Universal Policy Tracking: Scheduling for Wireless Networks with Delayed State Observation

Bai Liu and Eytan Modiano
Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, Cambridge, MA

Abstract—Numerous scheduling algorithms have been proposed to optimize various performance metrics like throughput, delay and utility in wireless networks. However, these algorithms often require instantaneous access to network state information, which is not always available. While network stability can sometimes be achieved with delayed state information, other performance metrics such as latency may degrade. Thus, instead of simply stabilizing the system, our goal is to design a framework that can mimic arbitrary scheduling algorithms with performance guarantees. A naive approach is to make decisions directly with delayed information, but we show that such methods may lead to poor performance. Instead, we propose the Universal Tracking (UT) algorithm that can mimic the actions of arbitrary scheduling algorithms under observation delay. We rigorously show that the performance gap between UT and the scheduling algorithm being tracked is bounded by constants. Our numerical experiments show that UT significantly outperforms the naive approach in various applications.

I. INTRODUCTION

Next generation (NextG) wireless networks have been extensively discussed and studied in recent years. Due to economic concerns, instead of building new infrastructures, an increasing number of NextG wireless service providers (SPs) prefer adapting the over-the-top (OTT) framework [24]. Under the OTT framework, multiple SPs utilize common network infrastructure providers (INPs) to serve end users, as shown in Figure 1. Each end user is connected to one or multiple INPs to send or receive data packets. An INP consists of multiple processing nodes and cannot be directly controlled by the SPs. SPs are connected with the Internet backbone and interact with INPs through edge nodes. The service is bi-directional: in the uplink direction, INPs receive packets from end users and process them, then SPs collect packets (via wireless transmission) from the INPs and send them to the Internet backbone. In the downlink direction, SPs receive data packets from the Internet backbone and dispatch (via wireless transmission) the packets to INPs, the INPs then process the received packets and send them to destined end users. In this paper, we aim to develop a practical scheduling algorithm for general uplink, downlink and bi-directional systems. While there has been enormous amount of work on the problems of wireless scheduling, the issue of delayed state information has received limited attention, and only in the context of specific performance objectives such as stability. In contrast, our approach is to build upon previous work by developing a mechanism to “track” the actions of any arbitrary scheduling algorithm.

Most of the existing scheduling algorithms require instantaneous network state information (e.g., backlogs, new external arrivals). However, in practice, queueing, processing and propagation delays may be significant, making it difficult for controllers to obtain fresh network state [7]. It was shown in [16] that throughput optimality can be achieved in general network systems with observation delays. However, many ad-hoc algorithms are designed to optimize other performance metrics such as latency, fairness and power consumption. The performance of such algorithm may suffer if state information is delayed. Therefore, our goal is to design a scheme that can mimic arbitrary scheduling algorithms with desirable performance guarantees.

An intuitive naive approach is to implement the scheduling algorithm based on the delayed network state. However, since the delayed network state may be different from the real time state, the naive approach may have poor performance. Consider a simple server allocation system of one receiver and two transmitters as in Figure 2. During each time slot, 5 packets arrive at transmitter 1, while 8 packets arrive at transmitter 2 at time $t = 0, 2, 4, \ldots$. The channel data rates are constantly 10 and 8. Due to interference, the receiver can only receive packets from one transmitter in any time slot. If the receiver observes the real time backlogs of the transmitters, we can stabilize the system by always selecting the transmitter with the larger backlog. However, in practice, the receiver only knows the backlog of transmitters $D$ slots ago. Under the naive approach, the receiver always selects the transmitter with larger backlog $D$ slots ago. Since the delayed backlogs may be different from the real time backlogs, the performance suffers. For example, even if $D = 1$, it can
be shown that the policy with fresh state information has an average backlog of 2.5 packets, whereas following this naive approach would result in an average backlog of 15.5 packets. We further propose an example in which the naive approach fails to stabilize the system. The detailed analysis is presented in Section II-D.

The naive approach may fail to follow certain policies and the performance is not guaranteed. In this paper, we propose the Universal Tracking (UT) algorithm that, with delayed observation, can mimic any scheduling policy and obtain provable performance guarantees. We show in Section II-D that UT addresses the stability issues and has significantly better performance than the naive approach.

Numerous scheduling algorithms have been proposed for wireless systems. For instance, for server allocation problems with one server and multiple queues, serving the longest connected queue (LCQ) stabilizes the network [26] and achieves minimal delay if the system parameters are symmetric [5]. For load balancing problems with one dispatcher (transmitter) and multiple servers (receivers), join the shortest queue (JSQ) can stabilize the system [4] and has been widely applied in practice [8]. For more general server allocation and load balancing problems with multiple transmitters and receivers, the Maximum Weighted Matching (MWM) has been shown to be a stabilizing scheduling scheme [25], yet it is a centralized algorithm and suffers from high computational complexity [20]. An alternative method named Greedy Maximal Matching (GMM) was proposed in [9], which can be deployed in a distributed manner and is guaranteed to reach at least 50% of the maximum possible throughput. Moreover, numerous ad-hoc algorithms were designed to optimize other performance metrics including latency [14], fairness [12], [18], power consumption [3] and general network utilities [19], [29].

