algorithms

MDPI

Article

A Bayesian Multi-Armed Bandit Algorithm for Dynamic End-to-End Routing in SDN-Based Networks with Piecewise-Stationary Rewards

Pedro Santana [1,2,*] **and José Moura** [1,3,*]

[1] ISCTE—University Institute of Lisbon (ISCTE-IUL), Av. das Forças Armadas, 1649-026 Lisboa, Portugal
[2] ISTAR—Information Sciences and Technologies and Architecture Research Center, Av. das Forças Armadas, 1649-026 Lisboa, Portugal
[3] Instituto de Telecomunicações (IT), Av. das Forças Armadas, 1649-026 Lisboa, Portugal
* Correspondence: pedro.santana@iscte-iul.pt (P.S.); jose.moura@iscte-iul.pt (J.M.)

Abstract: To handle the exponential growth of data-intensive network edge services and automatically solve new challenges in routing management, machine learning is steadily being incorporated into software-defined networking solutions. In this line, the article presents the design of a piecewise-stationary Bayesian multi-armed bandit approach for the online optimum end-to-end dynamic routing of data flows in the context of programmable networking systems. This learning-based approach has been analyzed with simulated and emulated data, showing the proposal's ability to sequentially and proactively self-discover the end-to-end routing path with minimal delay among a considerable number of alternatives, even when facing abrupt changes in transmission delay distributions due to both variable congestion levels on path network devices and dynamic delays to transmission links.

Keywords: networks; routing; congestion; variable link delay; SDN; algorithm design; multi-armed bandits

Citation: Santana, P.; Moura, J. A Bayesian Multi-Armed Bandit Algorithm for Dynamic End-to-End Routing in SDN-Based Networks with Piecewise-Stationary Rewards. *Algorithms* **2023**, *16*, 233. https://doi.org/10.3390/a16050233

Academic Editor: Frank Werner

Received: 1 April 2023
Revised: 21 April 2023
Accepted: 25 April 2023
Published: 28 April 2023

1. Introduction

Due to the high complexity of managing the traffic on the network edge [1], centralized routing algorithms are more efficient than distributed ones. In fact, distributed routing algorithms, such as Border Gateway Protocol (BGP) or Open Shortest Path First (OSPF) could have significant delays to both discover an operational failure on the used routing path and find an alternative reliable path. A centralized routing algorithm with a more complete knowledge of the routing state can solve routing issues quicker than their distributed counterparts.

Along the centralized approaches to control the network resources, the Software-Defined Networking (SDN) paradigm has emerged in recent years as a very popular paradigm to reduce the cost of the network operation and maintenance, as well as to hide the network complexity from the high-level programmed algorithms. Ucoming networking scenarios are predicted to be highly volatile in terms of the network topology or network load, among other aspects. Consequently, it is imperative that the operation of centralized algorithms becomes automatically adjusted via learning agents to any unexpected changes in the network operation. The authors in [2] analyze previous work on Machine Learning (ML) techniques for routing optimization in SDN. In addition, ref. [3] revises the available literature focused on ML research opportunities and evolution in distinct areas of networking. A tutorial about the application of ML algorithms for wireless networks is available in [4]. For usage scenarios with high quantities of available data about the system operation, Deep Learning (DL) methods are becoming a popular alternative to traditional ML methods, because DL is able to automatically extract high-level features

and corresponding correlations from the input data, avoiding the need for the manual engineering of these features, which is often required by traditional ML methods [5].

An important family of ML algorithms tailored for sequential decision making is known as Reinforcement Learning (RL) [6]. Interestingly, little work has been carried out considering RL for dynamic routing controlled by SDN systems [2]. Thus, we identified the pertinence of carrying out further research on learning agents capable of enhancing the performance of SDN-based systems.

In recent years, Multi-Armed Bandit (MAB) solutions [7,8], which are a form of RL, have attracted the attention of many scholars due to their simple implementation and high flexibility, enabling agent learning in heterogeneous scenarios [9], such as recommendation systems, medicament trials, anomaly detection, and network configuration. MAB algorithms address a fundamental RL problem, which is balancing exploration and exploitation when choosing between uncertain actions: should the agent explore an action that seems to be less rewarding with the expectation that one might be wrong about it, or simply opt for the action that appears most rewarding, assuming one's current knowledge is sufficiently accurate?

To robustly handle the highly dynamic behavior of communication networks, we devised and included in our agent a novel MAB algorithm, React-UCB, prepared to learn from reward distributions that are piecewise-stationary and may abruptly change at unknown moments. React-UCB combines the following features: *optimism in the face of uncertainty*; discounted rewards to favor the present over the past; reset of reward estimates when abrupt path delay distribution changes are detected; reward correlations to transfer learning among paths; and suspended exploration after a good reward has been found to reduce the agent's accumulated regret. The proposal's main aim of finding the minimum delay path can be also seen as an intelligent way of enhancing the network operation in terms of its robustness against congestion scenarios, because when a network device is highly congested, due to internal output buffering delays, it negatively increases the transmission message delay from that device to their network neighbors. Considering this scenario, the end-to-end network paths that include more congested network devices are expected to exhibit a considerable increase on their delay. Paths with increased delays will be learned by the agent as less-rewarding paths (actions). Thus, by not recommending those less-rewarding paths to the associated SDN controller, the agent protects the controlled traffic from being negatively impacted by the more high-congestion issues.

The proposed solution was assessed by conducting a set of experiments on simulated and emulated data, regarding the most efficient system operation. The obtained results show that React-UCB can robustly track the path with the lowest end-to-end delay, meaning the less congested path, despite the presence of abrupt stationary change-points, as well as those that are planned to be difficult to discover by the agent.

The rest of the article is structured as described. Section 2 discusses the related work. Further details on the SDN-based system enhanced by the MAB agent are discussed in Section 3. Section 4 presents and discusses the evaluation results. Section 5 concludes the manuscript with future work.

2. Related Work

This section analyzes related contributions, highlighting the main novelties of the current contribution in relation to the available literature. Machine learning (ML) can be used to exploit the data originating from mobile sensors [10], but also to control the networks underlying the computational edge systems. Tang et al. [11] present a comprehensive literature analysis covering ML in the next generation of mobile networks to guarantee the quality of end-to-end applications. More specifically, they classified the available ML-related literature in the next four parts, network access in the media access control (MAC) layer, end-to-end network routing, end-to-end network congestion control and the end-to-end adaptive streaming control. Finally, they discuss some open research issues and relevant guidelines for further investigation into upcoming mobile networking scenarios.

In the networking area, the developed MAB algorithms have focused on learning about the optimal operation policies for enhancing intrusion detection [12,13] or routing via a hybrid mesh network [14] to optimize the throughput. Differently, our MAB proposal aims to achieve end-to-end path delay minimization, meaning a more robust network connectivity despite the unexpected congestion situations that occur in the network devices.

Considering a SDN scenario with multiple controllers, Huang et al. [15] investigate how learning can boost both the switch–controller association and the load of the control channel. Rischke et al. [16] and Casas-Velasco et al. [17] use RL to learn the optimum routing path, but these force the SDN controller within each system time slot to actively measure the delay in every network link, which excessively overloads the system, as evidenced in the results of Section 4.3. Alternatively, we propose a more scalable MAB-based approach, which measures a single end-to-end path delay in each time slot. We investigate a centralized MAB agent that acts as a topmost system manager to learn which end-to-end path traverses the set of less-congested network devices. Then, the SDN controller configures this optimum end-to-end path in the generic network by transferring proper flow rules to the network devices involved in that path, aiming to detect and avoid abnormal congestion situations in network devices and preserving the network quality offered to the system's topmost-layer applications or services.

