Optimal Network Control in Partially-Controllable Networks

Optimal Network Control in Partially-Controllable Networks

Qingkai Liang and Eytan Modiano
LIDS, MIT

***Abstract*—The effectiveness of many optimal network control algorithms (e.g., BackPressure) relies on the premise that all of the nodes are fully controllable. However, these algorithms may yield poor performance in a partially-controllable network where a subset of nodes are uncontrollable and use some unknown policy. Such a partially-controllable model is of increasing importance in real-world networked systems such as overlay-underlay networks. In this paper, we design optimal network control algorithms that can stabilize a partially-controllable network. We first study the scenario where uncontrollable nodes use a queue-agnostic policy, and propose a low-complexity throughput-optimal algorithm, called Tracking-MaxWeight (TMW), which enhances the original MaxWeight algorithm with an explicit learning of the policy used by uncontrollable nodes. Next, we investigate the scenario where uncontrollable nodes use a queue-dependent policy and the problem is formulated as an MDP with unknown queueing dynamics. We propose a new reinforcement learning algorithm, called Truncated Upper Confidence Reinforcement Learning (TUCRL), and prove that TUCRL achieves tunable three-way tradeoffs between throughput, delay and convergence rate.**

I. Introduction

Optimal network control has been an active area of research for more than thirty years, and many efficient routing algorithms have been developed over the past few decades, such as the well-known throughput-optimal BackPressure routing algorithm [26]. The effectiveness of these algorithms usually relies on the premise that all of the nodes in a network are fully controllable. Unfortunately, an increasing number of real-world networked systems are only *partially controllable*, where a subset of nodes are not managed by the network operator and use some unknown network control policy, such as overlay-underlay networks.

An overlay-underlay network consists of overlay nodes and underlay nodes [3], [16], [19]. The overlay nodes can implement state-of-the-art algorithms while the underlay nodes are uncontrollable and use some unknown protocols (e.g., legacy protocols). Figure 1 shows an overlay-underlay network where the communications among overlay nodes rely on the uncontrollable underlay nodes. Overlay networks have been used to improve the capabilities of computer networks for a long time (e.g., content delivery [21]).

Due to the unknown behavior of uncontrollable nodes, the existing routing algorithms may yield poor performance in a partially-controllable network. For example, Figure 2 shows an example where the well-known Backpressure routing algorithm [26] fails to deliver the maximum throughput when some nodes are uncontrollable. In particular, uncontrollable node 3 adopts a policy that does not preserve the flow conservation law such that its backlog builds up, but uncontrollable node 2 hides this backlog information from node 1. As a result, if node 1 uses Backpressure routing, it always transmits packets to node 2, although these packets will never be delivered. A smarter algorithm should be able to learn the behavior of the uncontrollable nodes such that node 1 only sends packets along route $1 \to 5 \to 4$.

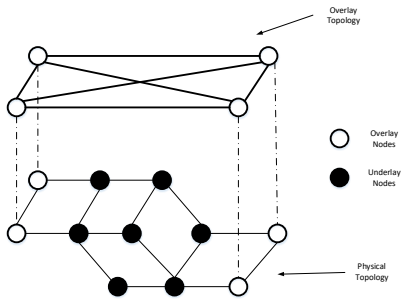


Fig. 1. An example of overlay-underlay networks.

As a result, it is important to develop new network control algorithms that achieve consistently good performance in a partially-controllable environment. In this paper, we study efficient network control algorithms that can stabilize a partially-controllable network whenever possible. In particular, we consider two scenarios.

First, we investigate the scenario where uncontrollable nodes use a *queue-agnostic* policy, which captures a wide range of practical network protocols, such as shortest path routing (e.g., OSPF, RIP), multi-path routing (e.g., ECMP) and randomized routing algorithms. In this scenario, we propose a low-complexity throughput-optimal algorithm, called Tracking-MaxWeight (TMW), which enhances the original MaxWeight algorithm [26] with an explicit learning of the policy used by uncontrollable nodes.

Second, we study the scenario where uncontrollable nodes use a *queue-dependent* policy, i.e., the action taken by uncontrollable nodes relies on the observed queue length vector (e.g., Backpressure routing). In this scenario, we show that the queueing dynamics become unknown and no longer follow the classic Lindley recursion [7], which makes the problem fundamentally different from the traditional network optimization framework: we not only need to know

This work was supported by NSF Grant CNS-1524317 and by DARPA I2O and Raytheon BBN Technologies under Contract No. HROO 1 l-l 5-C-0097.

how to perform optimal network control but also need to learn the queueing dynamics in an efficient way. We formulate the problem as a Markov Decision Process (MDP) with unknown dynamics, and propose a new reinforcement learning algorithm, called Truncated Upper Confidence Reinforcement Learning (TUCRL), that is shown to achieve network stability under mild conditions.

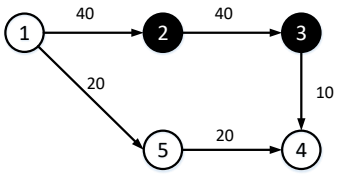


Fig. 2. Counterexample where the well-known Backpressure routing algorithm fails to deliver the maximum throughput in a partially-controllable network. The number next to each link is its capacity. Each node can transmit only to one of its neighbors in each time slot. There is only one flow: $1 \rightarrow 4$ (at rate 20). Black nodes are uncontrollable nodes. Node 2 transmits any packet it received to node 3 at full rate, so that its queue length is always zero; node 3 adopts a non-work-conserving policy that holds any packet it received. When node 1 uses Backpressure routing, it always transmits packets to node 2 since its queue length is always zero, which hides the fact that backlog builds up at node 3.

A. Related Work

Most of the existing works on network optimization in a partially-controllable environment are in the context of overlay-underlay networks where the underlay nodes are not controllable and may adopt arbitrary (unknown) policies. The objective is to find efficient control policies for the controllable overlay nodes in order to optimize certain performance metrics (e.g., throughput). In [3], the authors showed that the well-known BackPressure algorithm [26], which was shown to be throughput-optimal in a wide range of scenarios, may lead to a loss in throughput when used in an overlay-underlay setting, and proposed a heuristic routing algorithm for overlay nodes called Overlay Backpressure Policy (OBP). An optimal backpressure-type routing algorithm for a special case, where the underlay paths do not overlap with each other, was given in [16]. Recently, [19] showed that the overlay routing algorithms proposed in [3] [16] are not throughput-optimal in general, and developed the Optimal Overlay Routing Policy (OORP) for overlay nodes. However, all of the existing overlay routing algorithms [3], [16], [19] impose very stringent assumptions about the behavior of underlay nodes. In particular, the underlay nodes are required to use fixed-path routing (e.g., shortest-path routing) and maintain stability whenever possible, which fails to account for many important underlay policies (e.g., underlay nodes may use multi-path routing).