However, the aforementioned algorithms all require instantaneous network state information and full cooperation among nodes. The limited observability in our model can be captured by an overlay-underlay framework [23], where some underlay network components are modeled as black boxes and the controllers at overlay can only make decisions with limited underlay information. On the analysis side, numerous works focus on studying the impact imposed by observation delay [10], [30] and operation delay [28]. On the control side, several algorithms that do not require instantaneous underlay state information have been proposed [11], [21], [22]. However, the existing overlay-underlay control algorithms only aim at reaching stability. In this paper, our target is to mimic scheduling algorithms with arbitrary objectives.

To design a scheme that mimics general scheduling algorithms, a potential framework is network tracking, which tracks certain variables from the past and make decisions using the tracked information. If the controllers track the actions taken by desired scheduling policies, it is possible to mimic such policies with the tracked information. An application of the tracking framework is to track the actions taken by uncontrollable nodes in overlay-underlay networks with both stochastic dynamics [14] and adversarial dynamics [1], [2], [13], [15]. When the state information is delayed, the work of [16] constructs an emulated system to track the states of the underlay nodes and makes decisions based on the emulated system. However, existing tracking algorithms focus only on stability and cannot mimic general scheduling algorithms.

Therefore, existing scheduling methods either require instantaneous observation of network states, or only guarantee throughput optimality. In this paper, we propose the UT algorithm which, to the best of our knowledge, is the first algorithm that can mimic arbitrary scheduling algorithms with delayed state observations. We rigorously show that the performance gap between UT and the desired policy is upper bounded by the product of delay and total arrival rate.

We also analyze the naive approach that directly applies the scheduling policy with delayed state information. We propose two examples to show that the naive approach may greatly degrade the performance or even fail to stabilize the system (while UT still achieves stability). Through extensive numerical experiments, we show that UT achieves significant improvements compared with the naive approach under various settings.

The paper is organized as follows. In Section II, we study the control for uplink traffic, which can be viewed as a server allocation problem. In Section II-A we formulate the problem and introduce notation. Section II-B gives an outline of our approach and presents the details of UT. We rigorously analyze the performance of UT in Section II-C. In Section II-D, we discuss and analyze the limits of the naive approach with two examples. In Section III, we study the control for downlink traffic, which can be viewed as a load balancing problem. In Section IV, we evaluate UT and the naive approach through numerical experiments under dynamic server allocation (Section IV-A) and load balancing (Section IV-B).

II. UPLINK

We first consider scheduling in the uplink direction. The system is shown in Figure 2. Packets arrive at transmitters from external source nodes. The receivers are controllers and make decisions on selecting transmitters and serving their packets. The receivers then transmit buffered packets to external sink nodes. If we assume that the receivers clear
all received packets instantly and have no queue backlogs, the
problem can be viewed as a classic server allocation problem
[26].

A. System Model

The sets of receivers and transmitters are denoted by \( \mathcal{M} \)
and \( \mathcal{N} \), respectively. The network has multiple classes of
traffic destined to different sink nodes, with the set of classes
denoted by \( \mathcal{K} \). Each receiver may connect to one or multiple
sink nodes, and thus the topology between receivers and sinks
nodes can be arbitrary. For simplicity, we use \( s \) to denote
the aggregation of source nodes and sink nodes outside the
system. We assume that the time is slotted and the time
horizon is \( T \). At the beginning of time slot \( t \), transmitter
\( j \) has \( Q_{jk}(t) \) buffered packets of class \( k \), and receiver \( i \) has
\( Q_{ik}(t) \) buffered packets of class \( k \) that need to be transmitted
to a sink node outside the system. Transmitter \( j \) receives
\( A_{jk}(t) \) external packets that are of class \( k \). We allow \( A_{jk}(t) \)
to be non-stochastic and non-stationary over time \( t \). The wireless
channels between the transmitters and the receivers evolve dynamically under a stationary stochastic process. The
current channel data rate from transmitter \( j \) to receiver \( i \) is
\( C_{ji}(t) \) and is known to the receivers prior to the decision
making process. Receiver \( i \) then attempts to serve (receive)
\( F_{jik}(t) \) packets of class \( k \) from transmitter \( j \) based on the
transmitter state information delayed by \( D \) slots. The number
of actually served packets \( F_{jik}(t) \) may be less than \( F_{jik}(t) \)
if the buffered packets are less than \( F_{jik}(t) \). After receiving
packets from the transmitters, receiver \( i \) decides to transmit
\( F_{isk}(t) \) packets to the sink node outside the system. The
process is summarized as follows,

\[
\begin{align*}
Q_{jk}(t+1) &= \left[ Q_{jk}(t) + A_{jk}(t) - \sum_{i \in \mathcal{M}} F_{jik}(t) \right]^+ \quad (1) \\
Q_{ik}(t+1) &= \left[ Q_{ik}(t) + \sum_{j \in \mathcal{N}} F_{jik}(t) - F_{isk}(t) \right]^+ \quad (2)
\end{align*}
\]

where \([x]^+\) denotes \( \max\{0,x\} \).