A multitude of MAB algorithms have been crafted over the years to handle different types of reward distribution [7,8]. A family of algorithms is dedicated to handling stochastic reward distributions, with the well-known UCB algorithm [18] being a remarkable representative. Other algorithms, such as Exp3 [19], have been developed to tackle the more general case, in which rewards can be specified by an adversary, which may not behave in a stochastic way. Although more general, adversarial algorithms tend to underperform compared to stochastic algorithms when facing stochastic rewards.

The success of UCB has triggered the proposal of several extensions, such as KL-UCB [20], which is known to outperform the standard UCB algorithm. Despite their abilities, most MAB algorithms for stochastic domains are restricted to handle stationarity reward distributions, an assumption that hardly holds in dynamic networking scenarios. A well-known extension of the UCB algorithm for non-stationary reward distributions is known as Discounted UCB [21,22], which considers recent rewards to be more relevant than older ones. We exploit the strength of both KL-UCB and Discounted UCB to proposed an hybrid solution. However, it is often necessary to include additional mechanisms to handle sudden and abrupt distribution changes. Some algorithms cope with this problem by detecting changes in reward distributions and resetting the algorithm when those events occur, such as M-UCB [23]. We build on this intuition and tune it to produce a more robust solution for our proposed MAB agent.

When the dimension of the control problem grows, it becomes harder for MAB algorithms to properly explore the actions space in an efficient way. In these cases, it is a common strategy to exploit prior knowledge regarding the correlations between actions to estimate rewards for untried actions from the rewards collected from tried actions. A typical solution to this problem is to constrain the operation of a standard MAB algorithm to a sub-set of potentially interesting actions [24]. We approach this problem in a principled Bayesian way. In summary, our proposed algorithm, React-UCB, builds upon a judicious innovative integration and the adjustment of well-established MAB components.

3. System Architecture

Figure 1 depicts the proposed architecture. It includes an SDN controller that, at power-on, obtains the complete list of hosts, switches, and links from the controlled network topology. Using these data, the controller creates a Networkx [25] algebraic network representation. Then, the controller, by querying that abstracted topology, obtains a list with all paths among hosts. The unique identifiers of all these paths are transferred to an MAB agent that decides, in each system time slot, which path (arm in MAB nomenclature) to use for message-routing. These decisions are made to maximise the expected reward

in the long run; reward, in this case, represents the inverse of end-to-end delay associated with the path previously selected by the agent. Hence, in each time slot, the MAB agent chooses a path and relays its unique identifier to the controller, which, in turn, deletes old flow rules and installs new ones in each switch of that path. Then, the controller actively measures the selected path's end-to-end bidirectional delay and returns it to the MAB agent so that a reward can be computed.

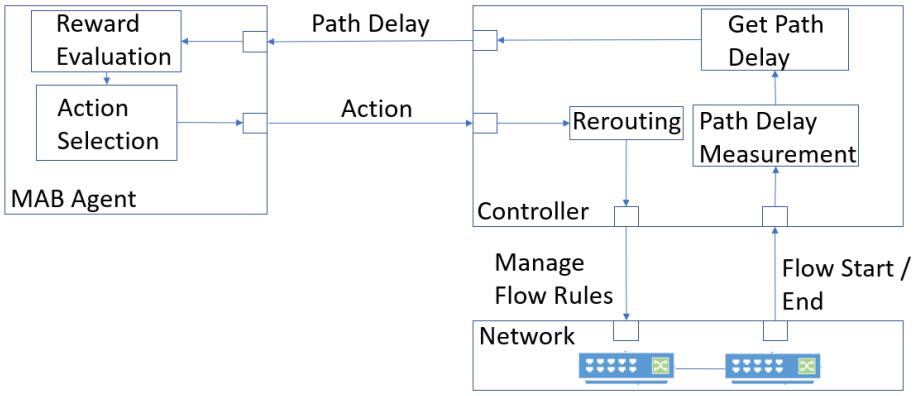

Figure 1. Proposed architecture of a programmable system augmented by an MAB learner agent for online optimum end-to-end dynamic routing path.

The end-to-end path delay is the sum of the individual delay contributions by all the network devices involved in that path. In addition, the delay induced by each network device to the end-to-end path is directly proportional to the local congestion level at that device. This means, when considering an end-to-end path with two devices, A and B, if device A is more congested than device B, device A's delay contribution to the end-to-end delay path will be higher than that of device B.

The MAB agent is controlled by a novel and generic MAB algorithm, React-UCB, which we introduce to optimize the routing in dynamic communication networks. The next sections detail React-UCB, explaining how it decides the arm to pull under piecewise-stationary reward distributions in each time step. The main goal of React-UCB is to learn about end-to-end path delay minimization (i.e., learn about the more robust end-to-end path with the minimum aggregated congestion level), mapping arms to paths and rewards to path delays.

3.1. The MAB Problem

At each time step $t \in \{1, 2, \ldots\}$, the MAB agent pulls (selects) and uses arm $I_t \in \mathcal{K}$, $\mathcal{K} = \{1, 2, ..., K\}$, according to a policy π and, as a consequence, the environment returns a corresponding reward $X_t(I_t) \in \Theta$ with $\Theta = [0, 1]$ to signal how beneficial it was to pull arm I_t.

In this article, we address a specific class of non-stationary bandit problems, in which the reward distributions are piecewise-stationary and may abruptly change at unknown time steps; hereafter, change-points. To cope with this type of distribution, the rewards for each arm $i \in \mathcal{K}$, $\{X_t(i)\}_{\forall t \in \{1,2,\ldots\}}$, are a sequence of independent random variables from potentially different distributions, which are unknown to the agent and could abruptly change over time (time-dependent). $\mu_t(i)$ is the expected reward of arm i.

The policy performance π depends on the regret, i.e., the expected difference between the policy's collected rewards and the rewards that would have been gathered by the optimal policy (an oracle pulling the arm with the largest expected reward) over a time horizon T:

$$\mathbb{E}_\pi \left[\sum_{t=1}^{T} \left(\max_{i \in \mathcal{K}} (\mu_t(i)) - \mu_t(I_t) \right) \right], \tag{1}$$

where $\mathbb{E}_\pi$ is the expectation under policy π.

3.2. Preliminary: UCB / D-UCB

The UCB-1 method [18] chooses the arm i that maximizes the upper bound of a confidence interval for the stationary expected reward. To handle non-stationary expected rewards, the Discounted UCB (D-UCB) [21,22], a UCB-variant, provides higher weights to recent rewards than to older rewards, considering a discounted empirical average reward, $\bar{X}_t^{std}(i)$ and a discounted number of arm pulls, $N_t^{std}(i)$, in the computation of the confidence interval's upper bound $U_t^{std}(i)$:

$$U_t^{std}(i) = \bar{X}_t^{std}(i) + C\sqrt{\frac{\log \sum_{j=1}^{K} N_t^{std}(j)}{N_t^{std}(i)}}, \tag{2}$$

$$\bar{X}_t^{std}(i) = \frac{1}{N_t^{std}(i)} \sum_{s=1}^{t} \gamma^{t-s} \mathbb{1}_{\{I_s=i\}} X_s(i), \tag{3}$$

$$N_t^{std}(i) = \sum_{s=1}^{t} \gamma^{t-s} \mathbb{1}_{\{I_s=i\}}, \text{ with } N_1^{std}(i) = 0, \tag{4}$$

where the scalar $\gamma \in [0,1]$ is a discount factor, $C > 0$ is a constant controlling the exploration level, and *std* indicates that this formulation refers to the standard D-UCB. D-UCB selects the arm to be pulled at time t as the one with the highest upper confidence bound (principle known as *optimism in the face of uncertainty*):