In terms of technical tools, our work leverages techniques from reinforcement learning, since a partially-controllable network with queue-dependent uncontrollable policy can be formulated as an MDP with unknown dynamics. Over the past few decades, many reinforcement learning algorithms have been developed, such as Q-learning [22], actor critic [4] and policy gradient [27]. Recently, the successful applications of deep neural networks in reinforcement learning algorithms have produced many deep reinforcement learning algorithms such as DQN [9], DDPG [6] and TRPO [20]. However, most of these methods are heuristic-based and do not have any performance guarantees. Among the existing reinforcement learning algorithms there are a few that do provide good performance bounds, such as model-based reinforcement learning algorithms UCRL [1], [2] and PSRL [13], [15]. Unfortunately, these algorithms require that the size of the state space be relatively small, which cannot be directly applied in our context since the state space (i.e., queue length) contains countably-infinite states.

B. Our Contributions

In this paper, we investigate optimal network control for a partially-controllable network. Whereas existing works (e.g., [3], [16], [19]) imposed very stringent assumptions about the behavior of uncontrollable nodes for analytical tractability, this is the first work that establishes stability results under the generalized partially-controllable network model. In particular, we develop two network control algorithms.

First, we develop a low-complexity Tracking-MaxWeight (TMW) algorithm that is guaranteed to achieve network stability if uncontrollable nodes adopt queue-agnostic policies. The Tracking-MaxWeight algorithm enhances the original MaxWeight algorithm [26] with an explict learning of the policy used by uncontrollable nodes.

Next, we propose a new reinforcement learning algorithm (i.e., the TUCRL algorithm) in the more challenging scenario where uncontrollable nodes may use queue-dependent policies. It combines the state-of-art model-based UCRL algorithm [1], [2] with a queue truncation technique to overcome the problem with countably-infinite queue length space. We prove that TUCRL achieves network stability by dropping a negligible fraction of packets. We also show that the TUCRL algorithm maintains a three-way tradeoff between delay, throughput and convergence rate.

II. System Model

Consider a networked system with N nodes (the set of all nodes is denoted by $\mathcal{N}$). There are K flows in the network and each node i maintains a queue for buffering undelivered packets for each flow k. As a result, there are NK queues in the network, and we denote by $\mathbf{Q}(t)$ the queue length vector at the beginning of time slot t, where its element $Q_{ik}(t)$ represents the queue length for flow k at node i.

Let ω_t be the *network event* that occurs in slot t, which includes information about the current network parameters, such as a vector of channel conditions for each link and a vector of exogenous arrivals to each queue. We assume that the sequence of network events $\{\omega_t\}_{t\geq 0}$ follow a stationary stochastic process. In particular, the vector of exogenous packet arrivals is denoted by $\mathbf{a}(\omega_t) = \{a_{ik}(t)\}_{i,k}$, where $a_{ik}(t)$ is the number of exogenous arrivals to queue (i,k) in slot t. Denote by $\lambda_{ik} = \mathbb{E}[a_{ik}(t)]$ the expected exogenous packet arrival rate to queue (i,k) in steady state.

At the beginning of each time slot t, after observing the current network event ω_t and the current queue length vector

$\mathbf{Q}(t)$, each node i needs to make a routing decision $f_{ijk}(t)$ indicating the offered transmission rate for flow k over link $i \to j$. The corresponding network routing vector is denoted by $\mathbf{f}(t) = \{f_{ijk}(t)\}_{i,j,k}$.

There are two types of nodes in the network: controllable nodes (the set of controllable nodes is denoted by $\mathcal{C}$) and uncontrollable nodes (the set of uncontrollable nodes is denoted by $\mathcal{U}$). The network operator can only control the routing behavior for controllable nodes while the routing actions taken by uncontrollable nodes cannot be regulated and are only observable at the end of each time slot. In this case, the network routing vector $\mathbf{f}(t)$ can be decomposed into two parts: $\mathbf{f}(t) = (\mathbf{f}^c(t), \mathbf{f}^u(t))$. Here, $\mathbf{f}^c(t) = \{f_{ijk}(t)\}_{i\in\mathcal{C}}$ represents the routing decisions made by controllable nodes (referred to as the **controllable action**) and $\mathbf{f}^u(t) = \{f_{ijk}(t)\}_{i\in\mathcal{U}}$ corresponds to the routing decisions made by uncontrollable nodes (referred to as the **uncontrollable action**). The routing vectors $\mathbf{f}^c(t)$ and $\mathbf{f}^u(t)$ are constrained within some action spaces $\mathcal{F}^c_{\omega_t}$ and $\mathcal{F}^u_{\omega_t}$, respectively, that may depend on the current network event ω_t, respectively. The action space for all nodes is denoted by $\mathcal{F}_{\omega_t} = \mathcal{F}^c_{\omega_t} \cup \mathcal{F}^u_{\omega_t}$. The action space can be used to specify routing constraints (e.g., the total transmission rate over each link should not exceed its capacity) or describe scheduling constraints (e.g., each node can only transmit to one of its neighbors in each time slot).

Note that when there is not enough backlog to transmit, the actual number of transmitted packets may be less than the offered transmission rate. In particular, we denote by $\widetilde{f}_{ijk}(Q_{ik}(t))$ (or simply $\widetilde{f}_{ijk}(t)$ if the context is clear) the actual number of transmitted packets in flow k over link $i \to j$ in slot t under the current queue length $Q_{ik}(t)$. Clearly, we have $\widetilde{f}_{ijk}(Q_{ik}(t)) \le \min\{f_{ijk}(t), Q_{ik}(t)\}$. We further assume that the routing decision can always be chosen to respect the backlog constraints (but the actual actions may not necessarily be queue-respecting). This can be done simply by never attempting to transmit more data than we have. Under such notations, the queuing dynamics are given by

$$\begin{aligned} &Q_{ik}(t+1) \\ =&Q_{ik}(t) + a_{ik}(t) + \sum_{j\in\mathcal{N}} \widetilde{f}_{jik}(t) - \sum_{j\in\mathcal{N}} \widetilde{f}_{ijk}(t) \\ \le&\Big[Q_{ik}(t) + a_{ik}(t) + \sum_{j\in\mathcal{N}} f_{jik}(t) - \sum_{j\in\mathcal{N}} f_{ijk}(t)\Big]^+, \end{aligned}$$

where $[z]^+ = \max\{z, 0\}$. We also make the following boundedness assumption: the amount of exogenous arrivals and the offered transmission rate in each time slot are bounded by some constant D, i.e.,

$$0 \le a_{ik}(t) \le D, \quad 0 \le f_{ijk}(t) \le D, \ \forall i, j, k.$$

A network control policy π is a mapping from the observed network event ω and queue length vector $\mathbf{Q}$ to a feasible routing action. In particular, denote by $\pi_c : (\omega, \mathbf{Q}) \mapsto \mathbf{f}^c$ a **controllable policy** and $\pi_u : (\omega, \mathbf{Q}) \mapsto \mathbf{f}^u$ an **uncontrollable policy**. In this paper, we assume that the uncontrollable policy π_u remains fixed over time but is unknown to the network operator. Our objective is to find a controllable policy π_c such that network stability can be achieved, as is defined as follows.