The problem we aim to solve is, given an uplink traffic
scheduling policy \( \pi_u \), how to mimic the behavior of applying
\( \pi_u \) in the ideal system without observation delay.

B. Our Approach

Mathematically, the naive approach directly applies \( \pi_u \)
with the transmitter state information \( D \) slots ago, i.e.,
\[
F_u(t) = \pi_u(Q_u(t-D) + A_u(t-D), C_u(t)).
\]

We define the actions taken under \( \pi_u \) in the ideal system
without delay as \( F^\pi_u \). It is likely that \( Q_u(t-D) + A_u(t-D) \)
significantly differs from \( Q_u(t) + A_u(t) \), which makes the
action \( F_u(t) \) deviate from \( F^\pi_u(t) \). Due to the delay, it is
impractical to maintain an accurate estimate of \( Q_u(t) + A_u(t) \)
and new methods need to be introduced.

We define the backlog of applying \( \pi_u \) to the ideal system
with instantaneous network state observation as \( Q^\pi_u \). If
the receivers can maintain a relatively accurate estimate of
\( Q^\pi_u(t-D) \) and decide on \( F_u(t) \) based on the estimate, then
\( F_u(t) \) can mimic the actions in the ideal system \( D \) slots ago
and approach the ideal performance. However, just having
delayed queue information based on \( F_u \) (as opposed to \( F^\pi_u \))
is not enough. Thus, we aim to emulate an ideal system based
on \( F^\pi_u \).

More specifically, our Universal Tracking (UT) algorithm
operates in the following manners. During the first \( D \) slots,
the receivers do not have the state information of the trans-
mitters and just take actions using available information, i.e.,
\[
F_u(t) = \pi_u(\text{available information}), \quad 0 \leq t \leq D - 1.
\]

At time \( t = D \), the receivers construct an emulated system
with its initial backlogs being the same as the real system,
i.e., \( Q^e_u(0) = Q_u(0) \) (we use superscript \( e \) to denote
variables in the emulated system).

For time \( t \geq D \), the receivers compute the action that
should be taken under policy \( \pi_u \) at time \( t - D \), yet with the
current channel data rate \( C_u(t) \), i.e.,
\[
F_u(t) = \pi_u(Q^e_u(t-D) + A_u(t-D), C_u(t)), \quad t \geq D.
\]

The receivers apply \( F_u(t) \) to the real system, and the back-
logs in the real system evolve as (1) and (2). We restrict \( F_u(t) \)
not to exceed available packets in the emulated system, which
simplifies the evolution in the emulated system as follows.

\[
\begin{align*}
Q^e_{jk}(t-D + 1) &= Q^e_{jk}(t-D) + A_{jk}(t-D) - \sum_{i \in \mathcal{M}} F_{jik}(t) \quad (3) \\
Q^e_{ik}(t-D + 1) &= Q^e_{ik}(t-D) + \sum_{j \in \mathcal{N}} F_{jik}(t) - F_{isk}(t) \quad (4)
\end{align*}
\]

The receivers then use \( F_u(t) \) and \( A_u(t-D) \) to update the
emulated system and compute \( Q^e_u(t-D + 1) \) as in (3) and
(4).

The details are presented in Algorithm 1.

We use the example in Figure 2 to illustrate the UT
algorithm. We assume the initial backlogs at the transmitters
to be \( Q_u(0) = (0,0) \). The scheduling policy \( \pi_u \) we aim
to mimic is to serve the transmitter with larger backlog.
During time slot \( t = 0 \), since the receiver has no state
information of the transmitters, we assume it does not serve
either transmitter. Therefore, At the beginning of time slot
\( t = 1 \), we have
\[
Q_u(1) = Q_u(0) + A_u(0) = (5,8).
\]

During time slot \( t = 1 \), the receiver constructs an emulated
system with \( Q^e_u(0) = Q_u(0) = (0,0) \) and observes that
Algorithm 1 The UT algorithm for scheduling uplink traffic

1: **Input:** $\pi_u$, $Q_u(0)$
2: Set $Q^e_u(0) \leftarrow Q_u(0)$
3: **for** time $t \leftarrow 0, 1, \cdots, D - 1$ **do**
4: Observe $C_u(t)$
5: **Set**
6: Apply $F_u(t) \leftarrow \pi_u(\text{available information})$
7: **end for**
8: **for** time $t \leftarrow D, D + 1, \cdots, T - 1$ **do**
9: Observe $C_u(t)$
10: Observe $A_u(t - D)$
11: **Set**
12: Update the emulated system using (3) and (4)
13: Apply $F_u(t)$ to the real system
14: **end for**
15: **Output:** a sequence of actions $\{F_u(t)\}_{t=0,1,\cdots,T-1}$