$$I_t^{std} = \arg\max_{i \in \mathcal{K}} U_t^{std}(i). \tag{5}$$

3.3. Incremental Formulation and Reset

To speed up computation, React-UCB avoids the summations in $\bar{X}_t^{std}(i)$ and $N_t^{std}(i)$ with an equivalent incremental formulation:

$$\bar{X}_t(i) = \gamma \bar{X}_{t-1}(i) R_{t-1} + \mathbb{1}_{\{I_t=i\}} X_t(i), \tag{6}$$

$$N_t(i) = \gamma N_{t-1}(i) R_{t-1} + \mathbb{1}_{\{I_t=i\}}, \tag{7}$$

$$\bar{X}_0(i) = 0, N_0(i) = 0, R_0 = 0, \tag{8}$$

where $R_t \in \{0,1\}$ is a modulatory variable that allows for the agent to reset $\bar{X}_t(i)$ and $N_t(i)$, when an abrupt reward distribution change in the pulled arm I_t is detected (such a change would be difficult to accommodate with discounting alone). This change detection occurs on the list of rewards accumulated by each agent's arm $i \in \mathcal{K}$, whose size $n_t(i)$ increases whenever i is pulled at time step t, and whose j-th element is given by $x_j(i)$:

$$n_t(i) = n_{t-1}(i) + \mathbb{1}_{\{I_t=i\}}, \tag{9}$$

$$x_{j<1}(i) = n_{j<1}(i) = 0, \tag{10}$$

$$x_{n_t(i)}(i) = \mathbb{1}_{\{I_t=i\}} X_t(i) + \mathbb{1}_{\{I_t \neq i\}} x_{n_{t-1}(i)}(i). \tag{11}$$

Similar to M-UCB [23], React-UCB detects changes by comparing some reward statistics across the two most recent and contiguous series of w rewards gathered by the pulled arm, I_t. The two ordered sets of indices of these two series in the rewards list are:

$$v_{1,t} = \{n_t(I_t) - w, \ldots, n_t(I_t) : n_t(I_t) \geq 2w\}, \tag{12}$$

$$v_{2,t} = \{n_t(I_t) - 2w, \ldots, n_t(I_t) - w : n_t(I_t) \geq 2w\}. \tag{13}$$

M-UCB detects changes based on the absolute difference between average rewards across the two contiguous series as a function of a fixed average reward difference threshold. However, a fixed threshold can rapidly become inaccurate in scenarios in which the rewards'

variance changes unpredictably. That is, a low threshold may result in excessive false positives (i.e., resets being issued even when the reward distribution has not changed) if the rewards' variance rises unexpectedly, whereas a high threshold may result in excessive false negatives (i.e., resets not issued when the reward distribution has changed) if the rewards' variance lowers unexpectedly. The practical implication of being unable to properly determine when what has been learned should be forgotten (i.e., a reset should be issued) increases regret.

Alternatively, a change detection in React-UCB is only reported if: (i) the average reward difference between $\{x_i(I_t)\}_{\forall i \in \nu_{1,t}}$ and $\{x_i(I_t)\}_{\forall i \in \nu_{2,t}}$ is above a small threshold ζ (small, 0.1 in our tests, to avoid false negatives); (ii) it is statistically significant (not due to chance); and (iii) it is of practical significance for the system optimization goal. Hence, in React-UCB, the chances of triggering a reset are dynamically adjusted according to the rewards' variance, instead of being ruled by a fixed threshold throughout the whole time horizon.

Statistical significance holds when $p_t < \alpha_p$, where p_t is the probability of the observed difference in average rewards under the null hypothesis $H_0 \triangleq (\mu_{1,t} = \mu_{2,t})$ and for a significance level α_p (we opt for the standard $\alpha_p = 0.05$), and is estimated via an independent, two-sample, two-sided Welch's t-test:

$$\mu_{1,t} = w^{-1} \sum_{v \in \nu_{1,t}} x_v(I_t), \tag{14}$$

$$\mu_{2,t} = w^{-1} \sum_{v \in \nu_{2,t}} x_v(I_t). \tag{15}$$

Practical significance holds when $\Delta_t > \alpha_\Delta$, where Δ_t is the effect size metric known as Glass' Δ and α_Δ is the effect size threshold (we used $\alpha_\Delta = 0.8$ for large effect size [26]). Glass' Δ was selected instead of the well-known Cohen's d because the former (but not the latter) does not assume equal variances in both groups:

$$\Delta_t = \sigma_{1,t}^{-1} |\mu_{2,t} - \mu_{1,t}|, \tag{16}$$

$$\sigma_{1,t} = \sqrt{w^{-1} \sum_{i \in \nu_{1,t}} (x_i(I_t) - \mu_{1,t})^2}. \tag{17}$$

Resetting aims to level all arms when the previous agent's knowledge is expected to be more detrimental than beneficial. This occurs when a reward increase above the current best reward estimate is observed in sub-optimum arms and when a reward decrease is observed in the arm considered the best. In React-UCB, all the pre-conditions for a reset to occur are considered in the reset modulatory variable R_t, as follows:

$$R_t = \mathbb{1}(C_{1,t} \wedge C_{2,t} \wedge (C_{3,t} \vee C_{4,t})), \tag{18}$$

$$C_{1,t} = \mathbb{1}(n_t(I_t) \geq 2w), \tag{19}$$

$$C_{2,t} = \mathbb{1}(p_t < \alpha_p \wedge \Delta_t > \alpha_\Delta \wedge |\mu_{2,t} - \mu_{1,t}| > \zeta), \tag{20}$$

$$C_{3,t} = \mathbb{1}(I_t = m_t \wedge \mu_{2,t} < \mu_{1,t}), \tag{21}$$

$$C_{4,t} = \mathbb{1}\left(I_t \neq m_t \wedge \mu_{2,t} > \mu_{1,t} \wedge \mu_{2,t} \geq \frac{\bar{X}_t(m_t)}{N_t(m_t)}\right), \tag{22}$$

where the most frequently pulled arm in recent history is indicative of the best arm for resetting purposes:

$$m_t = \arg\max_{k \in K} (n_t(k) - n_{t-2w}(k)). \tag{23}$$

3.4. Upper Confidence Bound

For the upper confidence bound's computation, React-UCB builds upon KL-UCB [20], a UCB variant known to achieve lower regret than the standard UCB-1. However, the direct integration of KL-UCB in React-UCB would be limited in scope, as KL-UCB has not been designed to cope with non-stationary reward distributions. That is, KL-UCB can hardly keep track of the optimal arm in the face of change. This limitation stems from the fact that KL-UCB formulation accounts for undiscounted rewards. React-UCB reformulates KL-UCB to account for discounted empirical average rewards, $\bar{X}_t(i)$, and a discounted number of arm pulls, $N_t(i)$, enabling its operation in non-stationary regimes:

$$U_t(i) = \max\left\{q \in \Theta : d\left(\frac{\bar{X}_t(i)}{N_t(i)}, q\right) \leq \frac{C\log(t')}{N_t(i)}\right\}, \tag{24}$$

where $t' = \sum_{j=1}^{K} N_t(j)$ and $d(x,y) = x/y - 1 - \log(x/y)$ is the Kullback–Leibler divergence for exponential distributions (an adequate fit to the distribution of time delays).