Definition 1. *A network is rate stable if*

$$\lim_{t\to\infty} \frac{\mathbb{E}[Q_{ik}(t)]}{t} = 0, \ \forall i, k.$$

Rate stability means that the average arrival rate to each queue equals the average departure rate from that queue.

III. Queue-Agnostic Uncontrollable Policy

In this section, we consider the scenario where the uncontrollable policy is queue-agnostic, which simply observes the current network event ω_t and makes a routing decision $\mathbf{f}^u(t) \in \mathcal{F}^u_{\omega_t}$ as a stationary function only of ω_t, i.e., $\pi_u : \omega_t \mapsto \mathbf{f}^u(t)$. In the stochastic network optimization literature, such a policy is also referred to as an **ω-only policy** [11]. Despite their simple form, ω-only policies can capture a wide range of network control protocols in practice, such as shortest-path routing protocols (e.g., OSPF, RIP) , multi-path routing protocols (e.g., ECMP) and randomized routing protocols.

Unfortunately, even under simple ω-only uncontrollable policies, existing routing algorithms may fail to stabilize the network. For example, as is illustrated in Figure 2, the well-known Backpressure routing algorithm achieves low throughput when uncontrollable node uses queue-agnostic policies. In this example, the failure is due to the fact that some uncontrollable node uses a non-stabilizing policy that does not preserve flow conservation but the Backpressure algorithm is not aware of this non-stabilizing behavior.

In this section, we propose a low-complexity algorithm that learns the behavior of uncontrollable nodes and achieves network stability under any ω-only uncontrollable policy.

A. Tracking-MaxWeight Algorithm

Now we introduce an algorithm that achieves network stability whenever uncontrollable nodes use an ω-only policy. The algorithm is called Tracking-MaxWeight (TMW), which enhances the original MaxWeight algorithm [26] with an explicit learning of the policy used by uncontrollable nodes. Throughout this section, we let $\{\mathbf{f}^u(t)\}_{t\ge0}$ be the sequence of routing actions that are actually executed by uncontrollable nodes.

The details of the TMW algorithm are presented in Algorithm 1. In each slot t, the TMW algorithm generates the routing actions $\mathbf{g}^c(t) = \{g_{ijk}(t)\}_{i\in\mathcal{C}}$ for controllable nodes and also produces an "imagined" routing action $\mathbf{g}^u(t) = \{g_{ijk}(t)\}_{i\in\mathcal{U}}$ for uncontrollable nodes, by solving the optimization problem (2). With these calculated actions, the TMW algorithm then updates two virtual queues. The first virtual queue $\mathbf{X}(t)$ tries to emulate the physical queue $\mathbf{Q}(t)$ but assumes that the imagined uncontrollable action $\mathbf{g}^u(t)$ is applied (while the physical queue is updated using the true uncontrollable action $\mathbf{f}^u(t)$). The second virtual

queue $\mathbf{Y}(t)$ tracks the cumulative difference between the imagined uncontrollable actions $\{\mathbf{g}^u(t)\}_{t\geq 0}$ and the actual uncontrollable actions $\{\mathbf{f}^u(t)\}_{t\geq 0}$. In particular, we use $\Delta_{ijk}(t)$ to measure the difference between the imagined routing action $g_{ijk}(t)$ and the true routing action $f_{ijk}(t)$ taken by uncontrollable node $i \in \mathcal{U}$, which is given by

$$\Delta_{ijk}(t) = g_{ijk}(t) - \widetilde{f}_{ijk}(t),\ \forall i \in \mathcal{U}, \tag{1}$$

where $\widetilde{f}_{ijk}(t)$ is the actual number of transmitted packets under the true routing action $f_{ijk}(t)$ given the current queue backlog $\mathbf{Q}(t)$. Note that for each controllable node $i \in \mathcal{C}$, we simply set $\Delta_{ijk}(t) = 0$.

The optimization problem (2) aims at maximizing a weighted sum of flow variables, which is similar to the optimization problem solved in the original MaxWeight algorithm [26] except for the setting of weights. In the original MaxWeight algorithm, the weight is $W_{ijk}(t) = Q_{ik}(t) - Q_{jk}(t)$ corresponding to the physical queue backlog differential, while in the Tracking-MaxWeight algorithm the weight $W_{ijk}(t) = X_{ik}(t) - X_{jk}(t) - Y_{ijk}(t)$ accounts for both the backlog differential for virtual queue $\mathbf{X}(t)$ and the backlog of virtual queue $\mathbf{Y}(t)$. The derivation of (2) is based on the minimization of quadratic Lyapunov drift terms for the two virtual queues:

$$\min_{\mathbf{g}(t)\in\mathcal{F}_{\omega_t}} \quad \sum_{i,k} X_{ik}(t)\Big[a_{ik}(t) + \sum_j g_{jik}(t) - \sum_j g_{ijk}(t)\Big] + \sum_{i,j,k} Y_{ijk}(t)\Big(g_{ijk}(t) - \widetilde{f}_{ijk}(t)\Big),$$

where the first term corresponds to the Lyapunov drift of virtual queue $\mathbf{X}(t)$ and the second term is the Lyapunov drift of virtual queue $\mathbf{Y}(t)$. Note that the minimization is done over controllable actions $\mathbf{g}^c(t)$ and "imagined" uncontrollable actions $\mathbf{g}^u(t)$. Cleaning up irrelevant constants, i.e., $a_{ik}(t)$ and $\widetilde{f}_{ijk}(t)$, and rearranging terms yield the optimization problem (2).

