experiments with stochastic bandits, performance reaches a steady state at about 500 slots for $N = 5$ arms, and at around $1,000$ slots or later, for $N = 10$ and $N = 20$ arms. Finally, in Table I, we show results on the times t_q when the percentage of best arm selection reaches value $q\%$. Results verify the superiority of Boltzmann in terms of faster learning.

B. Adversarial bandit numerical experiments

For the adversarial bandit, we consider two alternatives for the nonstationary delay process:
(i) Case A: A process where delays at time t and arm j are given by

$$W_j(t) = r_j \cdot |\cos t| + 3j \cdot r'_j \cdot r''_j\,, \tag{17}$$

(ii) Case B: A delay process in which it is more difficult to distinguish between different arms, with

$$W_j(t) = \rho_j \cdot |\cos t| + \frac{t}{1000} \cdot \frac{j}{10} \cdot \rho'_j \cdot |\cos t|\,, \tag{18}$$

where r_j, r'_j, r''_j, ρ_j, ρ'_j are uniformly distributed in $(0, 1)$. The learning parameter of EXP3 is $\beta = \sqrt{\log N/TN}$.

In Figure 5, we show the performance of EXP3 for $N = 5$ arms, in terms of cumulative regret and percentage of best arm selections for the process of Case A. We consider mixing parameter $\gamma = 0$ and $\gamma = 0.05$. The former choice achieves a little better performance in terms of percentage of best arm selections (95% vs. 92%). We observe that the learning horizon is at least an order of magnitude larger than that in stochastic bandits, and steady-state is reached at about $2,500$ slots. In Figure 6, we provide results for the more difficult process of Case B, for $N = 5$ and $N = 10$ arms, where learning is still ongoing after $10,000$ and $15,000$ slots respectively. For example, after $10,000$ slots, EXP3 with $\gamma = 0$ achieves a percentage of 41% and 32% respectively.

VII. CONCLUSION

We viewed the SDN controller placement problem through the lens of online learning. We fit the stochastic and adversarial MAB models in the problem. Numerical experiments showed that algorithm performance depends strongly on certain parameters that characterize the problem instance, such as the number of arms and the variance of RTT delays. Further, Boltzmann exploration achieves the best performance in terms of different performance metrics. Our work is the first step towards understanding and optimizing online learning algorithms for controller placement. We have just scratched the surface, and there exist several open questions in the directions of model enhancement and of practical performance evaluation.

In this work, we left the request arrival process out of the formulation. However, this process is another source of uncertainty which may or may not be characterized statistically. Second, we assumed that RTT feedback is received within the time slot where the decision is made. We can enhance our model with delayed feedback, whereby the RTT value observation is available after some time slots. There exists evidence that an $O(\sqrt{T})$ regret is achievable, and the work [40] is a good starting point. Third, if we need to select more than one controllers, the problem becomes challenging due to the arising switch-controller association problem. The work [27] considered a MAB model when the controllers are fixed and the possible choices are the switch-controller pairs. When controllers need to be selected as well, the bandit model becomes nontrivial.

Finally, in the practical evaluation front, the selection of the appropriate algorithm and hyper-parameters can be made empirically only, since the associated theory does not exist. An interesting idea is to use emulators or SDN testbeds to obtain RTT delay measurements, and then choose the proper algorithm and configure its hyper-parameters based on such data.

Acknowledgment

This work has been carried out in the context of the project entitled Software defined MObile Tactical Ad hoc NETwork (SMOTANET), which has received funding from the European Defence Industrial Development Programme (EDIDP) under grant agreement No EDIDP-CSAMN-SDN-2019-038-SMOTANET. This paper reflects only the authors' views and the European Commission is not responsible for any use that may be made of the information contained herein.

References

[1] I. F. Akyildiz, S.-C. Lin, and P. Wang, "Wireless software-defined networks (W-SDNs) and network function virtualization (NFV) for 5G cellular systems: An overview and qualitative evaluation, *Comput. Netw.,* 93(12):66–79, 2015.

[2] A. Gudipati, D. Perry, L. Erran Li, and S. Katti, "SoftRAN: software defined radio access network", *Proc. of the second ACM SIGCOMM workshop on Hot topics in software defined networking (HotSDN),* 2013.