3.5. Correlated Arms

The deterministic nature of D-UCB poses challenges in the detection of reward changes in currently low-performing arms, which may eventually become the optimal ones in subsequent time steps, without incurring expensive arm exploration by increasing C. Resets allow the agent to deal with this problem, enabling all actions to be sufficiently sampled. M-UCB samples all arms periodically to achieve this purpose. However, setting a proper full sampling period may be difficult without knowing the environment dynamics beforehand. A long period may not be sufficient to gather enough evidence to engage in a reset in due time, whereas a short period may gather redundant evidence, negatively impacting regret without added value. Moreover, the rewards from one arm can provide information on the (pseudo-)rewards of correlated arms [24], as often occurs in paths sharing network links, rendering these valuable in spreading the sampling over time rather than periodically.

To cope with these challenges, React-UCB samples actions according to a distribution, ensuring that all arms have a positive probability of being sampled in every time step, while ensuring that the more promising arms, according to their accumulated rewards and correlations, are more densely sampled. In this way, the algorithm becomes continuously sensible to reward distribution changes, rather than only periodically. This stochastic approach borrows from algorithms such as Thompson-Sampling [27] and Exp3 [19], bringing a new algorithmic ability to handle adversarial bandits. An alternative to this approach would be to use pseudo-rewards to restrict, in each time step, the activity of the basic bandit algorithm (e.g., UCB) to a sub-set of competitive arms [24]. However, by not including randomness and by ruling out some arms in each time step, it is not clear how this alternative handles piecewise-stationary bandits.

Let us define Y_t and Y_{t-1} as random variables representing the distributions underlying arm pulls I_t and I_{t-1}, respectively. Although arm pulls in React-UCB are not actually ruled by these distributions, these will be used to reason about the value of each arm. Let us also define the probability that arm i has been pulled in the previous step, $t-1$, as the softmax of its upper confidence bound, which, despite not being accurate, is a useful approximation (η to sharpen the distribution, $\eta = 10$ in our tests):

$$P(Y_{t-1} = i) = \frac{e^{\eta U_{t-1}(i)}}{\sum_{k \in \mathcal{K}} e^{\eta U_{t-1}(k)}}. \tag{25}$$

Let the probability that arm i is the pulled one in the current time step t, given that arm j was the pulled one in the previous time step, $t-1$, be defined as the softmax of a correlation function between arms a and b, $s(a,b) : \mathcal{K}^2 \mapsto [0,1]$, which, in the network case, is the ratio of shared links between arms:

$$P(Y_t = i|Y_{t-1} = j) = \frac{e^{\eta s(i,j)}}{\sum_{k \in \mathcal{K}} e^{\eta s(i,k)}}. \quad (26)$$

It follows that the joint probability of arm i being selected at time step t and arm j being selected at time step $t-1$ is given by:

$$P(Y_t = i, Y_{t-1} = j) = P(Y_t = i|Y_{t-1} = j)P(Y_{t-1} = j). \quad (27)$$

The probability of arm i being selected at time step t can be obtained by marginalizing over the distribution of the arms selected in the previous time step $t-1$:

$$P(Y_t = i) = \sum_{k \in \mathcal{K}} P(Y_t = i, Y_{t-1} = k). \quad (28)$$

Finally, the probability mass function F_{Y_t} identifies promising arms (i.e., those correlated with the most-rewarded arms):

$$F_{Y_t}(i) = P(Y_t = i), \forall i \in \mathcal{K}. \quad (29)$$

3.6. Reactive-UCB

Algorithm 1 presents the proposed MAB algorithm (also includes the default parametrisation used in the experiments), Reactive-UCB, which evaluates the equations presented in the previous sections (assuming constant input in Line **1**) to exploit the synergies of the following arm-sampling policies: opportunistic, complete, correlated, and optimistic.

Opportunistic sampling occurs when the average reward is good enough, i.e., statistically above ξ (0.9 in our tests) and, thus, exploration is not required (Lines **7–8**). The three other sampling policies are only available when opportunistic sampling is not active.

Complete sampling refers to the sequential execution of every arm $k \in \mathcal{K}$, which occurs at $t = 1$ and every time a reset occurs, i.e., whenever $R_t = 1$ (Lines **4–5**). This ensures that all arms are fairly sampled at least once.

Optimistic sampling (in the face of uncertainty), which occurs with probability $1 - \epsilon$ ($\epsilon = 0.1$ in our tests) when not performing complete sampling, deterministically selects the arm with the highest upper confidence bound (breaking ties randomly) (Lines **9–10**).

Correlated sampling, which occurs with probability ϵ when not performing complete sampling, selects the arm by sampling the probability mass function F_{Y_t} (Line **12**). This ensures that all arms are likely to be selected by exploiting their correlations.

The adaptation of the algorithm to the SDN-based routing problem occurs at Line **15** by assuming that an arm represents a path and that the reward is defined in terms of the path's delay. Formally, the reward obtained from choosing a path $i \in \mathcal{K}$, $X_t(i)$, is defined as a function of the observed instantaneous path's delay, $\delta_t(i)$, which ensures that: (i) the higher the delay, the lower the reward; (ii) the delay's changes are felt more strongly in the reward for lower delay values, helping the agent to be more accurate in its best-arm choices; and (iii) 10 ms is the lowest path delay. Formally,

$$X_t(i) = \max\Big(0, 0.5 \log\Big(\max(\delta_t(i), 0.01)^{-1}\Big)\Big). \quad (30)$$

3.7. Complexity Analysis

This section presents a complexity analysis of the proposed algorithm, React-UCB. For this purpose, the big-O notation is used to describe the worst-case asymptotic execution time of the algorithm, $T(n)$, as the number of arms (n in the context of complexity analysis) grows to infinity. The goal of this analysis is to assess the scalability of the algorithm as the problem size increases. The number of arms was considered the growing variable as it is the one with highest impact in terms of execution time.

As depicted in Algorithm 1, in each time step t, the algorithm is mostly composed of a set of sequential and conditional constant time operations (e.g., Line **5**), which have a joint order of complexity $O(1)$. In addition to these constant time operations, Algorithm 1 and

the equations evaluated by it also include vector-wise operations, such as element-wise products, summations, and *max* operations (e.g., Line **10**), which exhibit a linear order of complexity, $O(n)$, as well as their resulting sequences.

Algorithm 1 Reactive-UCB (with default parametrisation)

1: **Input:** $\mathcal{K} = \{1, 2, .., 25\}, \gamma = 0.99, w = 100, C = 0.01$
   Input: $\alpha_p = 0.05, \alpha_\Delta = 0.8, \epsilon = 0.1, \zeta = 0.1, \xi = 0.9$
2: $\tau \leftarrow 1$
3: **for** $t = 1, 2, \ldots$ **do**
4: **if** $t < |\mathcal{K}|$ **or** $t - \tau < |\mathcal{K}|$ **then**
5: $I_t \leftarrow t - \tau + 1$
6: **else**
7: **if** $|\mu_{2,t-1} - \xi| / \sigma_{1,t-1} > \alpha_\Delta$ **then**
8: $I_t \leftarrow I_{t-1}$
9: **else if** $u_t < 1 - \epsilon$ with $u_t \sim U(0, 1)$ **then**
10: $I_t \leftarrow \arg\max_i \{U_t(k)\}_{\forall k \in \mathcal{K}}$ [Equation (24) with γ, C]
11: **else**
12: $I_t \leftarrow f_t$ with $f_t \sim F_{Y_t}(\mathcal{K})$ [Equation (29)]
13: **end if**
14: **end if**
15: Pull arm I_t and collect reward $X_t(I_t) \in [0, 1]$
16: **if** $R_t = 1$ [Equation (18) with $w, \alpha_p, \alpha_\Delta, \zeta$] **then**
17: $\tau \leftarrow t$
18: **end if**
19: **end for**

The computation of the probabilistic mass function described by Equation (29), used to account for correlated arms, requires a quadratic execution time, $O(n^2)$ due to the marginalization in Equation (28). As a result, the algorithm's asymptotic global execution time, per time step t, is, at most, $T(n) = O(n^2)$. Nevertheless, the quadratic component applies only to a small set of mathematical operations, and thus has a limited impact to the global execution time unless the number of arms grows to unpractical numbers. Section 4.4 presents a set of empirical results that corroborate this prediction and the underlying theoretical analysis.