$A_u(0) = (5, 8)$, which leads to $Q^e_u(0) + A_u(0) = (5, 8)$. Since transmitter 2 has larger backlog in the emulated system, the receiver chooses to serve transmitter 2, i.e., $F_u(1) = (0, 8)$. The receiver then applies $F_u(1)$ to the real system and uses it to update the emulated system, as follows.

$$
\begin{align*}
Q_u^e(2) &= Q_u(1) + A_u(1) - F_u(1) = (10, 0), \\
Q^e_u(1) &= Q^e_u(0) + A_u(0) - F_u(1) = (5, 0).
\end{align*}
$$

During time slot $t = 2$, the receiver obtains that $A_u(1) = (5, 0)$, and thus $Q^e_u(1) + A_u(1) = (10, 0)$. Therefore, the receiver chooses to serve transmitter 1, i.e., $F_u(2) = (10, 0)$. The receiver then applies $F_u(2)$ to the real system and uses it to update the emulated system, as follows.

$$
\begin{align*}
Q_u(3) &= Q_u(2) + A_u(2) - F_u(2) = (5, 8), \\
Q^e_u(2) &= Q^e_u(1) + A_u(1) - F_u(2) = (0, 0).
\end{align*}
$$

The process repeats afterwards. In Table I, we summarize the process in both the real system and the ideal system without observation delay. As can be seen from the table, the actions taken under UT exactly mimic the ideal actions (delayed by one slot), and the backlogs in the emulated system is exactly the same as the ideal system. Therefore, the emulated system mimics the ideal system. Since the receiver takes actions based on the emulated system, the actions applied to the real system are close to the ideal actions (in a delayed manner). Note that the average backlog using UT is $(5 + 8 + 10 + 0)/2 = 11.5$ packets.

**C. Performance Analysis**

As illustrated in Figure 4, to analyze the performance, we compare the actions taken in three systems: the real system, the emulated system constructed by the receiver and the ideal system which applies $\pi_u$ without observation delay. We first compare the real system and the emulated system. Starting from time $t = D$, both systems take exactly the same actions.

| $t$ | $A_u(t)$ | $Q_u(t)$ | $Q^e_u(t)$ | $F_u(t)$ | $Q^e_u(t)$ | $F^e_u(t)$ |
|-----|----------|----------|------------|----------|------------|-----------|
| 0   | (5, 8)   | (0, 0)   | (0, 0)     | (0, 0)   | (0, 0)     | (0, 0)    |
| 1   | (5, 8)   | (5, 0)   | (5, 0)     | (0, 8)   | (0, 8)     | (0, 8)    |
| 2   | (5, 8)   | (10, 0)  | (10, 0)    | (0, 0)   | (0, 0)     | (0, 8)    |
| 3   | (5, 0)   | (5, 0)   | (5, 0)     | (0, 8)   | (0, 8)     | (0, 8)    |
| 4   | (5, 8)   | (10, 0)  | (10, 0)    | (0, 0)   | (0, 0)     | (0, 8)    |

However, for the first $D$ time slots, the actions taken by the real system may be arbitrary and thus causing a performance gap. We then compare the emulated system and the ideal system. The emulated system and the ideal system both have instantaneous observation of the transmitters, with the only difference being that the channel data rates are shifted by $D$ slots. We show via stochastic coupling that both systems have the same expected backlogs. Combining the above analysis, we are able to bound the gap between the real system and the ideal system, with the emulated system as a bridge. Due to space constraints, we only outline the proof of our main results. The complete proof will be made available in a technical report [33].

**Theorem 1.** For any arrival sequence $\{A_u(t)\}_{t=0,1,\cdots,T-1}$ and any uplink scheduling policy $\pi_u$, we have

$$
E[Q_u] \leq E[Q^e_u] + D \cdot \sum_{j \in N, k \in K} \lambda_{jk}.
$$

The above results show that the gap between the performance of the real system under UT and the ideal system under $\pi_u$ is always upper bounded by the expected number of external arrival during the interval $D$. We emphasize that the uplink scheduling policy $\pi_u$ can be arbitrary, and thus our tracking algorithm has universal applicability.

**D. Comparison with Naive Approach**

As analyzed in Section II-B, under the UT algorithm, the average backlog of the example in Figure 2 is 11.5. We use this example to illustrate why the naive approach may degrade the performance compared with UT. Recall that the naive approach directly decides $F_u$ with the transmitter state information $D$ slots ago, i.e.,

$$
F_u(t) = \pi_u(Q_u(t - D) + A_u(t - D), C_u(t)),
$$

the process under the naive approach is illustrated in Table II.

