Resource management in distributed SDN using reinforcement learning

Liang Ma, Ziyao Zhang, Bongjun Ko, Mudhakar Srivatsa, Kin K Leung

Liang Ma, Ziyao Zhang, Bongjun Ko, Mudhakar Srivatsa, Kin K Leung, "Resource management in distributed SDN using reinforcement learning," Proc. SPIE 10635, Ground/Air Multisensor Interoperability, Integration, and Networking for Persistent ISR IX, 106350M (4 May 2018); doi: 10.1117/12.2306087

SPIE.

Event: SPIE Defense + Security, 2018, Orlando, Florida, United States

Resource Management in Distributed SDN Using Reinforcement Learning

Liang Ma[a], Ziyao Zhang[b], Bongjun Ko[a], Mudhakar Srivatsa[a], and Kin K. Leung[b]

[a]IBM T. J. Watson Research Center, 1101 Kitchawan Road, Yorktown Heights, NY 10598, United States

[b]Imperial College London, Exhibition Road, London SW7 2AZ, United Kingdom

ABSTRACT

Distributed software-defined networking (SDN), which consists of multiple inter-connected network domains and each managed by one SDN controller, is an emerging networking architecture that offers balanced centralized/distributed control. Under such networking paradigm, resource management among various domains (e.g., optimal resource allocation) can be extremely challenging. This is because many tasks posted to the network require resources (e.g., CPU, memory, bandwidth, etc.) from different domains, where cross-domain resources are correlated, e.g., their feasibility depends on the existence of a reliable communication channel connecting them. To address this issue, we employ the reinforcement learning framework, targeting to automate this resource management and allocation process by proactive learning and interactions. Specifically, we model this issue as an MDP (Markov Decision Process) problem with different types of reward functions, where our objective is to minimize the average task completion time. Regarding this problem, we investigate the scenario where the resource status among controllers is fully synchronized. Under such scenario, the SDN controller has complete knowledge of the resource status of all domains, i.e., resource changes upon any policies are directly observable by controllers, for which Q-learning-based strategy is proposed to approach the optimal solution.

Keywords: Resource Management, SDN, Task Fragmentation, Reinforcement Learning, Q-learning

1. INTRODUCTION

Software-Defined Networking (SDN),[1–4] an emerging networking architecture, significantly improves the network performance due to its programmable network management, easy reconfiguration, and on-demand resource allocation, which has therefore attracted considerable research interests.

One key attribute that differentiates SDN from classic networks is the separation of the SDN's data and control plane. Specifically, in SDN, all control functionalities are implemented and abstracted in the *SDN controller*, which sits in the control plane, for operational decision making, while the data plane only passively executes the instructions received from the control plane. Since the centralized SDN controller has full knowledge of the network status, it is able to make the global optimal decision.

In tactical networks, coalition partners may deploy network devices with heterogeneous computing, storage, and communication capabilities. To enable flexible, effective, and robust services in coalition networks, SDN-based network architecture has been proposed to address these coalition needs with dynamic configurability. However, existing SDN techniques rely on centralized control and synchronization protocol design; such centralized control suffers from major scalability issues. In particular, as a network grows, the number of service requests and operational constraints are likely to increase exponentially. Such high computation and communication requirements may impose substantial burden on the SDN controller, thus potentially resulting in significant performance degradation (e.g., delays) or even network failures.

Further author information: (Send correspondence to L. Ma and Z. Zhang)
L. Ma: E-mail: maliang@us.ibm.com, Telephone: +1 914 945 1247
Z. Zhang: E-mail: ziyao.zhang15@imperial.ac.uk, Telephone: +44 (0) 20 7594 6267
B. Ko: E-mail: bongjun_ko@us.ibm.com
M. Srivatsa: E-mail: msrivats@us.ibm.com
K. Leung: E-mail: kin.leung@imperial.ac.uk

Ground/Air Multisensor Interoperability, Integration, and Networking for Persistent ISR IX, edited by Michael A. Kolodny, Dietrich M. Wiegmann, Tien Pham, Proc. of SPIE Vol. 10635, 106350M · © 2018 SPIE · CCC code: 0277-786X/18/$18 · doi: 10.1117/12.2306087