4. Evaluation

To assess the proposed system's ability to track the optimal network route under non-stationary settings, an evaluation setup based on simulated and emulated data was designed and a set of experimental results were collected. The evaluation setup, the collected results, and their analysis are presented below.

4.1. Datasets and Evaluation Setup

In order to facilitate the integration of the developed system with other tools and libraries commonly used in networks and multi-armed bandit research, React-UCB was implemented in the programming language Python. The KL-UCB component included in React-UCB is based on a well-known Python implementation from [28].

To evaluate React-UCB's performance with its default parametrisation, as specified in Algorithm 1, a set of experiments were run on two different evaluation setups: emulation setup and simulation setup. In the emulation setup, the agent is tasked to find the lowest-delay path in an emulated network topology. Conversely, in the simulation setup, the agent is challenged to find the arm with highest reward in a generic simulation of stochastic bandits. Both setups assume that the agent needs to handle $K = 25$ paths (arms), whose normally distributed delays are piecewise-stationary, with reward distribution change-points occurring every $P = 1000$ time steps, up to the time horizon $T = 10^5$ and $T = 10^6$ in

the emulation and simulation setups, respectively. The larger time horizon used for the simulation setup aims to cope with the higher diversity of distribution changes that occur in this setup.

The two evaluation setups were used due to their complementary characteristics. A network emulator is adequate for testing the system against highly likely reward distributions, enabling an analysis focused on expected events. However, it is also important to assess the system's behaviour when facing uncommon distribution changes, which are hard to find with a network emulator without engaging with extensive topology manipulations. On the other hand, a generic simulation setup is adequate for testing the system against arbitrary network delay distribution changes, enabling an analysis that is independent of the network topologies specifically designed for the evaluation process.

The emulation setup is based on the open-source network emulator Mininet [29]. Mininet was selected as it is a well-established solution for the creation of realistic virtual networks capable of running real, that is, deployable code. Moreover, Mininet is particularly suited to studying SDN-based systems. By relying on Mininet, the emulated network operation is highly accurate because it runs in real-time and the programmed delays and jitters operate as if they were emerging from real hardware.

Figure 2 depicts the emulated network for this study, whose topology, composed of seven OpenvSwitches, results in 25 end-to-end paths between end-hosts H1 and H2. The seven switches are controlled via OpenFlow v1.3 by a Ryu [30] SDN controller. In the emulation setup, React-UCB is tasked to select each time slot which one of the 25 end-to-end paths is selected for data transmission (an arm per path).

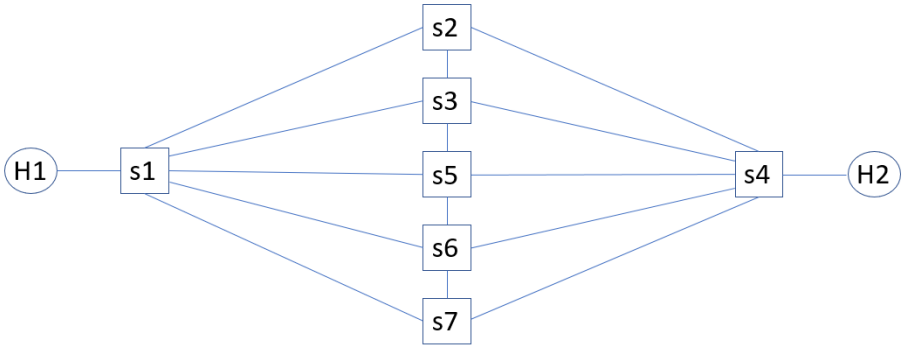

Figure 2. Network topology with seven OpenvSwitch switches and 25 end-to-end paths between end hosts H1 and H2. The switches are controlled via OpenFlow v1.3 by a Ryu [30] SDN controller (not visualized).

During the emulation runs, and as any used path can be decomposed in several hops, each hop's delay contribution to the end-to-end single direction path delay is the sum of the processing delay at the hop source switch (influenced by the congestion level at the hop source switch), the link transmission delay between the hop source switch and the hop destination switch (influenced by the sum of the link transmission delay and the link propagation delay). The bidirectional delay in each end-to-end path changes according to a set of three distributions $\mathcal{D} = \{d_1, d_2, d_3\}$, which are summarized in Table 1. The table only presents paths that are optimal for at least one of the distributions. For every P steps, the active distribution changes. Concretely, assuming $d \in \mathcal{D}$ to be the currently active distribution, the next active distribution is randomly sampled from $\mathcal{D} \setminus \{d\}$.

Table 1. Path delay variation (AVG ± STDEV ms) across emulated distributions; the best distribution path in each distribution is in bold. Each path is represented by a tuple composed of the switches along the path. Only a subset of 3 of the 25 possible end-to-end paths are listed.

Distribution	Path (1, 2, 4)	Path (1, 3, 4)	Path (1, 5, 4)
d_1	**200 ± 20.0**	230 ± 23.0	270 ± 27.0
d_2	200 ± 20.0	**100 ± 10.0**	240 ± 24.0
d_3	80 ± 8.0	100 ± 10.0	**20 ± 2.0**

The distributions in D were carefully crafted to challenge React-UCB's ability to detect and react to sudden and abrupt reward distributions. This is attained by ensuring that the path with minimum delays is different in each distribution and that it is not always possible for React-UCB to detect distribution changes by simply sensing a reward drop in the current optimal path (this reward may remain the same across distribution change-points), thus requiring a proper handling of the exploration–exploitation trade-off.

In the simulation setup, each of the 25 arms returns a stochastic reward according to a normal distribution, whose parameters are randomly selected in every reward distribution change-point. Hence, in this setup, React-UCB faces challenging scenarios, in which reward distributions change in unpredictable and diverse ways.

Concretely, in each simulation time step t, the delay expectation of every path (arm) $i \in \mathcal{K}$, $\mu_t^\delta(i)$, and standard deviation $\sigma_t^\delta(i)$, are either updated or maintained. Formally, if $t = 1$ or t is a multiple of P, then $\mu_t^\delta \sim U(0,1)$ and $\sigma_t^\delta \sim U(0,0.2)$; otherwise $\mu_t^\delta \leftarrow \mu_{t-1}^\delta$ and $\sigma_t^\delta \leftarrow \sigma_{t-1}^\delta$. The delay observed by the agent when selecting path (pulling arm) I_t is obtained by sampling a normal distribution parameterized with the randomly selected expected delays and their standard deviations, $\delta_t(I_t) \sim \mathcal{N}(\mu_t^\delta(I_t), \sigma_t^\delta(I_t))$.

4.2. Results

This section presents the set of results that were collected with both emulation and simulation setups with four different tested React-UCB configurations. Testing with different configurations aims to assess the contribution of each React-UCB's feature (e.g., arms correlation, reset functionality) to the overall algorithm's performance.

The baseline configuration, referred to as B, represents the implemented discounted KL-UCB (Equation (24)) augmented with ϵ-greedy for coverage guarantees in non-stationary reward distributions. This configuration, composed of standard components, serves as a benchmark for the remaining configurations.