From Table II, we observe that at time $t = 12$, the system has the same state as time $t = 6$, which indicates that the system circulates the states between time $t = 6$ and time $t = 11$. Therefore, the average backlog can be calculated by taking the average backlog between time $t = 6$ to time $t = 11$, which gives us $E[Q^N_{\text{naive}}] = 15.5$. Comparing the average backlogs under the two algorithms, we see that even for a simple system and a good uplink scheduling policy $\pi_u$,...
We consider a toy system with only one receiver and one transmitter as in Figure 5. The external arrivals to the transmitter are constantly packets per time slot. The delay for the receiver to observe the transmitter is $D = 2$. The scheduling policy suspends service when the transmitter is congested: the receiver serves one transmitter as in Figure 5. The external arrivals to the receiver are observed and make decisions to dispatch packets to the receivers.

UT has significant performance improvement compared to the naive approach.

Even worse, the naive approach may fail to stabilize the system. For instance, some flow control algorithms may suspend service when the transmitter is congested: the receiver serves one transmitter as in Figure 5. The external arrivals to the receiver are observed and make decisions to dispatch packets to the receivers.

UT has significant performance improvement compared to the naive approach.

Even worse, the naive approach may fail to stabilize the system. For instance, some flow control algorithms may suspend service when the transmitter is congested: the receiver serves one transmitter as in Figure 5. The external arrivals to the receiver are observed and make decisions to dispatch packets to the receivers.

In the ideal system without delay, all external packets get served once they arrive at the transmitter and the queue backlog is always zero. However, in the real system with delay, the naive approach will destabilize the system in the following manner. At time $t = 0$ and $t = 1$, the receiver does not know the state of the transmitter and chooses not to serve, making the transmitter backlog $Q(0) = 10$ and $Q(1) = 20$. At time $t = 2$, the receiver obtains the delayed information that $Q(0) = 10$, and thus serves 10 packets. However, at time $t = 3$, the receiver observes that $Q(1) = 20$, and thus suspends the service. Similarly, for $t \geq 3$, we can show that $Q(t-2) > 10$, which prevents the receiver from serving any packet, and thus destabilizes the system.

On the other hand, it is straightforward to show that by applying the UT algorithm, $Q^*(t) \equiv 0$. Therefore, the receiver always attempts to serve 10 packets. The average backlog is 20 and the system is stabilized.

The fundamental reason for the naive approach to have degraded performance is that the observed state information may be distorted by the delay. When the uplink scheduling policy $\pi_u$ is sensitive to backlogs, the performance can degrade significantly. Whereas the UT algorithm, by constructing an emulated system, tracks relatively accurate states under the uplink scheduling policy $\pi_u$ and makes decisions based on them. As we showed in Section II-C, the UT algorithm can mimic any scheduling policy $\pi$ within a guaranteed gap.

### III. Downlink

We next consider scheduling in the downlink direction. The system is shown in Figure 6. Packets arrive at transmitters from external source nodes. The transmitters are controllers and make decisions to dispatch packets to the receivers. The receivers then transmit buffered packets to external sink nodes according to some unknown policy. Note that the

| $t$ | $A_u(t)$ | $Q_u(t-1)$ | $Q_u(t-1) + A_u(t-1)$ | $F_u(t)$ |
|-----|----------|-------------|------------------------|---------|
| 0   | (5, 8)   | (0, 0)      | undefined              | (0, 0)  |
| 1   | (5, 0)   | (5, 8)      | (5, 8)                 | (0, 8)  |
| 2   | (5, 8)   | (10, 0)     | (10, 8)                | (10, 0) |
| 3   | (5, 0)   | (5, 8)      | (15, 8)                | (10, 0) |
| 4   | (5, 8)   | (0, 8)      | (10, 8)                | (10, 0) |
| 5   | (5, 0)   | (0, 16)     | (5, 16)                | (0, 8)  |
| 6   | (5, 8)   | (5, 8)      | (5, 16)                | (0, 8)  |
| 7   | (5, 0)   | (10, 8)     | (10, 16)               | (0, 8)  |
| 8   | (5, 8)   | (15, 8)     | (15, 8)                | (10, 0) |
| 9   | (5, 0)   | (10, 8)     | (20, 8)                | (10, 0) |
| 10  | (5, 8)   | (5, 8)      | (15, 8)                | (10, 0) |
| 11  | (5, 0)   | (0, 16)     | (10, 16)               | (0, 8)  |
| 12  | (5, 8)   | (5, 8)      | (5, 16)                | (0, 8)  |
| ... | ...      | ...         | ...                    | ...     |
downlink scheduling is significantly different from the uplink scheduling since now the controllers are the transmitters instead of receivers. If we assume that the receivers serve all received packets instantly and have no queue backlogs, the problem can be viewed as a classic load balancing problem [27].