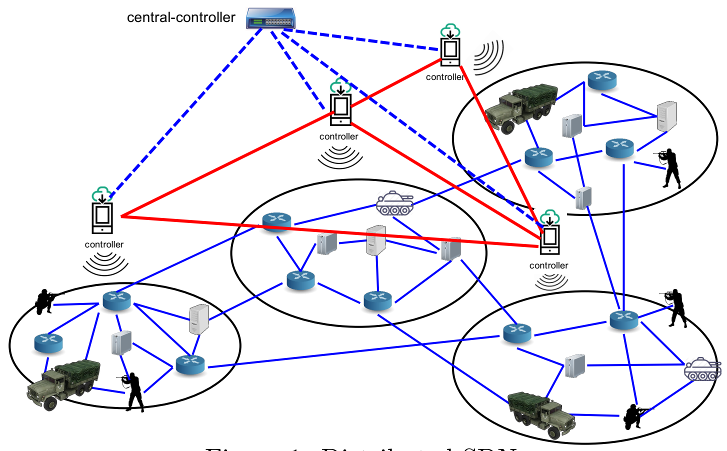

Figure 1: Distributed SDN.

In this regard, in military network scenario which generally exhibits high mobility and QoS (quality of service) fluctuations, SDN is implemented in an distributed manner. In particular, distributed SDN is composed of multiple domains (or enclaves), where each domain represents a connected sub-network of a coalition partner. Each domain contains one logical enclave controller (which may correspond to multiple physical controllers) and connects to the controllers in other domains or the central-controller (which is also a logical entity where its functionalities can be distributed and co-located at the enclave controllers), such as the central commander in the network, depending on how the network architecture is designed. These controllers, which are collectively referred as the SDN controllers, coordinate to utilize all infrastructure resources to achieve mission objectives and satisfy coalition policies. Note that multiple distributed SDN networks can be defined in the same coalition physical infrastructure. As an example, Fig. 1 illustrates an SDN-based military network, which consists of four inter-connected domains. Furthermore, each domain contains one domain-controller connecting to the central-controller; all domain-controllers together with the central-controller form the control plane in distributed SDN.

Under such distributed SDN architecture, when the control plane receives a request, the control plane needs to decide where to dispatch this request in the most efficient way. For instance, the request posted to the control plane may require a certain amount of computation and storage resources. Moreover, it is possible that no single domain could meet the resource requirement for the given task; nevertheless, multiple domains can work together to jointly provide various types of resources to complete the posted task. The challenge in this problem lies in how to make an optimized decision for resource management across different domains by taking complicated and correlated network conditions and resource requests into account. Specifically, the requests may arrive in a dynamic manner with different priorities and different resource requirement. For this case, the control plane needs to make an online optimization to efficiently utilize the network resources in different domains so that the service quality averaged over time is maximized. On this other hand, a task might be regarded as a combination of sub-tasks, which therefore can be processed in different domains in parallel; thus, this imposes more burden on the control plane as the total number of task fragmentation methods is exponential in the total number of sub-tasks in each task. Therefore, such resource management problem is highly complicated in the military distributed SDN scenario.

To address this problem, we propose *reinforcement-resource-management*, a reinforcement-learning-based algorithm with performance and convergence rate guarantees, to optimally decide which tasks received by the control plane should be processed in the next time slot, and how these tasks should be fragmented for parallel processing. Here, the goal of the reinforcement-resource-management is to minimize the average task slowdown for all tasks (that are sequentially received by the control plane). Comparing to traditional approaches, our method exhibits significant advantages. First, we do not have any specific assumptions or requirements on the network conditions or the resource requirement. In contrast, traditional approaches generally impose restrictions, mostly on the network (e.g., network structures and size) or on requests, so as to simplify the algorithm development task. Second, the decision-making process of reinforcement-resource-management is *on-demand* and *pay-as-you-go*. Unlike traditional optimization algorithms with fixed optimization levels, reinforcement-resource-management lets one decide the optimization level. This is achieved by tuning the number of iterations the algorithm runs. It is a desirable feature because the network commander has the ability to decide exactly how much computation power he/she want to consume in order to achieve certain performance level. This flexibility is important especially in situations where the network commander is overloaded. Third, the reinforcement-resource-management algorithm is fully compatible with the SDN architecture, i.e., the existing SDN infrastructure and protocol suite are *off-the-shelf* for its implementation. In fact, the reinforcement-resource-management algorithm itself can be regarded as an application running on the central SDN controller. All information needed by reinforcement-resource-management are collected during routine network status updates between the central controller and and the domain controllers through southbound interface (e.g., OpenFlow[5]). More importantly, reinforcement-resource-management is adjustable to network dynamics, e.g., changing resource status and requirements. These appealing features make reinforcement-resource-management suitable to our resource management problem in the distributed SDN environment.