[3] ARL project. Dais ITA: The distributed analytics and information science international technology alliance, 2016–2021.

[4] K. Poularakis, Q. Qin, E. Nahum, M. Rio and L. Tassiulas, "Bringing SDN to the mobile edge," *Workshop on Distributed Analytics InfraStructure and Algorithms for Multi-Organization Federations, (in proc. of IEEE Smart World Congress),* 2017.

[5] K. Poularakis, G. Iosifidis and L. Tassiulas, "SDN-Enabled Tactical Ad Hoc Networks: Extending Programmable Control to the Edge," *IEEE Commun. Mag.*, vol. 56, no. 7, pp. 132-138, July 2018.

[6] T. Das, V. Sridharan and M. Gurusamy, "A Survey on Controller Placement in SDN," *IEEE Communications Surveys & Tutorials,* vol. 22, no. 1, pp. 472-503, First quarter 2020.

[7] A. Kumari and A.S. Sairam, "Controller placement problem in software-defined networking: A survey", *Networks journal Wiley*, vol.78, no.2, pp. 195– 223, Sept. 2021.

[8] Q. Qin, K. Poularakis, G. Iosifidis, S. Kompella and L. Tassiulas, "SDN Controller Placement With Delay-Overhead Balancing in Wireless Edge Networks," *IEEE Trans. on Network and Service Management,* vol. 15, no. 4, pp. 1446-1459, Dec. 2018.

[9] B. Heller, R. Sherwood, and Nick McKeown, "The controller placement problem", *Proc. 1st ACM workshop on Hot topics in software defined networks (HotSDN)* 2012.

[10] G. Yao, J. Bi, Y. Li, and L. Guo, "On the capacitated controller placement prob- lem in software defined networks", *IEEE Commun. Lett.*, 18(8):1339–1342, 2014.

[11] T. Y. Cheng, M. Wang and X. Jia, "QoS-Guaranteed Controller Placement in SDN," *Proc. IEEE GLOBECOM*, 2015.

[12] V. Huang, G. Chen, Q. Fu and E. Wen, "Optimizing Controller Placement for Software-Defined Networks," *Proc. IFIP/IEEE Symposium on Integrated Network and Service Management (IM),* 2019.

[13] S. Lange, S. Gebert, T. Zinner, P. Tran-Gia, D. Hock, M. Jarschel, and M. Hoffmann, "Heuristic Approaches to the Controller Placement Problem in Large Scale SDN Networks", *IEEE Trans. on Netw. and Serv. Management*, vol.12, no.1 March 2015.

[14] G. Wang, Y. Zhao, J. Huang, Q. Duan, and J. Li, "A k-means based network partition algorithm for controller placement in software defined network", *Proc. IEEE ICC*, 2016.

[15] J. Zhao, H. Qu, J. Zhao, Z. Luan, and Y. Guo, "Towards controller placement problem for software-defined network using affinity propagation", *Electron. Lett.,* 2017.

[16] G. Wang, Y. Zhao, J. Huang, and Y. Wu, "An effective approach to controller placement in software defined wide area networks", *IEEE Trans. Netw. Serv. Manage.,* vol15, no.1, pp. 344–355, 2018.

[17] A. Ksentini, M. Bagaa, T. Taleb, and I. Balasingham, "On using bargaining game for optimal placement of SDN controllers", *Proc. IEEE ICC,* 2016.

[18] Q. Qin, K. Poularakis, G. Iosifidis, S. Kompella and L. Tassiulas, "SDN Controller Placement With Delay-Overhead Balancing in Wireless Edge Networks", *IEEE Trans. on Network and Service Management,* vol. 15, no. 4, pp. 1446-1459, Dec. 2018.

[19] H. C. Cheng, Z. Wang and S. Chen, "DHA: Distributed decisions on the switch migration toward a scalable SDN control plane" *Proc. IFIP Networking Conference*, 2015.

[20] T. Wang, F. Liu, J. Guo, and H. Xu, "Dynamic SDN controller assignment in data center networks: Stable matching with transfers", *Proc. IEEE INFOCOM*, 2016.