The sets of transmitters and receivers are denoted by $\mathcal{M}$ and $\mathcal{N}$, respectively. At the beginning of time slot $t$, transmitter $i$ has $Q_{ik}(t)$ buffered packets destined to class $k$, and receiver $j$ has $Q_{jk}(t)$ buffered packets of class $k$ that need to transmit to external sink nodes. Transmitter $i$ receives $A_{ik}(t)$ external packets of class $k$. We also allow $A_{ik}(t)$ to be non-stochastic and non-stationary over time $t$. The wireless channels from transmitters to receivers evolve dynamically over time following a stationary stochastic process. We assume that transmitter $i$ knows $C_{ij}(t)$, the current channel data rate to receiver $j$, prior to transmission. Transmitter $i$ then decides to transmit $F_{ijk}(t)$ packets to receiver $j$ based on $Q_{jk}(t - D)$, where $D$ is the delay in observing receivers’ buffers. After receiving the packets, receiver $j$ attempts to transmit $B_{jnk}(t)$ packets to external sinks. We assume that $B_{jnk}(t)$ follows a stationary stochastic process, but is uncontrollable by the transmitters. The process is summarized as follows.

$$
\begin{align*}
Q_{ik}(t + 1) &= \left[ Q_{ik}(t) + A_{ik}(t) - \sum_{j \in \mathcal{N}} F_{ijk}(t) \right]^{+} \quad (5) \\
Q_{jk}(t + 1) &= \left[ Q_{jk}(t) + \sum_{i \in \mathcal{M}} F_{ijk}(t) - B_{jnk}(t) \right]^{+} \quad (6)
\end{align*}
$$

Our goal is, given a downlink traffic scheduling policy $\pi_d$, to mimic the behaviors of applying $\pi_d$ in the ideal system without observation delay.

The UT algorithm for downlink traffic is symmetric to the algorithm in Section II-B. We use subscript $d$ to denote vectors for the downlink traffic. The core idea is to let the transmitters maintain an emulated system that estimate the delayed backlogs in the ideal system $Q_d^{e}(t - D)$ and make decisions based on $Q_d^{e}(t - D)$ and $A_d(t - D)$. The transmitters update the emulated system with delayed observation of the receiver service $B(t - D)$. The evolution in the emulated system is as follows (we also restrict $F_d(t)$ not to exceed the available packets in the emulated system).

$$
\begin{align*}
Q_{ik}^{e}(t + 1) &= Q_d^{e}(t - D) + A_{ik}(t - D) - \sum_{j \in \mathcal{N}} F_{ijk}(t) \quad (7) \\
Q_{jk}^{e}(t + 1) &= \left[ Q_d^{e}(t - D) + \sum_{i \in \mathcal{M}} F_{ijk}(t) - B_{jnk}(t - D) \right]^{+} \quad (8)
\end{align*}
$$

The details are presented in Algorithm 2.

\begin{algorithm}[h]
\caption{The UT algorithm for scheduling downlink traffic}
1: \textbf{Input:} $\pi_d$, $Q_d(0)$
2: Set $Q_d^{e}(0) \leftarrow Q_d(0)$
3: for time $t \leftarrow 0, 1, \ldots, D - 1$ do
4: \hspace{1em} Observe $C_d(t)$
5: \hspace{1em} Set $F_d(t) \leftarrow \pi_d(\text{available information})$
6: \hspace{1em} Apply $F_d(t)$ to the real system
7: end for
8: for time $t \leftarrow D, D + 1, \ldots, T - 1$ do
9: \hspace{1em} Observe $C_d(t)$
10: \hspace{1em} Observe $A_d(t - D)$ and $B(t - D)$
11: \hspace{1em} Set $F_d(t) \leftarrow \pi_d(Q_d^{e}(t - D) + A_d(t - D), C_d(t))$
12: Update the emulated system using (7) and (8)
13: Apply $F_d(t)$ to the real system
14: end for
15: \textbf{Output:} a sequence of actions $\{F_d(t)\}_{t=0,1,\ldots,T-1}$
\end{algorithm}

Similar to Theorem 1, we show that Theorem 2. For any arrival sequence $\{A_d(t)\}_{t=0,1,\ldots,T-1}$ and any downlink scheduling policy $\pi_d$, we have

$$
\mathbb{E}[Q_d] \leq \mathbb{E}[Q_d^{e}] + D \cdot \sum_{i \in \mathcal{M}, k \in \mathcal{K}} \lambda_{ik}.
$$

Similar to Theorem 1, Theorem 2 shows that the gap between the performance of the real system under UT and the ideal system under $\pi_d$ is always upper bounded by the external arrival rates during the delayed durations.

IV. NUMERICAL EXPERIMENTS

A. Uplink

We consider a system of uplink direction with one receiver and ten transmitters. Only one transmit channel can be activated during a time slot. At the beginning of each time slot, external packets arrive at receivers. The receiver then observes the channel data rates and select a transmitter to serve its buffered packets. Such systems are called dynamic server allocation systems. It is shown in [26] that if the channel data rates are binary, the throughput optimality is obtained by always selecting the connected transmitter with the longest queue (LCQ). From simulation, we find that LCQ performs well for more general settings, and is thus an ideal
uplink scheduling policy to mimic. However, LCQ requires instantaneous observation of the transmitter backlogs, which may be unrealistic in practice. The system parameters are shown in Figure 7.