Next we show that Tracking-MaxWeight achieves stability whenever uncontrollable nodes use an ω-only policy and the network is *within the stability region*, i.e., there exists a sequence of feasible routing vectors $\{\mathbf{f}^c(t)\}_{t\geq 0}$ for controllable nodes such that

$$\lambda_{ik} + \sum_{j\in\mathcal{N}} \widetilde{f}_{jik} - \sum_{j\in\mathcal{N}} \widetilde{f}_{ijk} \leq 0,\ \forall i,k, \tag{3}$$

where

$$\widetilde{f}_{ijk} = \lim_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E}[\widetilde{f}_{ijk}(Q^*_{ik}(t))]$$

is the long-term average actual flow transmission rate under $\{\mathbf{f}^c(t), \mathbf{f}^u(t)\}_{t\geq 0}$ and $\{\mathbf{Q}^*(t)\}_{t\geq 0}$ is the corresponding optimal queue length trajectory. In other words, (3) requires that flow conservation should be preserved for every queue under the optimal controllable policy, otherwise no algorithm can stabilize the network. It is important to note that in (3) the flow conservation law is with respect to the *actual* transmissions since an uncontrollable node may not preserve flow conservation in terms of its offered transmissions (e.g., in Figure 2, the offered incoming rate to node 3 is 40 while the offered outgoing rate from node 3 is 0). The only way to stabilize these nodes is by limiting the amount of backlog such that the actual endogenous arrivals to these nodes are smaller. The performance of Tracking-MaxWeight is given in the following theorem.

Algorithm 1 Tracking-MaxWeight (TMW)

1: In each slot t, observe the current network event ω_t and solve the following optimization problem to obtain the controllable action $\mathbf{g}^c(t)$ and the *imagined* uncontrollable action $\mathbf{g}^u(t)$:

$$\max_{\mathbf{g}(t)\in\mathcal{F}_{\omega_t}} \quad \sum_{(i,j)} \sum_k g_{ijk}(t) W_{ijk}(t), \tag{2}$$

where

$$W_{ijk}(t) = X_{ik}(t) - X_{jk}(t) - Y_{ijk}(t).$$

2: Controllable nodes execute the routing decision $\mathbf{g}^c(t)$.

3: Observe the true routing action $\mathbf{f}^u(t)$ taken by uncontrollable nodes and update virtual queues:

$$X_{ik}(t+1) = \Big[X_{ik}(t) + a_{ik}(t) + \sum_{j\in\mathcal{N}} g_{jik}(t) - \sum_{j\in\mathcal{N}} g_{ijk}(t)\Big]^+$$

$$Y_{ijk}(t+1) = Y_{ijk}(t) + \Delta_{ijk}(t)$$

where $\Delta_{ijk}(t)$ is defined in (1).

Theorem 1. *When uncontrollable nodes use an ω-only policy and the network is within the stability region, Tracking-MaxWeight achieves rate stability.*

Proof: The proof first shows that the two virtual queues $\mathbf{X}(t)$ and $\mathbf{Y}(t)$ are stable under the TMW algorithm using the Lyapunov drift analysis. Then we prove that whenever the two virtual queues are stable, the physical queue $\mathbf{Q}(t)$ is also stable. See technical report [5] for details. ■

IV. Queue-Dependent Uncontrollable Policy

The previous section investigated the scenario where uncontrollable nodes use a queue-agnostic policy (i.e., ω-only policy). In this section, we study a more general case where the uncontrollable policy may be *queue-dependent*, which can be used to describe many state-of-the-art optimal network control protocols. For example, the well-known Backpressure algorithm makes routing decisions based on the currently observed queue length vector. In this scenario, the uncontrollable policy is a fixed mapping from the observed network event ω_t and the observed queue length vector $\mathbf{Q}(t)$ to a routing vector $\mathbf{f}^u(t)$ for uncontrollable nodes, i.e., $\pi_u : \Big(\omega_t, \mathbf{Q}(t)\Big) \mapsto \mathbf{f}^u(t)$.

Note that the queueing dynamics are

$$Q_{ik}(t+1) \leq \Big[Q_{ik}(t) + a_{ik}(t) + \sum_{j\in\mathcal{C}} f_{jik}(t) + \sum_{j\in\mathcal{U}} f_{jik}(t) - \sum_{j\in\mathcal{N}} f_{ijk}(t)\Big]^+,\ \forall i,k.$$

Since for each $j \in \mathcal{U}$, its routing variable $f_{jik}(t)$ is an arbitrary (unknown) function of $\mathbf{Q}(t)$, the above queueing dynamics could depend on $\mathbf{Q}(t)$ in an arbitrary (unknown) way that is not in the simple piecewise-linear form as in the classic Lindley recursion. As a result, we rewrite the queueing dynamics as

$$\mathbf{Q}(t+1) = \beta(\mathbf{f}^c(t), \mathbf{Q}(t), \omega_t), \tag{4}$$

where $\beta(\cdot)$ is some unknown function that depends on our controllable routing action $\mathbf{f}^c(t)$, the current queue length vector $\mathbf{Q}(t)$ and the observed network event ω_t.

Due to the unknown queueing dynamics, many analytical tools for optimal network control break down. For example, the previous Tracking-MaxWeight algorithm utilizes the Lyapunov drift analysis which is not applicable if the queueing dynamics do not follow the Lindley recursion. As a result, optimal network control becomes very challenging and fundamentally different from the traditional stochastic network optimization framework. In the following, we first formulate the problem a **Markov Decision Process (MDP) with unknown dynamics** and then propose a new reinforcement learning algorithm that can achieve network stability under mild conditions.

Before moving on to the technical details, we first introduce some notations and assumptions that will be used throughout this section. For convenience, we define action $\alpha_t \triangleq \mathbf{f}^c(t)$ and simply write "controllable routing action $\mathbf{f}^c(t)$" as "action α_t", since the uncontrollable routing action $\mathbf{f}^u(t)$ has been implicitly treated as a part of the environment (see queueing dynamics (4)). For the same reason, "controllable policy π_c" and "policy π" are also used interchangeably. The action space for α_t is denoted by $\mathcal{A}$ which is assumed to be fixed and finite. We also make the following assumption regarding the optimal system performance.

Assumption 1. *There exists a policy π^* such that $\sum_{i,k} Q^*_{ik}(t) < \infty$ with probability 1 for any $t \geq 0$, where $\mathbf{Q}^*(t)$ is the queue length vector in slot t under policy π^*.*

In other words, it is required that the total queue length should remain bounded under an optimal policy π^* otherwise there is no hope for stabilizing the network. In essence, Assumption 1 requires that the network be stabilizable by some controllable policy π^*.

A. MDP Formulation

We formulate the problem of achieving network stability as an MDP $M = (\mathcal{A}, \mathcal{S}, \theta, P)$. Here $\mathcal{A}$ is the routing action space for controllable nodes, and $\mathcal{S}$ is the state space that corresponds to the queue length vector space $\mathcal{Q}$. The cost function $\theta(\alpha_t, \mathbf{Q}(t))$ under action α_t and state $\mathbf{Q}(t)$ is given by $\theta(\alpha_t, \mathbf{Q}(t)) = \sum_{i,k} Q_{ik}(t)$, which corresponds to the sum of queue lengths in slot t. In addition, P is the state transition matrix, where $P(\mathbf{Q}'|\mathbf{Q}, \alpha)$ is the probability that the next state is $\mathbf{Q}'$ when action α is taken under the current state $\mathbf{Q}$. Note that the transition matrix P is generated according to the queueing dynamics (4), and that the influence of network event and uncontrollable routing action has been implicitly incorporated into the probabilistic transition matrix P. Note also that the queueing dynamics $\beta(\cdot)$ are unknown, so this is an MDP with unknown dynamics, which is also referred to as a **Reinforcement Learning (RL)** problem [23].