Military operations can significantly benefit from the efficient, agile, optimal resource management strategy in the distributed SDN structure. Specifically, using reinforcement-resource-management, we can automate many aspects of war fighters' tasks in setting up and operating an SDN infrastructure, improving agility, reducing misconfigurations, and avoiding security risks, in complex, contested settings, when making relevant information, data, and resources available at the point of need.

1.1 Related Work

As the development of machine learning techniques, applying reinforcement learning to solve networking problem[6] has demonstrated strong potential. In particular, when there is only one agent (for decision making), authors[7] leverage the deep reinforcement learning framework to deal with the traffic engineering problem by formulating the network reward as a NUM (Network Utility Maximization) function. Further, a specific routing issue is addressed in works[8] using a data-driven approaches. When there are multiple agents, researchers[9] investigate radio power allocation in mobile vehicular networks from the perspective of physical layer techniques. The work that is most related to ours is the research work[10] in that it also studies a resource management problem. However, the problem studied in paper[10] essentially is a scheduling problem in a single network domain without task fragmentations. In contrast, in this paper, we study a multi-domain resource management problem, where we are allowed to partition a task into independent sub-tasks for parallel processing in different domains; therefore, our problem encounters a larger action space, thus more challenging.

Another thread of research that is related to ours is in reducing unnecessary SDN controller involvements in situations where regular network elements are good enough for handling user requests. In this way, the controller can focus on important decision-making tasks that require high-level abstractions and centralized network-wide views. As such, regular nodes may take over most routine tasks and operations that require only local information. For example, Kandoo[11] is proposed to enable two types of applications to coexist, i.e., locally scoped applications and those require network-wide states. Moreover, there are also proposals[12,13] suggesting that regular nodes (SDN switches) should be equipped with general purpose CPUs to allow for greater flexibility and on-switch processing. Along this line, the works[14,15] propose the idea of content router, which is similar to the concept of SDN switches. Similarly, works[16,17] propose different methods, with different focuses, to implement content routers. However, the problem formulation in most of these works is too specific, thus not fully taking advantage of the new features introduced by distributed SDN.

The rest of the paper is organized as follows. We describe how the system works in Section 2. Section 3 formally formulates the problem. Section 4 describes the details of the reinforcement-resource-management. Finally, we conclude the paper in Section 5.

2. RESOURCE MANAGEMENT IN DISTRIBUTED SDN

In this section, we describe how the distributed SDN operates in resource management, key features of which are captured and abstracted in Section 3.

2.1 Network Architecture

In this paper, we consider a hierarchical SDN network architecture. Specifically, as aforementioned, each domain has one SDN domain-controller. Moreover, there exists a central-controller that connects to each of the domain-controllers (via a physical or a logical link)[*]; see Fig. 1.

Naturally, the SDN central-controller and the domain-controllers are the most essential infrastructures in the resource management problem.

2.2 Central-Controller

We assume all resource requests arrive at the central-controller first, forming a request queue. Since the central-controller connects to each domain-controller, the central-controller is able to collect the resource status (e.g., availability) in each domain directly from the domain-controllers; therefore, the controller has the up-to-date global view of the network. The central-controller then employs the current status of the network resources as well as the request queue to decide which requests (tasks) should be selected to dispatch to different domains for the required services. The decision-making algorithms are essentially high-level applications offered by the central-controller, which is programmable by the network administrator.

2.3 Domain-Controller

The domain-controllers are responsible for assisting the central-controller in collecting relevant resource information in their domains. The collected information is then reported to the central-controller during the routine update process. The domain-controllers only passively execute the decision made by the central-controller, i.e., provide the resource/service to the allocated tasks/sub-tasks. In practice, when executing the service request from the central-controller, the domain-controller further decides which network elements with available resources in its domain could be selected to provide the required services; nevertheless, such intra-domain decision making is transparent to the central-controller, and out of the scope of this paper.