The receiver obtains the backlog information cyclically from transmitter 1 to transmitter 10. The external arrival processes to the transmitters are mixed. The arrival to transmitter $i = 1, 2, 3, 4, 5$ is of Poisson distributions with rate $\lambda_i$’s. While for transmitter $i = 6, 7, 8, 9, 10$, the arrival is not stationary over time, but according to the annotated arrival sequences. The receiver and the transmitters are connected with the annotated probabilities, and the channel data rates are 100 once connected. We assume that the data rates between the receiver and the sink nodes are large enough so that all packets are immediately cleared once they arrive at the receiver. The scheduling policy $\pi$ we track is the LCQ policy. In both UT and the naive approaches, during the first ten slots, the receiver decides its action based on the available information (i.e., queue backlogs of some transmitters). The results are shown in Figure 8.

Since we assume instantaneous observation when implementing LCQ, its performance serves as a lower bound and may not be achievable in the actual system. From the simulation results, we can see that higher observation delay downgrades the performance for both UT and the naive approach. However, even when $D = 10$, UT significantly outperforms the naive approach.

B. Downlink

We consider a system similar to Figure 7, but the data packets now only flow in the downlink direction: external packets arrive to transmitters, the transmitters transmit packets to the receivers, and the receivers serve the packets. During each time slot, the transmitter needs to select one of the receivers to dispatch buffered packets. Such scheduling problems are called load balancing problems, and a throughput optimal policy is known to be joining the shortest queue (JSQ) [27]. We consider a load balancing problem as shown in Figure 9.

The transmitter obtains the state information of the receivers in a similar manner to Section IV-A. The external arrivals to the transmitter are Poisson distributed with rate $\lambda = 15$. The transmitter and the receivers are connected with annotated probabilities, and the channel data rates are 100 once connected. The service process from receiver $j$ to end users is of uniform distribution with the annotated rate $\mu_j$. The results are shown in Figure 10.
similar to the simulation in Section IV-A, JSQ has instantaneous observation of the receiver queues and only serves as a lower bound for comparison. The results also show that smaller observation delay help improve the performance, and UT significantly outperforms the naive approach.

V. CONCLUSION

In this paper, we focus on mimicking arbitrary scheduling policies in wireless networks with delayed state information. We propose the UT algorithm that can mimic any scheduling policy, and show that the gap between UT and the desired policy (assuming instantaneous observation) is upper bounded by a constant. Numerical experiments validate our conclusion and show that UT has significantly better performance under various settings.

A potential direction is to analyze the system with adversarial dynamics (e.g., the external arrivals evolve dynamically according to our actions to maximize attack). Another possible direction is to coordinate with the policies inside the INPs and extend the framework into multi-hop settings.

ACKNOWLEDGEMENT

This work was supported by NSF grants CNS-1907905 and CNS-1735463 and by ONR grant N00014-20-1-2119.

REFERENCES

[1] Matthew Andrews and Lisa Zhang. Scheduling over nonstationary wireless channels with finite rate sets. In IEEE INFOCOM 2004, volume 3, pages 1&ndash;1704. IEEE, 2004.

[2] Matthew Andrews and Lisa Zhang. Scheduling over a time-varying user-dependent channel with applications to high-speed wireless data. Journal of the ACM (JACM), 52(5):809–834, 2005.

[3] Seung Jun Baek, Gustavo De Veciana, and Xun Su. Minimizing energy consumption in large-scale sensor networks through distributed data compression and hierarchical aggregation. IEEE Journal on selected Areas in Communications, 22(6):1130–1140, 2004.

[4] Robert D Foley and David R McDonald. Join the shortest queue: stability and exact asymptotics. Annals of Applied Probability, pages 569–607, 2001.

[5] Anand Ganti, Eytan Modiano, and John N Tsitsiklis. Optimal transmission scheduling in symmetric communication models with intermittent connectivity. IEEE Transactions on Information Theory, 53(3):998–1008, 2007.

[6] Junjie Geng. Dci-nacc: flow scheduling and congestion control based on programmable data plane in high-performance data center networks. The International Journal of Advanced Manufacturing Technology, pages 1–13, 2022.

[7] AB Goldstein, AA Zarubin, AV Oufnienko, VS Elgin, and IA Belozertsev. Synchronization of delay for os in lte. In 2018 Systems of Signal Synchronization, Generating and Processing in Telecommunications (SYNCHRONINFO), pages 1–4. IEEE, 2018.

[8] Varun Gupta, Mor Harchol Balter, Karl Sigman, and Ward Whitt. Analysis of join-the-shortest-queue routing for web server farms. Performance Evaluation, 64(9-12):1062–1081, 2007.