Let $J^\pi\Big(M, \mathbf{Q}(0)\Big)$ be the time-average expected total queue length when policy π is applied in MDP M and the initial queue length vector is $\mathbf{Q}(0)$, i.e.,

$$J^\pi\Big(M, \mathbf{Q}(0)\Big) = \lim_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E}_{\pi,M}\Big[\sum_{i,k} Q_{ik}(t)\Big|\mathbf{Q}(0)\Big],$$

where the expectation $\mathbf{E}_{\pi,M}[\cdot]$ is with respect to the randomness of the queue length trajectory $\{\mathbf{Q}(t)\}_{t\geq 0}$ when policy π is applied in MDP M. Also let $J^*\Big(M, \mathbf{Q}(0)\Big) = \min_\pi J^\pi\Big(M, \mathbf{Q}(0)\Big)$ be the minimum time-average expected queue length under an optimal policy π^*. Our objective is to find an optimal policy that solves the MDP and achieves the minimum average queue length.

B. Challenges to Solving the MDP

The MDP has an unknown transition structure, which gives rise to an "exploration-exploitation" tradeoff. On one hand, we need to exploit the existing knowledge to make the best (myopic) decision; on the other hand, it is necessary to explore new states in order to learn which states may lead to lower costs in the future. Moreover, there might be some "trapping" sub-optimal states that take a long time (or is even impossible) for any policy to escape. Any algorithm that has zero knowledge about system dynamics at the beginning is likely to get trapped in these states during the exploration phase. Therefore, we need to impose restrictions on the transition structure in the MDP model. In particular, we restrict our consideration to weakly communicating MDPs with finite communication time, defined as follows.

Assumption 2. *For any two queue length vectors $\mathbf{Q}$ and $\mathbf{Q}'$ (except for those which are transient under every policy), there exists a policy π that can move from $\mathbf{Q}$ to $\mathbf{Q}'$ within $L||\mathbf{Q}' - \mathbf{Q}||_1$ time slots (in expectation), where L is a constant.*

In other words, it is assumed that there is no "trapping" state in the system otherwise no reinforcement learning algorithm can be guranteed to avoid the traps and optimally solve the MDP. Note that in a weakly communicating MDP, the optimal average cost does not depend on the initial state (cf. [18], Section 8.3.3). Thus we drop the dependence on the initial state $\mathbf{Q}(0)$, and write the optimal average cost (queue length) as $J^*(M)$.

Another challenge is that the MDP has a countably-infinite state space (i.e., queue length vector space). Existing reinforcement learning methods that can handle such an infinite state space are mostly heuristic-based (e.g., [9] [6] [20]), and do not have any performance guarantees. On

the other hand, there are a few reinforcement learning algorithms that do have good performance guarantees, but these algorithms require that the size of the state space be relatively small. Even if we consider a finite time horizon T, the size of the queue length vector space could be up to $O(T^N)$ (assuming bounded arrivals in each slot), which could lead to weak performance bounds. For example, in the UCRL algorithm [1], [2], the regret bound is $O(S\sqrt{T})$, where S is the size of the state space. If UCRL is applied in our context, the resulting regret bound would be $O(T^{N+0.5})$ which is a trivial super-linear regret bound.

C. TUCRL Algorithm

In this section, we develop an algorithm that achieves network stability under Assumptions 1 and 2. We call our algorithm Truncated Upper Confidence Reinforcement Learning (TUCRL), as it combines the model-based UCRL algorithm [1], [2] with a queue truncation technique that resolves the infinite state space problem.

Specifically, consider a *truncated system* where new exogenous packet arrivals are dropped when the total queue length $\sum_{i,k} Q_{ik}(t)$ reaches $V-1$ for some threshold $V \geq 1$. In such a truncated system, the state space is the truncated queue length vector space $\mathcal{Q}_V$ which contains all queue length vectors where the length of each queue does not exceed $V-1$. In order for packet dropping to be feasible, we assume that there is an admission control action that can shed new exogenous packets as needed.

Our TUCRL algorithm applies the model-based UCRL algorithm [1], [2] in the truncated system, which maintains an estimation for the unknown queueing dynamics and then computes the optimal policy under the estimated dynamics. It applies the "optimistic principle" for exploration, where under-explored state-action pairs are assumed to be able to result in lower costs, which implicitly encourages the exploration of novel state-action pairs.

The detailed description of TUCRL is presented in Algorithm 2, which is similar to the standard UCRL algorithm except that queue truncation is applied when appropriate. Specifically, the TUCRL algorithm proceeds in episodes, and the length of each episode is dynamically determined. In episode ℓ, the TUCRL algorithm first constructs an empirical estimation $\hat{P}$ for the transition matrix based on historical observations (step 1). In particular, the estimated transition probability from state $\mathbf{Q}$ to $\mathbf{Q}'$ under action α is

$$\hat{P}(\mathbf{Q}'|\mathbf{Q},\alpha) = \frac{n_\ell(\mathbf{Q},\alpha,\mathbf{Q}')}{n_\ell(\mathbf{Q},\alpha)}, \tag{5}$$

where $n_\ell(\mathbf{Q},\alpha)$ is the cumulative number of visits to state-action pair $(\mathbf{Q},\alpha)$ up until the beginning of episode ℓ and $n_\ell(\mathbf{Q},\alpha,\mathbf{Q}')$ is the number of times that transition $(\mathbf{Q},\alpha)\to\mathbf{Q}'$ happens up to the beginning of episode ℓ. Note that if $n_\ell(\mathbf{Q},\alpha)=0$, the estimated transition probability is set to be zero.

Then the TUCRL algorithm constructs an upper confidence set $\mathcal{M}_\ell$ for all plausible MDP models based on the empirical estimation $\hat{P}$ (step 2). The upper confidence set is constructed in a way such that it contains the true MDP model with high probability. Specifically, the upper confidence set $\mathcal{M}_\ell$ contains all the MDPs with truncated queue length space $\mathcal{Q}_V$ and transition matrix $P \in \mathcal{P}_\ell$ where

$$\mathcal{P}_\ell = \left\{P : \left|\left|P(\cdot|\mathbf{Q},\alpha) - \hat{P}(\cdot|\mathbf{Q},\alpha)\right|\right|_1 \leq \sqrt{\frac{C\log(2|\mathcal{A}|t_\ell V)}{\max\{1, n_\ell(\mathbf{Q},\alpha)\}}}\right\}. \tag{6}$$

Here, V is the queue truncation threshold, t_ℓ is the starting time of episode ℓ and $C \triangleq 2(2ND+1)^N$ is a constant.