3. PROBLEM FORMULATION

We formulate the resource management problem as a discrete Markov Decision Process (MDP), for which we propose (in Section 4) an algorithm with performance guarantees. In particular, our MDP is defined by a 4-tuple (S, A, P, r) in the discrete time horizon, detailed as below:

- S is the finite state space. In our problem, we assume there n types of resources that are distributed across the entire network in m domains. In other words, each domain has up to n resource types. We assume that a state $s \in S$ corresponds to the resource availability of all types of resources in each domain and the queueing status of the central-controller at the current time slot.
- A is the finite action space. We assume that for any request μ, our system is sufficient to provide all required resources when the system is idle, i.e., the system is providing services to any other requests. Furthermore, request μ_i can be fragmented into up to k_i sub-requests. Therefore, when request μ_i in the queue is picked by the central-controller to provide services, there are up to m^{k_i} possible cases to allocate μ_i to the m domains. An action $a \in A$ is to decide which requests are selected from the queue and how there are fragmented and allocated to different domains. Therefore, when the number of requests in the queue is large, the size of the action space can be significantly large.
- P is the state transition probability matrix, which is a function of the state-action pair (s, a), where $s \in S$ and $a \in A$. In particular, when certain requests are removed from the queue in the central-controller and the new requests arrive to the queue, the system enters a new state s' ($s' \in S$).

[*]The central-controller controls domain-controllers in an in-band or out-of-band manner. In-band control means that control traffic is communicated using existing data plane links, whereas out-of-band control means that the central-controller has dedicated direct control links to domain-controllers.[18]

- r represents the immediate reward function associated with state-action pairs, which we define as $r(s,a)$, where $s \in \mathcal{S}$ and $a \in \mathcal{A}$. There are multiple ways to define $r(s,a)$; the principal is to consider both the request difficulty and completion time so that the average task slowdown over all requests is minimized. We will provide the potential definitions of $r(s,a)$ in Section 3.1.

At the start of the MDP, suppose we are at the initial state s_0 (we do not require the system be completely idle or queue be empty). The MDP then enters a different state after each state-action pair. Each state transition generates a non-negative reward. The *optimal action* at each state, denoted by π^*, is defined as the action that gives the maximum long-term reward, which is the discounted sum of the expected immediate reward of all future state-action pairs from the current state. The reward for the state-action pair Δt steps ahead of the current state is discounted by $\gamma^{\Delta t}$, where γ is called the *discount factor* and $0 < \gamma < 1$. Here, γ trades off the importance between the current and the future reward. Therefore, starting from an initial state s_0, the problem is formulated to maximize the long-term accumulated reward expressed in the following Bellman equation by selecting a sequence of actions $\pi = \{a_t\}_{t=0}^{\infty}$:

$$V_\pi(s_0) = \mathbb{E}\big[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)\big], \tag{1}$$

where s_t and a_t constitute the state-action pair at time t, and

$$\pi^* = \arg\max_{\pi} \mathbb{E}\big[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)\big]. \tag{2}$$

3.1 Reward Function

The goal of the resource management problem is to minimize the average task slowdown. Therefore, we need to have a measure to quantify the benefit of an action at a state, i.e., the reward function $r(s,a)$. However, the definition of $r(s,a)$ is not unique. In this regard, we provide several candidate functions for $r(s,a)$.

Let τ_i denote the minimum number of time slots that are required by task μ_i with k_i sub-tasks, i.e., τ_i is achieved when the entire system is completely idle when μ_i is received. Furthermore, let c_{ij} be the duration starting from the time slot that μ_i is received in the system and ending at the time slot that subtask j in μ_i is completed. Therefore, the task slowdown w.r.t. μ_i is defined as $c_i := \max_j c_{ij}$. Suppose all scheduled tasks (which are removed from the queue and dispatched to different domains) at time t constitute set $\mathcal{F}$, then we define four candidate functions for $r(s,a)$.

- $r(s,a) = \sum_{i \in \mathcal{F}} (-c_i/\tau_i)$. In this reward function, essentially we want to eliminate the bias where requests with small τ are selected with high probability. Since τ indicates the request difficulty, the intuitive interpretation of this reward function is that a request with higher difficulty (i.e., larger τ) incurs larger reward, and larger task slowdown decreases the reward.

- $r(s,a) = \sum_{i \in \mathcal{F}} \gamma^{c_i - \tau_i}$. Similar to the above reward function, we again capture how a reward should be related to the state-action pair. Meanwhile, we leverage the discount factor γ to further capture the time sensitivity, as our objective is to minimize the average slowdown.

- $r(s,a) = \sum_{i \in \mathcal{F}} (\tau_i \gamma^{c_i})$. This reward function is proposed with the consideration that c_i is the dominant factor.