[9] Jaap-Henk Hoepman. Simple distributed weighted matchings. arXiv preprint cs/0410047, 2004.

[10] Matthew Johnston and Eytan Modiano. Controller placement in wireless networks with delayed csi. IEEE/ACM Transactions on Networking, 25(3):1775–1788, 2017.

[11] Nathaniel M Jones, Georgios S Paschos, Brooke Shrader, and Eytan Modiano. An overlay architecture for throughput optimal multipath routing. IEEE/ACM Transactions on Networking, 25(5):2615–2628, 2017.

[12] David Julian, Mung Chiang, Daniel O’Neill, and Stephen Boyd. Qos and fairness constrained convex optimization of resource allocation for wireless cellular and ad hoc networks. In Proceedings. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies, volume 2, pages 477–486. IEEE, 2002.

[13] Qingkai Liang and Eytan Modiano. Minimizing queue length regret under adversarial network models. Proceedings of the ACM on Measurement and Analysis of Computing Systems, 2(1):1–32, 2018.

[14] Qingkai Liang and Eytan Modiano. Optimal network control in partially-controllable networks. In IEEE INFOCOM 2019-IEEE Conference on Computer Communications, pages 397–405. IEEE, 2019.

[15] Qingkai Liang and Eytan Modiano. Optimal network control with adversarial uncontrollable nodes. In Proceedings of the Twentieth ACM International Symposium on Mobile Ad Hoc Networking and Computing, pages 101–110, 2019.

[16] Bai Liu and Eytan Modiano. Optimal control for networks with unobservable malicious nodes. Performance Evaluation, 151:102230, 2021.

[17] Bai Liu, Qiaomin Xie, and Eytan Modiano. Reinforcement learning for optimal control of queuing systems. In 2019 57th Annual Allerton Conference on Communication, Control, and Computing (Allerton), pages 601–408. IEEE, 2014.

[18] Ravi Mazumdar, Lorne G Mason, and Christos Douligeris. Fairness in network optimal flow control: optimality of product forms. IEEE Transactions on communications, 39(5):775–782, 1991.

[19] Michael J Neely, Eytan Modiano, and Chih-Ping Li. Fairness and optimal stochastic control for heterogeneous networks. IEEE/ACM Transactions On Networking, 16(2):396–409, 2008.

[20] Christos H Papadimitriou and Kenneth Steiglitz. Combinatorial optimization: algorithms and complexity. Courier Corporation, 1998.

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

[22] Anurag Rai, Rahul Singh, and Eytan Modiano. A distributed algorithm for throughput optimal routing in overlay networks. In 2019 IFIP Networking Conference (IFIP Networking), pages 1–9. IEEE, 2019.

[23] 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.

[24] Joshi Sujata, Sarkar Sohag, Dewan Tanu, Dharmani Chintant, Purohit Shubham, and Gandhi Sumit. Impact of over the top (ott) services on telecom service providers. Indian Journal of Science and Technology, 8(54):145–160, 2015.

[25] I Tassiulas and A Ephremides. Stability properties of constrained queuing systems and scheduling policies for maximum throughput in multihop radio networks. IEEE Transactions on Automatic Control, 37(12):1936–1948, 1992.

[26] Leandros Tassiulas and Anthony Ephremides. Dynamic server allocation to parallel queues with randomly varying connectivity. IEEE Transactions on Information Theory, 37(6):1455–1467, 1991.

[27] Mark van der Boor, Sem C Borst, Johan SH van Leeuwaarden, and Debankur Mukherjee. Scalable load balancing in networked systems: A survey of recent advances. arXiv preprint arXiv:1806.05444, 2018.

[28] Chang-Heng Wang, Siva Theja Magluru, and Tara Javidi. Heavy traffic queue length behavior in switches with reconfiguration delay. In IEEE INFOCOM 2017-IEEE Conference on Computer Communications, pages 1–9. IEEE, 2017.

[29] Yufang Xi and Edmund M Yeh. Distributed algorithms for spectrum allocation, power control, routing, and congestion control in wireless networks. In Proceedings of the 8th ACM international symposium on Mobile ad hoc networking and computing, pages 180–189, 2007.

[30] Yipeng Yang, Robert Buche, Mos-Hsiung Chang, and Vahid Ramezani. Power control for mobile communications with delayed state information in heavy traffic. In Proceedings of the 45th IEEE Conference on Decision and Control, pages 1923–1929. IEEE, 2006.

[31] Givon Yenihayat and Ezhan Karaşan. Analytical derivation of downlink data rate distribution for 5g hetnets with cell-edge located small cells. arXiv preprint arXiv:1611.01742, 2016.

[32] ZaiChen Zhang and Victor OK Li. Router-assisted layered multicast. In Proceedings. ICC 2002 (Cat. No. 02CH37333), pages 1–4. IEEE, 2002.