Next, the TUCRL algorithm selects an "optimistic MDP" M_ℓ that yields the minimum average queue length among all the plausible MDPs in the confidence set $\mathcal{M}_\ell$, and computes a nearly-optimal policy π_ℓ under MDP M_ℓ (step 3). The joint selection of the optimistic MDP and the calculation of the nearly-optimal policy are referred to an **optimistic planning** [10]. There are many efficient methods for performing optimistic planning, such as Extended Value Iteration [2] and OP-MDP [24]. We provide a description of Extended Value Iteration in technical report [5].

Finally, the computed policy π_ℓ is executed until the stopping condition of episode ℓ is triggered. An episode ends when the number of visits to some state-action pair doubles, i.e., when we encounter a state-action pair $(\mathbf{Q}(t),\alpha_t)$ such that its visiting frequency in episode ℓ ($v_\ell(\mathbf{Q}(t),\alpha_t)$) equals its cumulative visiting frequency up to the beginning of episode ℓ ($n_\ell(\mathbf{Q}(t),\alpha_t)$). We will show that this stopping condition guarantees that the total number of episodes up to time T is $O(V^N \log T)$ (see technical report [5]). Note that during the execution of policy π_ℓ, new packet arrivals may be dropped if the total queue length exceeds $V-1$. Here, the dropped packets could be any new arrivals to any queue. We will prove that the fraction of dropped packets is negligible if the threshold V is properly selected.

D. Performance of TUCRL Algorithm

The following theorem characterizes the performance of the TUCRL algorithm regarding its queue length, packet dropping rate and convergence rate.

Theorem 2. *Under Assumptions 1 and 2, the performance of the TUCRL algorithm is as follows.*

- ***(Queue Length)*** *The time-average expected queue length converges to a bounded value:*

$$\lim_{T\to\infty}\frac{1}{T}\sum_{t=0}^{T-1}\sum_{i,k}\mathbb{E}[Q_{ik}(t)] \leq \Theta(1) - \Theta\Big(\frac{1}{V}\Big).$$

- ***(Packet Dropping Rate)*** *The long-term expected fraction of dropped packets is*

$$\lim_{T\to\infty}\mathbb{E}[\eta_T] \leq \Theta\Big(\frac{1}{V}\Big),$$

where η_T is the fraction of dropped packets within T slots.

- ***(Convergence Rate)*** *The time-average expected queue length after T slots is within a $\widetilde{\mathcal{O}}\Big(\frac{poly(V)}{\sqrt{T}}\Big)$-neighborhood of the steady-state expected queue length, where $poly(V)$*

Algorithm 2 Truncated Upper Confidence Reinforcement Learning (TUCRL)

Input: queue truncation threshold V

Set $t = 0$

for episode $\ell = 1, 2, \cdots$ **do**

1. Initialize episode ℓ:

• Set the start of episode ℓ: $t_\ell = t$

• Initialize state-action count for episode ℓ: $v_\ell(\mathbf{Q}, \alpha) = 0$

• Update accumulative state-action count $n_\ell(\mathbf{Q}, \alpha)$ and transition count $n_\ell(\mathbf{Q}, \alpha, \mathbf{Q}')$ up to episode ℓ

• Estimate transition probability $\hat{P}(\mathbf{Q}'|\mathbf{Q}, \alpha)$ according to (5) for any $\mathbf{Q}, \mathbf{Q}' \in \mathcal{Q}_V$ and $\alpha \in \mathcal{A}$

2. Construct upper confidence set:

Construct a confidence set $\mathcal{M}_\ell$ that contains all the MDPs with truncated queue space $\mathcal{Q}_V$ and transition matrix $P \in \mathcal{P}_\ell$ as shown in (6)

3. Optimistic planning:

Compute the optimistic MDP model M_ℓ (that yields the minimum average total queue length) in the confidence set $\mathcal{M}_\ell$ and a nearly-optimal policy π_ℓ under M_ℓ (up to accuracy $\frac{1}{\sqrt{t_\ell}}$)

4. Execute policy (with packet dropping):

repeat

• Observe current queue length vector $\mathbf{Q}(t)$ and new exogenous arrivals $\mathbf{a}(t)$

• Arbitrarily drop $\left[\sum_{i,k}\left(Q_{ik}(t)+a_{ik}(t)\right)-V+1\right]^+$ newly arrived packets from the network

• Take action $\alpha_t = \pi_\ell(\mathbf{Q}(t))$

• Update $v_\ell(\mathbf{Q}(t), \alpha_t) = v_\ell(\mathbf{Q}(t), \alpha_t) + 1$

• $t = t + 1$

until $v_\ell(\mathbf{Q}(t), \alpha_t) = \max\{1, n_\ell(\mathbf{Q}(t), \alpha_t)\}$

end for

is some polynomial in V and $\widetilde{\mathcal{O}}$ is the big-O notation that ignores any logarithmic term.

Proof: We first find an upper bound on the total queue length under TUCRL in the truncated system. We further analyze the fraction of time when queue truncation is triggered by using concentration inequalities. See technical report [5] for details. ∎

There are several important observations regarding Theorem 2. First, the TUCRL algorithm achieves bounded queue length by dropping a negligible fraction of packets under a suitably large value of V. Second, there is a **three-way tradeoff** between total queue length (delay), packet dropping rate (throughput) and convergence rate. For example, by increasing the value of V, the packet dropping rate becomes smaller (i.e., throughput becomes higher) but the total queue length (delay) increases and the convergence becomes slower. Similar three-way tradeoffs between utility, delay and convergence rate are discussed in [8].

Complexity of TUCRL. The time complexity of TUCRL is dominated by the complexity of the optimistic planning module (step 3) which is implementation-dependent. For example, if a naive Extended Value Iteration (see technical report [5]) is used, the time complexity of each value iteration step is exponential in the number of queues and thus cannot scale to large-scale problems. One way to scale the optimisic planning module is by using approximate dynamic programming that employs various approximation techniques in the planning procedure, such as using linear functions or neural networks to approximate the value function (see [17] for a comprehensive introduction). Recent deep reinforcement learning techniques may also be leveraged to efficiently perform value iterations in large-scale problems, such as Randomized Least-Squares Value Iteration (RLSVI) [14], Value Iteration Networks (VIN) [25] and Value Prediction Networks (VPN) [12].. Such approximations will not lead to significant changes in the performance of TUCRL since we only require an approximate solution in step 3.