The second configuration, hereafter referred to BR, extends configuration B by including the reset functionality. Hence, a performance improvement in configuration BR over configuration B should be an indication that the reset functionality provides added value to the overall solution.

The third configuration, hereafter referred to BRC, builds upon configuration BR by including arms' correlation information in the decision-making process. The goal of this configuration is to assess whether the correlation between arms, that is, knowledge regarding how many links are shared across paths, can make a valuable contribution to the provision of pseudo-rewards.

Finally, the fourth configuration, referred to as BRCT, extends configuration BRC by including the mechanism that allows for the algorithm (Lines **7–8** in Algorithm 1) to cancel exploration when rewards rise above a given satisfactory threshold, that is, the rewards are good enough for the task at hand.

Table 2 summarizes the average (AVG) and the standard deviation (STD), per time step, of the regret obtained with each of the four tested configurations. These statistics were computed by considering the total regret accumulated over 30 independent runs, divided by the number of time steps in each run (a run lasts T time steps, with T varying according to the evaluation setup, as described earlier). A total of 30 independent runs was selected to increase the power of the statistical tests to compare several configurations; that is, to

minimise the impact that the randomness in both agent and reward systems may have on the variance in the results. The independence between runs was enforced by using a different random seed in the beginning of each run.

Table 2. Regret per iteration (AVG $\pm$ STDEV).

Config.	Simulation Setup	Emulation Setup
B	0.1346 ± 0.0020	0.0755 ± 0.0021
BR	0.1254 ± 0.0017	0.0712 ± 0.0021
BRC	0.1041 ± 0.0017	0.0646 ± 0.0023
BRCT	0.1028 ± 0.0019	0.0453 ± 0.0023

As expected, Table 2 shows that including each React-UCB's component results in a gradual decrease in terms of regret. These results are confirmed in both simulation and emulation setups, meaning that these results apply to both network and generic application scenarios. Hence, the devised solutions reset the MAB when a reward distribution change is detected, exploit the arms correlation to build pseudo-rewards, and stop exploration when task-dependent, good-enough rewards are reached. These all positively contribute to the algorithm's performance. In fact, the fourth configuration, BRCT, exhibits only 70% of the regret suffered by the baseline (benchmark) configuration, B, across both setups. Hence, this result indicates an overall important reduction in terms of agent regret.

Considering the emulation setup, Table 3 shows that the increase in the agent's average reward per iteration is well-aligned with the previously discussed decrease in the agent's regret in the Table 2. Table 3 also shows that enhancements on the agent's average reward per iteration make an important positive contribution to the network throughput, mainly after the last component (T, which inhibits the agent's exploration after a good reward has been discovered) is activated in the MAB's algorithm.

Table 3. Reward and throughput (messages/s) per iteration (AVG $\pm$ STDEV) for the emulation setup.

Config.	Reward	Throughput (Messages/s)
B	0.5422 ± 0.0102	12.1426 ± 1.0481
BR	0.5464 ± 0.0102	12.3836 ± 1.0481
BRC	0.5529 ± 0.0103	12.7605 ± 1.0487
BRCT	0.5723 ± 0.0118	13.9523 ± 1.0560

Table 4 shows that the regret differences observed between the four configurations (Table 2) are all statistically significant for a 99 % confidence level ($p < 0.01$), according to the Student's two-sample unequal variance t-test, considering a two-tailed distribution. Hence, it is very unlikely that the observed performance differences associated with the diverse agent components are due to chance. Table 4 also shows that the observed regret differences have practical significance; that is, they are likely to have practical benefits. Here, practical significance is defined in terms of effect size, measured with Cohen's d metric. All configuration comparisons indicated in Table 4 exhibit a large effect size [26] ($d \geq 0.8$), except for the difference between configurations BRC and BRCT, which presents a medium effect size [26] ($d \in [0.5, 0.8)$). Resets occur with a median delay from the reward distribution change-points of 11 time steps and 23 time steps in the emulation and simulation setups, respectively. Hence, assuming a plausible median period of 100 ms between time steps, resets are expected to occur with a median delay of approximately 1.1 s and 2.3 s in scenarios with similar reward distributions to those present in the emulation and simulation setups, respectively. This shows the algorithm's ability to adequately react to sudden and abrupt reward distribution changes, explaining its low empirical regret.

In summary, the results show that the several React-UCB's components provide added value to the whole algorithm and that this conclusion is supported by strong statistical and

practical significance validation tests, which were performed on the evaluation results of our proposal.

Table 4. Regret statistical tests: p-values and Cohen's d. The former test evaluates the significance of the statistical difference between the diverse configuration options of the MAB agent, and the latter test reveals the statistical practical significance of the agent components' contributions to the final MAB agent's learning performance.

Null Hypothesis	Simulation Setup	Emulation Setup
B = BR	$p < 0.001, d = 4.97$	$p < 0.001, d = 2.02$
BR = BRC	$p < 0.001, d = 12.50$	$p < 0.001, d = 2.93$
BRC = BRCT	$p < 0.01, \ d = 0.71$	$p < 0.001, d = 8.38$

4.3. Comparison to Standard SDN

The previous section analysed the benefits of the elements that comprise the devised MAB agent for end-to-end path delay minimisation, avoiding the negative impact of congested network devices in the quality of top system applications or services. This section compares the benefits of using an MAB agent for this purpose compared to a standard SDN solution without the support of any intelligent agent.

In a standard SDN solution, all links need to be periodically probed to determine their current delay. Without this periodic and global probing process, the SDN controller is unable to determine the optimal path for packet exchange. The periodic and global nature of the traffic imposed by these probing messages results in a considerable overhead on the network. Figure 3 presents the number of messages that need to be exchanged by a classic SDN-based solution, when the network is composed of four, seven, and ten switches, at both network and control levels. Analyzing these results, it is clear that the higher the number of switches, the higher the number of probing messages that flow through the network.

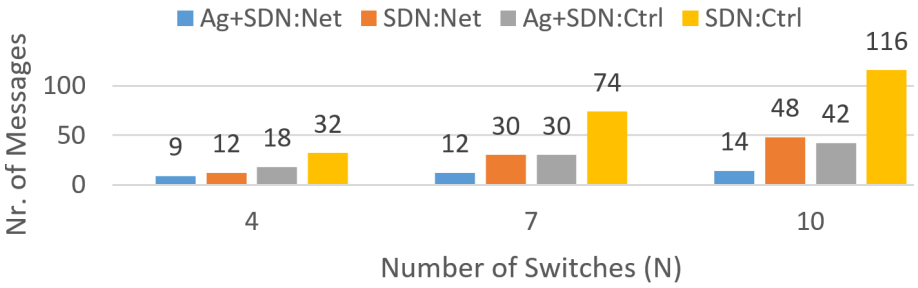

Figure 3. Average overhead comparison at both network (Net) and control (Ctrl) levels between controller enhanced by the MAB agent (Ag+SDN) and standalone controller (SDN). This considers several topology sizes.

The MAB-based solution for end-to-end path delay minimisation proposed in this article aims to reduce the number of probing messages by properly handling the exploration–exploitation trade-off. Figure 3 shows that the number of messages exchanged at both the network and control levels of the MAB agent is systematically lower than the number of messages exchanged by a standard SDN solution. Interestingly, the growth rate in the number of messages, as a function of the number of switches, is lower with the MAB-based solution than with the standard SDN solution. Hence, the MAB-based solution offers higher scalability than the standard SDN solution as the network increases in size.

4.4. Computational Complexity Results

React-UCB's complexity analysis, as presented in Section 3.7, was verified with empirical results. These results were obtained on a MacBook Pro 2019, with 32 GB DDR4 and

2.3 GHz 8-Core Intel Core i9 (Apple Inc., United States), using the simulation setup with 25, 100, 200, 300, and 400 arms.

