Cellular Network Traffic Scheduling with Deep Reinforcement Learning

Sandeep Chinchali [1], Pan Hu [2], Tianshu Chu [3], Manu Sharma [3], Manu Bansal [3], Rakesh Misra [3]
Marco Pavone [4] and Sachin Katti [1,2]
[1] Department of Computer Science, Stanford University
[2] Department of Electrical Engineering, Stanford University
[3] Uhana, Inc.
[4] Department of Aeronautics and Astronautics, Stanford University
{csandeep, panhu, pavone, skatti}@stanford.edu, {tchu, manusharma, manub, rakesh}@uhana.io

Abstract

Modern mobile networks are facing unprecedented growth in demand due to a new class of traffic from Internet of Things (IoT) devices such as smart wearables and autonomous cars. Future networks must schedule delay-tolerant software updates, data backup, and other transfers from IoT devices while maintaining strict service guarantees for conventional real-time applications such as voice-calling and video. This problem is extremely challenging because conventional traffic is highly dynamic across space and time, so its performance is significantly impacted if all IoT traffic is scheduled immediately when it originates. In this paper, we present a reinforcement learning (RL) based scheduler that can dynamically adapt to traffic variation, and to various reward functions set by network operators, to optimally schedule IoT traffic. Using 4 weeks of real network data from downtown Melbourne, Australia spanning diverse traffic patterns, we demonstrate that our RL scheduler can enable mobile networks to carry 14.7% more data with minimal impact on existing traffic, and outperforms heuristic schedulers by more than 2×. Our work is a valuable step towards designing autonomous, "self-driving" networks that learn to manage themselves from past data.

Introduction

Can learning algorithms help with optimally scheduling traffic in future computer networks? A central problem in computer networks, ranging from internet data centers to wireless cellular deployments, is how to best deliver traffic with widely different data demands, sensitivities to delay, and scheduling priorities. While today's networks are typically used for real-time, delay sensitive traffic such as video-streaming or web-browsing, the advent of Internet of Things (IoT) devices such as smart homes will require data-intensive sensor updates that are more tolerant to delay.

In this paper, we focus on a specific, practically-motivated scheduling problem. We focus on mobile networks, which are increasingly required to deliver a new class of applications, driven by IoT , that we call High Volume Flexible Time (HVFT) applications. This new class of HVFT traffic includes:

- Software and data updates to mobile IoT devices, e.g., updating maps for self-driving cars (Gerla et al. 2014) or delivery drones (Sundqvist and others 2015).

To appear, AAAI 2018

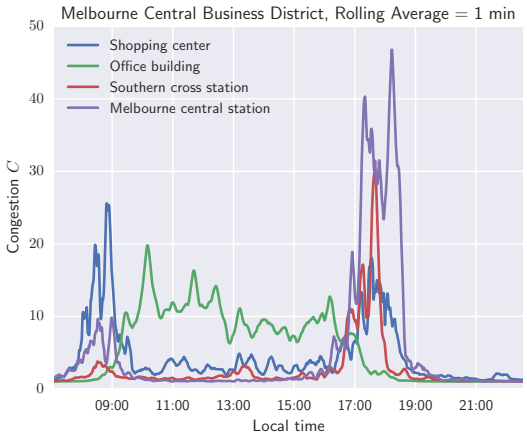


Figure 1: Time-variant congestion patterns in Melbourne.

- Large transfer of IoT sensor data to the cloud, such as energy usage measurements from a smart grid (Parikh, Kanabar, and Sidhu 2010).
- Pre-fetched ultra-high quality and bitrate video (McDonagh et al. 2011).

A common property of HVFT applications is that the mobile network operator must serve a large volume of traffic during the day but has significant flexibility in scheduling this traffic, i.e., these applications can tolerate delays on the order of a few hours up-to a day. The mobile network operator ideally would like to take advantage of this delay tolerance to schedule HVFT traffic during periods where the network is being lightly used by other delay sensitive traffic.

Intuitively, this scheduling problem can be thought of as a control problem where the control variable is the time when HVFT data is transferred and its share of network capacity and the reward function trades off the total HVFT traffic volume delivered with throughput degradation to other delay sensitive traffic. A simple solution, which has precedence in the networking community, is to have static priority classes per traffic type, i.e, real-time traffic has higher importance than delay-tolerant IoT traffic. Such an approach would unnecessarily penalize delay-tolerant traffic when networks are underloaded, and would not evolve to accommodate new

IoT traffic subclasses which have a spectrum of delay sensitivity. Ideally, network operators would like to simply declare high-level control objectives and incentivize new traffic classes to send at opportune times and rates to gracefully coexist with conventional data streams.

For the above reasons, a learning-based approach to network scheduling provides the promise of an adaptive mechanism for operators to interleave new traffic classes and thereby increase network utilization. Any viable scheduler for HVFT traffic must:

1. **Gracefully interact with other application classes**: A good HVFT scheduler should yield to conventional applications like live video streaming or web browsing that have delay requirements of less than a second. The HVFT scheduler's reward function penalizes per-minute throughput degradation to existing users of other application classes. One way to minimize throughput loss is to schedule more HVFT traffic during off-peak hours.
2. **Maximize scheduled HVFT traffic**: The reward function for an HVFT scheduler is the amount of HVFT traffic served across all cells during a day. To maximize reward, a scheduler should utilize off-peak times and opportunistically schedule traffic during transient utilization dips which often occur during peak hours.
3. **Generalize across cells**: A single metropolitan area contains hundreds of cell sites. These cells have differences in their capabilities e.g., cells with higher bandwidth have a larger data-rate, and cells with higher transmit power serve a much larger geographic area than smaller cells. Moreover, each cell has a unique spatiotemporal load pattern. Hand picking or fine-tuning parameters for each cell and day is clearly infeasible.