V. Simulation Results

A. Scenario 1: Queue-Agnostic Uncontrollable Policy

We first study the partially-controllable network shown in Figure 3. There are two flows: $1 \to 4$ and $6 \to 4$. Each node in the network needs to make a routing and scheduling decision in every time slot. The constraint is that each node can transmit to only one of its neighbors in each time slot and the transmission rate over each link cannot exceed its capacity. Node 2 and node 3 are uncontrollable nodes that use randomized queue-agnostic policies. Specifically, uncontrollable node 2 uses a randomized routing algorithm that transmits any packets it received to either node 3 or node 5 with an equal probability in each time slot. Uncontrollable node 3 uses a randomized scheduling policy that serves flow $1 \to 4$ or flow $6 \to 4$ with an equal probability in each time slot. The arrival rate of flow $6 \to 4$ is 5. In this case, it can be shown that the maximum supportable arrival rate for flow $1 \to 4$ is 25 given the routing constraints and the behavior of uncontrollable nodes.

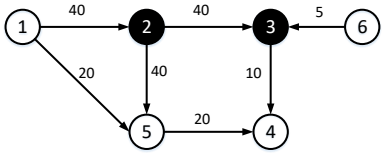
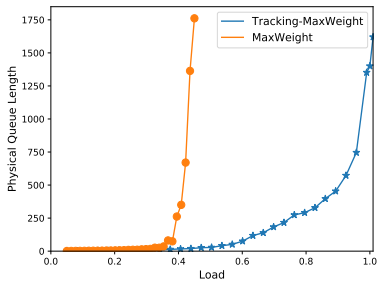
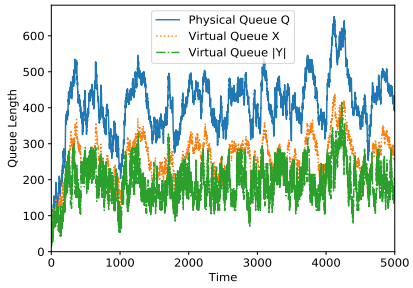


Fig. 3. Network topology used in simulation scenario 1. The number next to each link is its capacity. Each node can only transmit to one of its neighbors in each slot. Black nodes are uncontrollable nodes that use randomized queue-agnostic policies.

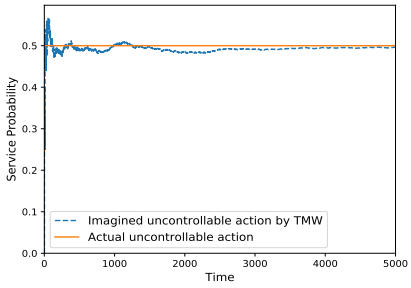
We have shown in Section III that the Tracking-MaxWeight (TMW) algorithm achieves the optimal throughput in this scenario. In Figure 4(a), we compare Tracking-MaxWeight with the well-known MaxWeight algorithm (i.e., BackPressure routing), in terms of the supportable rate for



(a) Throughput performance of MaxWeight and Tracking-MaxWeight.



(b) Queue length under the TMW algorithm (load = 0.99).



(c) The TMW algorithm quickly learns that node 3 serves flow $1 \to 4$ with probability 0.5.

Fig. 4. Performance of the Tracking-MaxWeight (TMW) algorithm in Scenario 1.

flow $1 \to 4$. Specifically, Figure 4(a) shows the total queue length achieved by MaxWeight and Tracking-MaxWeight under different system loads (if the load is ρ, then the arrival rate of flow $1 \to 4$ is 25ρ while the arrival rate of flow $6 \to 4$ is fixed to 5). It is observed that MaxWeight can only support around 40% arrivals (the queue length under MaxWeight blows up at load ≈ 0.4). By comparison, our Tracking-MaxWeight achieves the optimal throughput.

We further examine the behavior of the Tracking-MaxWeight algorithm in Figure 4(b) and Figure 4(c). Specifically, Figure 4(b) shows the queue length trajectory for the physical queue $\mathbf{Q}(t)$ and the two virtual queues $\mathbf{X}(t), \mathbf{Y}(t)$. As our theory predicts, both the physical queue $\mathbf{Q}(t)$ and the two virtual queues $\mathbf{X}(t), \mathbf{Y}(t)$ are stable under the TMW algorithm. Figure 4(c) shows the learning curve of the TMW algorithm for the uncontrollable policy used by node 3. In particular, node 3 uses randomized scheduling that serves flow $1 \to 4$ and flow $6 \to 4$ with an equal probability 0.5. It is observed in Figure 4(c) that the TMW algorithm quickly learns the service probability for flow $1 \to 4$ at node 3 (i.e., the "imagined uncontrollable action" in TMW approaches the true uncontrollable action).

B. Scenario 2: Queue-Dependent Uncontrollable Policy

Next we study a more challenging scenario where the action taken by uncontrollable nodes is queue-dependent. In particular, consider the network topology shown in Figure 5 where node 2 and node 3 are uncontrollable. There is only one flow $1 \to 4$ and the constraint is that each node can transmit to only one of its neighbours in each time slot. The policy used by the two uncontrollable nodes is as follows. Let $\mu_{24}(t)$ and $\mu_{34}(t)$ be the transmission rate that node 2 and node 3 allocates to the flow in slot t, respectively. Then

$$(\mu_{24}(t), \mu_{34}(t)) = \begin{cases} (0.5, 0) & Q_3(t) \le 10 \\ (0, 1) & Q_2(t) \le 10 \text{ and } Q_3(t) > 10 \\ (0.25, 0.25) & Q_2(t) > 10 \text{ and } Q_3(t) > 10 \end{cases}$$

As a result, the maximum throughput of 1 can be supported only if $Q_2(t)$ is small ($Q_2(t) \le 10$) and $Q_3(t)$ is large ($Q_3(t) > 10$). Although this is an artificial example, it sheds light on the challenges when uncontrollable nodes use queue-dependent policies: any throughput-optimal algorithm should be able to efficiently learn which queue length region can support the maximum throughput and keep the queue length within this region.

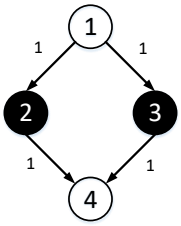


Fig. 5. Network topology used in simulation scenario 2 The number next to each link is its capacity. Each node can only transmit to one of its neighbors in each slot. There is only one flow $1 \to 4$. Black nodes are uncontrollable nodes that use queue-dependent policies.