- $r(s,a) = \sum_{i \in \mathcal{F}} (-c_i \gamma^{\tau_i})$. This reward function is proposed with the consideration that τ_i is the dominant factor.

The above candidate reward functions can be selected based on specific optimization and convergence objectives.

4. REINFORCEMENT-RESOURCE-MANAGEMENT

The *reinforcement-resource-management* algorithm is based on the classic model-free reinforcement learning technique Q-*learning*,[19] which is used to find the optimal state-action policy for any MDP. Q-learning is proved to be optimal under certain conditions. The idea of Q-learning is that an agent can move from state to state by taking certain actions. The agent attempts to maximize its accumulated reward by applying different state-action pairs and learn the optimal action at each state.

Q-placement uses the Q-*function* to estimate the quality of a state-action combination:

$$S \times A \to \mathbb{R}.$$

In particular, the optimal Q-function for a state-action pair in Q-placement is defined as:

$$Q^*(s,a) = \mathbb{E}[r(s,a) + \gamma \max_{a' \in A} Q^*(s',a')], \tag{3}$$

where s' and a' is a state-action pair at the next time instant. All learned values from the Q-function constitute the Q-*matrix*. Based on the Q-matrix, we know that the optimal value of (1) is $V^*(s_0) = \max_{a \in A} Q^*(s_0, a)$. At the start of the algorithm, the Q-matrix is initialized with all 0 entries. Therefore, the agent's Q-function, denoted as $Q(s,a)$, can be substantially different from the optimal Q-function at the beginning, due to the lack of experiences. However, the *value iteration update* of the Q-placement algorithm makes $Q(s,a) \to Q^*(s,a)$ when $t \to \infty$. In particular, as the learning process proceeds, the Q-matrix is updated using the following rule:

$$Q(s,a) = (1-\alpha)Q(s,a) + \alpha[R(s,a) + \gamma \max_{a' \in A} Q(s',a')], \tag{4}$$

where α $(0 < \alpha < 1)$ is the learning rate of Q-placement.

The procedures of the Q-placement algorithm is summarized in Algorithm 1. An important element in Algorithm 1 is the *exploration policy*. In simple terms, an exploration policy tells the agent how to choose an action given a state. At the beginning of Q-placement, the Q-matrix is sparse. Nevertheless, the agent's experiences build up over time, as manifested by the growing number of non-zero Q-values in Q-matrix. At each state, the agent decides the action either by choosing randomly from all available actions, or by certain strategy that exploits past experiences when positive Q-value exists for some state-action pairs.

Specifically, for each state, the agent decides how to exploit the actions using the ϵ-*greedy algorithm*,[20] which makes sure that the agent makes sufficient explorations and offers rapid convergence rate (see line 5). With ϵ-greedy exploration policy, given a state, the agent chooses the action with the maximum Q-value with probability $1-\epsilon$, and chooses a random action with probability ϵ, where ϵ is small. As the name suggests, this exploration policy is greedy in the sense that it tries, with high probability, to exploit actions with high known rewards to take advantage of the *learned experiences*.

As the Q-placement algorithm proceeds, all entries in Q-matrix eventually approach stable values. This happens when the algorithm converges. Then regarding the output Q-matrix, given state s, the best action to take is $a^* = \arg\max_a Q(s,a)$. In fact, we are able to tell how close the selected actions are to the optimality and how fast these are achieved, as stated below.

Optimality: One advantage of using Q-learning for solving MDP is that there exists provable optimality guarantee. Specifically, it has been proved[19] that the update rule (i.e., (4)) in Q-learning converges with probability 1 to the optimal Q-function w.r.t. the Bellman equation, as long as all state-actions pairs are visited infinitely often. In the reinforcement-resource-management algorithm, the ϵ-greedy exploration strategy is a justified exploration policy that meets this optimality requirement. Therefore, we have the assurance that reinforcement-resource-management eventually converges and the Bellman equation is maximized.