Figure 4 presents the obtained average CPU execution time per time-step t (i.e., per React-UCB's decision making) over a horizon of $T = 10^4$ time steps. Execution time was estimated with `time.process_time()` Python function in a conservative manner, that is, by including the whole algorithm, simulation setup, and corresponding interactions.

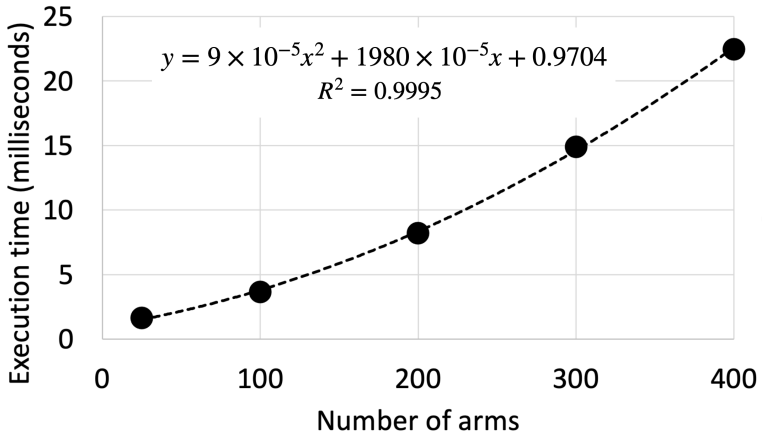

Figure 4. React-UCB's average execution time per time step over $T = 10^4$ time steps. The dashed line represents the second-order polynomial curve that was fit to the execution times represented by the black dots.

A second-order polynomial curve was fit to the obtained execution timings to assess the theoretical algorithm's asymptotic execution time of $T(n) = O(n^2)$ (see Section 3.7). The quadratic order of complexity was confirmed by the almost perfect coefficient of determination achieved by the regression process, $R^2 = 0.9995$. It is also notorious that the quadratic coefficient (9×10^{-5}) is several orders of magnitude smaller than the linear coefficient (1980×10^{-5}). This is due to the fact that the quadratic order of complexity was limited to the probabilistic inference step. The smaller quadratic coefficient results in a low execution time for a practical number of arms. For instance, considering 25 and 400 arms, the algorithm only takes, on average, 1.63 ms and 22.46 ms to output a decision per time step t, respectively. These are adequate processing timings from a practical standpoint.

5. Conclusions

The paper presents a novel approach to the online, optimum, end-to-end dynamic routing of data flows in the context of programmable networking systems. This is based on a proposed piecewise-stationary Bayesian MAB algorithm, React-UCB, which is a careful planned combination of several enhancing features for discovering the bidirectional end-to-end network path with the minimum delay from a high number of alternatives. In addition, our solution protects the quality of topmost system applications or services despite the occurrence of both unpredictable congestion situations at network devices and unexpected link delay variations.

React-UCB's features allow for the algorithm to properly cope with abrupt changes in the reward distributions by: (i) assuming the heuristic *optimism in the face of uncertainty*; (ii) discounting rewards to favor recent feedback over older feedback; (iii) resetting previously learned data after abrupt changes in the path delay's distribution are detected; (iv) considering reward correlations to transfer learning across paths; and (v) suspending exploration once a good reward is found. Jointly, these features positively reduce the agent's accumulated regret, contributing to an efficient network resource operation.

A set of experiments on simulated and emulated data were conducted to assess the individual contribution of each system's component. Comparing the diverse experimental results, we obtained solid conclusions based on the statistically significant performance

differences among the diverse system's components. The results also show React-UCB's ability to properly handle the exploration–exploitation trade-off in a more scalable way than alternative non-learning programmable solutions, which uniquely rely on periodic link delay monitoring.

Our current proposal can control, in a centralized way, operations within a single-domain network infrastructure. However, in our opinion, the current proposal could be investigated in more complex scenarios, involving several network domains, forming a federated scenario. Considering this, it will be necessary to deploy a distributed (and coherent) learning solution in the diverse agents, with each one centrally controlling, via the associated SDN controller, its own network domain.

The current research could be further developed to allow for the use of contextual bandits [31] to boost the optimal path's tracking convergence, as well as to change the reward function to be energy-aware. We are also planning to study React-UCB in other very challenging piecewise-stationary problems, such as elastic computational and networking assets at the network edge, following the trend of system's load demand [32,33]. Finally, although the used emulation setup is highly realistic, we will further evaluate React-UCB in a real multi-domain network that is congested by traffic from concurrent flows. The proposed framework will be compared with other frameworks.

Author Contributions: Conceptualization, methodology, software, validation, formal analysis, investigation, resources, data curation, writing—original draft preparation, writing—review and editing, project administration, funding acquisition: P.S. and J.M. All authors have read and agreed to the published version of the manuscript.

Funding: The work of Pedro Santana was supported by Fundação para a Ciência e Tecnologia (FCT) / Ministério da Ciência, Tecnologia e Ensino Superior (MCTES) under ISTAR projects UIDB/04466/2020 and UIDP/04466/2020. José Moura acknowledges the support given by Instituto de Telecomunicações. His work was funded by FCT/MCTES through national funds and, when applicable, co-funded by European Union (EU) funds under the project UIDB/50008/2020.

Informed Consent Statement: Not applicable.

Data Availability Statement: The datasets used and/or analysed during the current study are available from the authors on reasonable request.

Acknowledgments: The first author, Pedro Santana, acknowledges the support given by Information Sciences and Technologies and Architecture Research Center (ISTAR); the second author, Jose Moura, acknowledges the support given by Instituto de Telecomunicações (IT).

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

Notation

The following notation is used in this manuscript:

$\mathbb{1}_{\{c\}}$	Indicator function, which equals 1 if c is true and equals 0 if otherwise.
α_p	Significance level threshold.
α_Δ	Effect size threshold.
ϵ, C	Exploration probability, exploration level.
γ	Discount factor.
ζ	Threshold to detect average reward differences.
ξ	Threshold of average reward before exploration can be stopped.
$F_{Y_t}(i)$	Probability mass function associated with arm i.
I_t	Selected arm at time t.
$\mathcal{K}$	The set of arms available in the MAB.
$\mathcal{N}(\mu, \sigma)$	Normal distribution centred at μ with standard deviation σ.

$N_t(i)$	Discounted number of pulls applied to arm i.
P	Time steps between reward distribution change-points (for evaluation purposes).
R_t	Modulatory variable used in the reset functionality.
t, T	Time step, time horizon.
$U(a, b)$	Uniform distribution between a and b.
$U_t(i)$	Upper confidence bound of arm i.
$X_t(i)$	Reward collected as a result of pulling arm i.
$\bar{X}_t(i)$	Discounted empirical average reward of arm i.
w	Length of vector containing a contiguous series of w rewards.