We first compare the throughput performance of TUCRL with MaxWeight and Tracking-MaxWeight. Note that the TUCRL algorithm occasionally drops packets. In order to make a fair comparison, the throughput performance is measured with respect to the number of packets that have been delivered. It is observed in Figure 6 that TUCRL achieves the optimal throughput while MaxWeight or Tracking-MaxWeight only deliver a throughput of 0.5 in this scenario. It should be noted that the TUCRL algorithm takes longer time to learn and converge than MaxWeight or Tracking-MaxWeight.

Next we investigate the performance of the TUCRL algorithm under different values of the truncation threshold V. As we proved in Theorem 2, the value of V determines a three-way tradeoff between queue length, packet dropping rate and convergence rate. As is illustrated in Figure 7 and Figure 8, a larger value of V leads to a larger queue length and the convergence becomes slower, but the fraction of dropped packets becomes smaller. Note that when $V = 20$ and $V = 30$, the fraction of dropped packets becomes very small as time goes by. In contrast, when $V = 5$, the fraction of dropped packets remains non-negligible ($\sim 60\%$) since the TUCRL algorithm cannot explore the "throughput-optimal region" where $Q_2(t) \le 10$ and $Q_3(t) > 10$ with a queue truncation threshold $V = 5$.

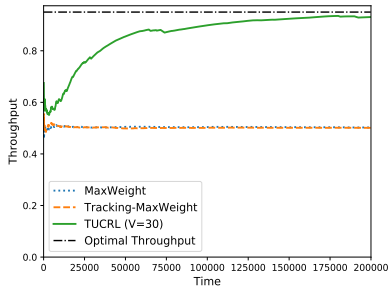


Fig. 6. Throughput comparison among MaxWeight, Tracking MaxWeight and TUCRL in Scenario 2 (load = 0.95).

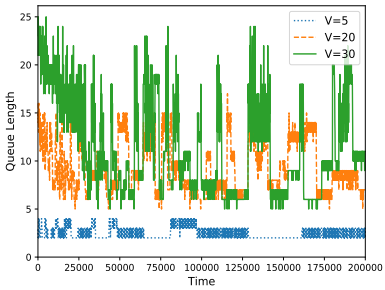


Fig. 7. Queue length under the TUCRL algorithm with different queue truncation threshold V (load = 0.95).

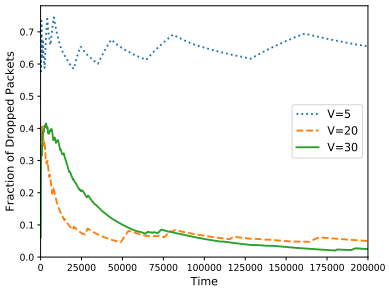


Fig. 8. Fraction of dropped packets under the TUCRL algorithm with different queue truncation thresholds V (load = 0.95).

REFERENCES

[1] Peter Auer and Ronald Ortner. Logarithmic online regret bounds for undiscounted reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 49–56, 2007.

[2] Thomas Jaksch, Ronald Ortner, and Peter Auer. Near-optimal regret bounds for reinforcement learning. *Journal of Machine Learning Research*, 11(Apr):1563–1600, 2010.

[3] Nathaniel M Jones, Georgios S Paschos, Brooke Shrader, and Eytan Modiano. An overlay architecture for throughput optimal multipath routing. *IEEE/ACM Transactions on Networking*, 2017.

[4] Vijay R Konda and John N Tsitsiklis. Actor-critic algorithms. In *Advances in neural information processing systems*, pages 1008–1014, 2000.

[5] Qingkai Liang and Eytan Modiano. Optimal network control in partially-controllable networks. *arXiv preprint arXiv:1901.01517*, 2019.

[6] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[7] David V Lindley. The theory of queues with a single server. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 48, pages 277–289. Cambridge University Press, 1952.

[8] Jia Liu, Atilla Eryilmaz, Ness B Shroff, and Elizabeth S Bentley. Heavy-ball: A new approach to tame delay and convergence in wireless network optimization. In *INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications, IEEE*, pages 1–9. IEEE, 2016.

[9] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[10] Rémi Munos et al. From bandits to monte-carlo tree search: The optimistic principle applied to optimization and planning. *Foundations and Trends® in Machine Learning*, 7(1):1–129, 2014.

[11] Michael J Neely. Stochastic network optimization with application to communication and queueing systems. *Synthesis Lectures on Communication Networks*, 3(1):1–211, 2010.

[12] Junhyuk Oh, Satinder Singh, and Honglak Lee. Value prediction network. In *Advances in Neural Information Processing Systems*, pages 6120–6130, 2017.

[13] Ian Osband, Daniel Russo, and Benjamin Van Roy. (more) efficient reinforcement learning via posterior sampling. In *Advances in Neural Information Processing Systems*, pages 3003–3011, 2013.

[14] Ian Osband, Daniel Russo, Zheng Wen, and Benjamin Van Roy. Deep exploration via randomized value functions. *arXiv preprint arXiv:1703.07608*, 2017.

[15] Yi Ouyang, Mukul Gagrani, Ashutosh Nayyar, and Rahul Jain. Learning unknown markov decision processes: A thompson sampling approach. In *Advances in Neural Information Processing Systems*, pages 1333–1342, 2017.

[16] Georgios S Paschos and Eytan Modiano. Throughput optimal routing in overlay networks. In *Communication, Control, and Computing (Allerton), 2014 52nd Annual Allerton Conference on*, pages 401–408. IEEE, 2014.

[17] Warren B Powell. *Approximate Dynamic Programming: Solving the curses of dimensionality*, volume 703. John Wiley & Sons, 2007.

[18] Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[19] Anurag Rai, Rahul Singh, and Eytan Modiano. A distributed algorithm for throughput optimal routing in overlay networks. *arXiv preprint arXiv:1612.05537*, 2016.

[20] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International Conference on Machine Learning*, pages 1889–1897, 2015.

[21] Ramesh K Sitaraman, Mangesh Kasbekar, Woody Lichtenstein, and Manish Jain. Overlay networks: An akamai perspective. *Advanced Content Delivery, Streaming, and Cloud Services*, 51(4):305–328, 2014.

[22] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[23] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[24] Balázs Szörényi, Gunnar Kedenburg, and Remi Munos. Optimistic planning in markov decision processes using a generative model. In *Advances in Neural Information Processing Systems*, pages 1035–1043, 2014.

[25] Aviv Tamar, Yi Wu, Garrett Thomas, Sergey Levine, and Pieter Abbeel. Value iteration networks. In *Advances in Neural Information Processing Systems*, pages 2154–2162, 2016.

[26] Leandros Tassiulas and Anthony Ephremides. Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks. *IEEE transactions on automatic control*, 37(12):1936–1948, 1992.

[27] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. In *Reinforcement Learning*, pages 5–32. Springer, 1992.