Rate of convergence: We now discuss the convergence rate of the Q-placement algorithm. The authors[21] proved that only on the order of $(N\log(1/\xi)/\xi^2)(\log(N) + \log\log(1/\xi))$ iterations are sufficient for Q-learning to come with ξ of the optimal policy, where N is the number of states of the MDP. This guarantees that the reinforcement-resource-management algorithm experiences a rather rapid convergence to optimality.

```
Algorithm 1: reinforcement-resource-management
  input  : Initial state $s_0$ and discount factor $\gamma$.
  output: $Q$-matrix.
1 Determine the number of iterations $T$ (time instants);
2 $t = 0$;
3 Initialize $Q$-matrix with all 0 entries;
4 foreach $t = 0, 1, \ldots, T$ do
5     Determine action $a \in A$ according to $\epsilon$-greedy algorithm;
6     Update $Q(s, a)$ according to (4);
7 end
```

5. CONCLUSIONS

We studied the resource management problem in a distributed SDN environment aiming at optimizing the resource utilization and minimizing the average task slowdown. We formulated the problem as an MDP problem with a set of candidate reward functions. For this issue, we proposed a reinforcement-learning-based algorithm with tunable parameters and performance guarantees.

ACKNOWLEDGMENTS

This research was sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defence under Agreement Number W911NF-16-3-0001. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defence or the U.K. Government. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

REFERENCES

[1] McKeown, N., "Software-defined networks and the maturing of the Internet." [Online]. Available: https://tv.theiet.org/?videoid=5447

[2] Kreutz, D., Ramos, F. M., Verissimo, P. E., Rothenberg, C. E., Azodolmolky, S., and Uhlig, S., "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE,* **103**(1), 14–76 (2015).

[3] Nunes, B. A. A., Mendonca, M., Nguyen, X.-N., Obraczka, K., and Turletti, T., "A survey of software-defined networking: Past, present, and future of programmable networks," *IEEE Communications Surveys & Tutorials,* **16**(3), 1617–1634 (2014).

[4] Casado, M., Freedman, M. J., Pettit, J., Luo, J., McKeown, N., and Shenker, S., "Ethane: Taking control of the enterprise," *ACM SIGCOMM Computer Communication Review,* **37**(4), 1–12 (2007).

[5] McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., and Turner, J., "Openflow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review,* **38**(2), 69–74 (2008).

[6] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D., "Human-level control through deep reinforcement learning," *Nature,* **518**(7540), 529–533 (2015).

[7] Xu, Z., Tang, J., Meng, J., Zhang, W., Wang, Y., Liu, C. H., and Yang, D., "Experience-driven networking: A deep reinforcement learning based approach," *IEEE INFOCOM* (2018).

[8] Valadarsky, A., Schapira, M., Shahaf, D., and Tamar, A., "A machine learning approach to routing," *CoRR,* **abs/1708.03074** (2017).

[9] Ye, H. and Li, G. Y., "Deep reinforcement learning for resource allocation in V2V communications," *CoRR,* **abs/1711.00968** (2017).

[10] Mao, H., Alizadeh, M., Menache, I., and Kandula, S., "Resource management with deep reinforcement learning," *ACM HotNets* (2016).

[11] Hassas Yeganeh, S. and Ganjali, Y., "Kandoo: A framework for efficient and scalable offloading of control applications," *ACM HotSDN* (2012).

[12] Lu, G., Miao, R., Xiong, Y., and Guo, C., "Using CPU as a traffic co-processing unit in commodity switches," *ACM HotSDN* (2012).

[13] Mogul, J. C. and Congdon, P., "Hey, you darned counters!: get off my asic!," *ACM HotSDN,* 25–30 (2012).

[14] Wong, W., Giraldi, M., Magalhaes, M. F., and Kangasharju, J., "Content routers: Fetching data on network path," *IEEE ICC* (2011).

[15] Wong, W., Wang, L., and Kangasharju, J., "Neighborhood search and admission control in cooperative caching networks," *IEEE GLOBECOM* (2012).

[16] Ming, Z., Xu, M., and Wang, D., "Age-based cooperative caching in information-centric networking," *IEEE ICCCN* (2014).

[17] Li, Z. and Simon, G., "Time-shifted tv in content centric networks: The case for cooperative in-network caching," *IEEE ICC* (2011).

[18] Soltani, A. and Bazlamacci, C. F., "Hyfi: Hybrid flow initiation in software defined networks," *IEEE ICICS* (2014).

[19] Watkins, C. J. and Dayan, P., "Q-learning," *Machine learning,* **8**(3-4), 279–292 (1992).

[20] Thrun, S. B., "Efficient exploration in reinforcement learning," tech. rep. (1992).

[21] Kearns, M. J. and Singh, S. P., "Finite-sample convergence rates for Q-learning and indirect algorithms," *Advances in Neural Information Processing Systems*, 996–1002 (1999).