Abbreviations

The following abbreviations are used in this manuscript:

BGP	Border Gateway Protocol
OSPF	Open Shortest Path First
RL	Reinforcement Learning
KL	Kullback–Leibler
MAB	Multi-Armed Bandit
SDN	Software-Defined Networking
UCB	Upper Confidence Bound
B	Baseline MAB Agent Configuration
R	Reset MAB Agent Configuration
C	MAB Agent Configuration using Correlated Arms
T	MAB Agent Configuration using a Threshold for the Highest Reward

References

1. Moura, J.; Hutchison, D. Modeling cooperative behavior for resilience in cyber-physical systems using SDN and NFV. *SN Appl. Sci.* **2020**, *2*, 1–13. [CrossRef]
2. Amin, R.; Rojas, E.; Aqdus, A.; Ramzan, S.; Casillas-Perez, D.; Arco, J.M. A Survey on Machine Learning Techniques for Routing Optimization in SDN. *IEEE Access* **2021**, *9*, 104582–104611. [CrossRef]
3. Boutaba, R.; Salahuddin, M.A.; Limam, N.; Ayoubi, S.; Shahriar, N.; Estrada-Solano, F.; Caicedo, O.M. A comprehensive survey on machine learning for networking: Evolution, applications and research opportunities. *J. Internet Serv. Appl.* **2018**, *9*, 16. [CrossRef]
4. Chen, M.; Challita, U.; Saad, W.; Yin, C.; Debbah, M. Artificial Neural Networks-Based Machine Learning for Wireless Networks: A Tutorial. *IEEE Commun. Surv. Tutorials* **2019**, *21*, 3039–3071. [CrossRef]
5. Hussain, F.; Hassan, S.A.; Hussain, R.; Hossain, E. Machine Learning for Resource Management in Cellular and IoT Networks: Potentials, Current Solutions, and Open Challenges. *IEEE Commun. Surv. Tutorials* **2020**, *22*, 1251–1275. [CrossRef]
6. Sutton, R.S.; Barto, A.G. *Reinforcement Learning: An Introduction*; MIT Press: Cambridge, MA, USA, 2018.
7. Slivkins, A. Introduction to multi-armed bandits. *Found. Trends® Mach. Learn.* **2019**, *12*, 1–286. [CrossRef]
8. Lattimore, T.; Szepesvári, C. *Bandit Algorithms*; Cambridge University Press: Cambridge, UK, 2020.
9. Bouneffouf, D.; Rish, I.; Aggarwal, C. Survey on applications of multi-armed and contextual bandits. In Proceedings of the 2020 IEEE Congress on Evolutionary Computation (CEC), Glasgow, UK, 19–24 July 2020; IEEE: New York, NY, USA, 2020; pp. 1–8.
10. Mariano, P.; Almeida, S.M.; Santana, P. On the automated learning of air pollution prediction models from data collected by mobile sensor networks. In *Energy Sources, Part A: Recovery, Utilization, and Environmental Effects*; Taylor and Francis: New York, NY, USA, 2021; pp. 1–17.
11. Tang, F.; Mao, B.; Kawamoto, Y.; Kato, N. Survey on machine learning for intelligent end-to-end communication toward 6G: From network access, routing to traffic control and streaming adaption. *IEEE Commun. Surv. Tutorials* **2021**, *23*, 1578–1598. [CrossRef]
12. Tariq, Z.U.A.; Baccour, E.; Erbad, A.; Guizani, M.; Hamdi, M. Network Intrusion Detection for Smart Infrastructure using Multi-armed Bandit based Reinforcement Learning in Adversarial Environment. In Proceedings of the 2022 International Conference on Cyber Warfare and Security (ICCWS), Warsaw, Poland, 18–20 October 2022; IEEE: New York, NY, USA, 2022; pp. 75–82.
13. Heartfield, R.; Loukas, G.; Bezemskij, A.; Panaousis, E. Self-configurable cyber-physical intrusion detection for smart homes using reinforcement learning. *IEEE Trans. Inf. Forensics Secur.* **2020**, *16*, 1720–1735. [CrossRef]
14. Henri, S.; Vlachou, C.; Thiran, P. Multi-armed bandit in action: Optimizing performance in dynamic hybrid networks. *IEEE/ACM Trans. Netw.* **2018**, *26*, 1879–1892. [CrossRef]
15. Huang, X.; Tang, Y.; Shao, Z.; Yang, Y.; Xu, H. Joint Switch–Controller Association and Control Devolution for SDN Systems: An Integrated Online Perspective of Control and Learning. *IEEE Trans. Netw. Serv. Manag.* **2020**, *18*, 315–330. [CrossRef]
16. Rischke, J.; Sossalla, P.; Salah, H.; Fitzek, F.H.; Reisslein, M. QR-SDN: Towards reinforcement learning states, actions, and rewards for direct flow routing in software-defined networks. *IEEE Access* **2020**, *8*, 174773–174791. [CrossRef]

17. Casas-Velasco, D.M.; Rendon, O.M.C.; da Fonseca, N.L.S. DRSIR: A Deep Reinforcement Learning Approach for Routing in Software-Defined Networking. *IEEE Trans. Netw. Serv. Manag.* **2022**, *19*, 4807–4820. [CrossRef]
18. Auer, P.; Cesa-Bianchi, N.; Fischer, P. Finite-time analysis of the multiarmed bandit problem. *Mach. Learn.* **2002**, *47*, 235–256. [CrossRef]
19. Auer, P.; Cesa-Bianchi, N.; Freund, Y.; Schapire, R.E. The nonstochastic multiarmed bandit problem. *SIAM J. Comput.* **2002**, *32*, 48–77. [CrossRef]
20. Garivier, A.; Cappé, O. The KL-UCB algorithm for bounded stochastic bandits and beyond. In Proceedings of the 24th Annual Conference on Learning Theory, Budapest, Hungary, 9–11 June 2011; pp. 359–376.
21. Kocsis, L.; Szepesvári, C. Discounted ucb. In Proceedings of the 2nd PASCAL Challenges Workshop, Graz, Austria, 10–12 April 2006; Volume 2, pp. 51–134.
22. Garivier, A.; Moulines, E. On upper-confidence bound policies for switching bandit problems. In Proceedings of the 22nd International Conference on Algorithmic Learning Theory (ALT), Espoo, Finland, 5–7 October 2011. Proceedings 22; Springer: Cham, Switzerland, 2011; pp. 174–188.
23. Cao, Y.; Wen, Z.; Kveton, B.; Xie, Y. Nearly optimal adaptive procedure with change detection for piecewise-stationary bandit. In Proceedings of the 22nd International Conference on Artificial Intelligence and Statistics, PMLR, Okinawa, Japan, 16–18 April 2019; pp. 418–427.
24. Gupta, S.; Chaudhari, S.; Joshi, G.; Yağan, O. Multi-armed bandits with correlated arms. *IEEE Trans. Inf. Theory* **2021**, *67*, 6711–6732. [CrossRef]
25. Network Analysis. Available online: https://networkx.org/ (accessed on 25 February 2023).
26. Cohen, B. *Statistical Power Analysis for the Behavioral Sciences*, 2nd ed.; Lawrence Erlbaum Associates: Mahwah, NJ, USA, 1988.
27. Thompson, W.R. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika* **1933**, *25*, 285–294. [CrossRef]
28. SMPyBandits. Available online: https://pypi.org/project/SMPyBandits/ (accessed on 25 February 2023).
29. Mininet. Available online: https://github.com/mininet/mininet (accessed on 25 February 2023).
30. Ryu. Available online: https://github.com/faucetsdn/ryu (accessed on 25 February 2023).
31. Abbasi-Yadkori, Y.; Pál, D.; Szepesvári, C. Improved algorithms for linear stochastic bandits. *Adv. Neural Inf. Process. Syst.* **2011**, *24*, 2312–2320.
32. Cardoso, P.; Moura, J.; Marinheiro, R.N. Elastic Provisioning of Network and Computing Resources at the Edge for IoT Services. *Sensors* **2023**, *23*, 2762. . [CrossRef] [PubMed]
33. Cardoso, P.; Moura, J.; Marinheiro, R. Software-Defined Elastic Provisioning of IoT Edge Computing Virtual Resources. *CoRR* 2023. Available online: https://arxiv.org/abs/2003.11999 (accessed on 2 April 2023).