[21] X. Lyu, C. Ren, W. Ni, H. Tian, R. P. Liu, and Y. J. Guo, "Multi-timescale decentralized online orchestration of software-defined networks", *IEEE J. Sel. Areas Commun.,* vol.36, no.12, pp.2716–2730, December 2018.

[22] X. Huang, S. Bian, Z. Shao, and Y. Yang, "Predictive switch-controller association and control devolution for SDN systems" *Proc. IEEE/ACM IWQoS,* 2019.

[23] S.-C. Lin, I. F. Akyildiz, P. Wang, and M. Luo, "QoS-aware adaptive routing in multi-layer hierarchical software defined networks: A reinforcement learning approach," *Proc. IEEE Int. Conf. Serv. Comput. (SCC),* 2016.

[24] P. Amaral, J. Dinis, P. Pinto, L. Bernardo, J. Tavares, and H. S. Mamede, "Machine learning in software defined networks: Data collection and traffic classification", *Proc. IEEE Int. Conf. Netw. Prot. (ICNP),* 2016.

[25] M.He, P.Kalmbach, A.Blenk, W.Kellerer, and S. Schmid, "Algorithm-data driven optimization of adaptive communication networks," *Proc. IEEE Int. Conf. Netw. Protocols (ICNP),* 2017.

[26] Y. Wu, S. Zhou, Y. Wei and S. Leng, "Deep Reinforcement Learning for Controller Placement in Software Defined Network," *Proc. IEEE INFOCOM 2020 Workshops*, 2020.

[27] X. Huang, Y. Tang, Z. Shao, Y. Yang and H. Xu, "Joint Switch–Controller Association and Control Devolution for SDN Systems: An Integrated Online Perspective of Control and Learning," *IEEE Trans. on Network and Service Management,* vol. 18, no. 1, pp. 315-330, March 2021.

[28] G. Burtini, J. Loeppky, and R. Lawrence, "A survey of online experiment design with the stochastic multi-armed bandit," 2015. [Online]. Available: arXiv: 1510.00757.

[29] D. Bouneffouf and I. Rish, "A survey on practical applications of multi-armed and contextual bandits," 2019. [Online]. Available: arXiv:1904.10040.

[30] E. V. Belmega, P. Mertikopoulos, R. Negrel, L. Sanguinetti, "Online convex optimization and no-regret learning: Algorithms, guarantees and applications", 2018. [Online]. Available: arXiV: 1804.04529.

[31] P.Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time Analysis of the Multiarmed Bandit Problem," *Machine Learning*, vol.47, pp.235–256, 2002.

[32] V. Kuleshov and D. Precup, "Algorithms for multi-armed bandit problems", 2014. [Online]. Available: arXiv: 1402.6028.

[33] S. Bubeck and N. Cesa-Bianchi, Regret analysis of stochastic and nonstochastic multi-armed bandit problems, Foundations and Trends in Machine Learning, 5(1):1–122, Now Publishers, Inc., 2012.

[34] J.-Y. Audibert, R. Munos, and C. Szepesvari, "Exploration-exploitation trade-off using variance estimates in multi-armed bandits", *Theoretical Computer Science,* vol.410, no.19, pp.1876-1902, April 2009.

[35] R. S. Sutton and A. G. Barto. Reinfocement Learning: An Introduction. MIT Press, 1998.

[36] N. Cesa-Bianchi, C. Gentile, G. Lugosi, and G. Neu, "Boltzmann exploration done right", *In Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS)*, 2017.

[37] P. Auer, N. Cesa-Bianchi, Y. Freund and R. E. Schapire, "Gambling in a rigged casino: The adversarial multi-armed bandit problem," *In Proc. IEEE 36th Annual Foundations of Computer Science,* 1995.

[38] E. Hazan, Introduction to online convex optimization. Found. Trends Optim., 2(3-4):157–325, 2016.

[39] P. Auer, N. Cesa-Bianchi, Y. Freund, and R. E. Schapire, "The Nonstochastic Multiarmed Bandit Problem", *SIAM J. Comput.*, vol.32, no.1, pp.48-77, 2003.

[40] N. Cesa-Bianchi, C. Gentile, and Y. Mansour. "Delay and cooperation in nonstochastic bandits" *J. Mach. Learn. Res.,* 20(1):613– 650, Jan. 2019.