A key challenge is that networks exhibit non-stationary dynamics, ranging from short-term, minute-scale variation to daily commute patterns. Further, our data shows how sports events and even holidays can cause drastic distributional shifts in network patterns. Figure 1 exemplifies non-stationary dynamics in Melbourne's Central Business District, illustrating spikes in congestion at train stations during commute hours and sustained utilization in offices. To efficiently use network resources, controllers must exploit spare capacity during transient drops in congestion, but also adapt to longer trends.

RL for network management

A central challenge facing HVFT control is modeling network dynamics. Since cellular operators deploy proprietary low-level packet schedulers built by external vendors to share network resources, modeling dynamics governed by trade-secret schedulers is infeasible. Instead, we seek a control policy that optimizes rewards directly through interaction without prior knowledge of system dynamics.

The demonstrated ability of reinforcement learning (RL) and multi-task networks to learn control tasks ranging from humanoid robot walking (Benbrahim and Franklin 1997), data-center management (Mao et al. 2016), and ATARI game-playing (Mnih et al. 2013) make RL an attractive solution choice. RL is especially fitting since network operators simply want to declare high-level control objectives or reward functions, and have networks learn to manage themselves from the large amount of data they have already collected. Rather than hand-tune control policies for cells of diverse loads, manual configurations, and capacities, an RL controller can flexibly generalize to different cells.

Key challenges of an RL approach include non-Markovian time-variant dynamics, abundance of training data needed for convergence, and safe exploration in operational networks. A recent survey by RL researchers (Amodei et al. 2016) cites robustness to model drift, safe-learning, and generalization of RL algorithms as central problems to the field. The similarity between HVFT control and challenges of RL make our case study for data-driven control especially interesting.

Our Contributions

The principal contributions of our work are:

1. **Identify inefficiencies in operational networks**: We use weeks of real network data to identify significant opportunities for improving network utilization due to transient drops in congestion (Figure 1). To our knowledge, we are the first to formulate the IoT traffic scheduling problem using a learning algorithm that makes control decisions from real-time cell network measurements.
2. **Network Modeling**: Our analysis shows network dynamics are time-variant and non-Markovian. We incorporate past measurements and historical commute patterns into our state representation to re-cast the problem as a Markov Decision Process (MDP) to leverage RL methods. Experiments on live networks validate our dynamics model.
3. **Adaptive RL controllers**: We devise general reward functions that allow network operators to maximize HVFT traffic with minimal degradation to conventional data streams, and allow operators to apply custom priorities for each traffic class that may vary across cells and time. Our trained RL policies are able to significantly increase network utilization (Figure 3), a potential source of substantial financial gain for operators.
4. **Data-driven simulator**: We build a realistic network simulator needed to train an RL agent, which allows us to validate controllers before real deployment and avoid unsafe exploration in functional networks.

Related work

Our work heavily applies recent developments in deep RL (Mnih et al. 2013; Silver et al. 2014), which draw inspiration from several other approximate RL methods (Konidaris, Osentoski, and Thomas 2011; Kakade and Langford 2002). Three types of classical RL methods are commonly used to determine control policies that map states to actions: Q-learning (Szepesvári 2010), policy gradient (Sutton et al. 1999), and actor-critic methods (Konda and Tsitsiklis 1999). These methods typically assume a discrete action space

which, in particular, simplifies the search for control actions. However, in HVFT control, it is more natural to consider a continuous traffic rate as the control decision. For this case, recently developed deterministic policy gradient methods allow one to directly learn and enact a deterministic instead of a stochastic policy (Silver et al. 2014). We use the Deep Deterministic Policy Gradient (DDPG) algorithm (Lillicrap et al. 2015) to train deep neural networks for actor and critic network estimation using *experience replay* (Lin 1992).

Prior studies have used RL or inverse RL for resource allocation in other systems, such as electricity grid management (Reddy and Veloso 2011), traffic signal control (Chu, Qu, and Wang 2016), stock market bidding (Nevmyvaka, Feng, and Kearns 2006), and task scheduling for cyberphysical systems like robots (Glaubius et al. 2012; Gombolay et al. 2016).

RL has also been applied to wireless networks, but mostly classical problems such as power control (Vengerov, Bambos, and Berenji 2005) or call admission control and routing (Bhorkar et al. 2012; Marbach, Mihatsch, and Tsitsiklis 1998). A recent application of RL to job scheduling in internet data centers is *deepRM* (Mao et al. 2016). The fundamental difference is that *deepRM* assumes jobs arrive in an i.i.d.-manner to data centers, while we use weeks of real network traces to model more stochastic, time-variant dynamics inherent to wireless networks.

Network Data

Network traces were collected in cooperation with a major operator, spanning 4 weeks of data from 10 diverse cells in Melbourne, Australia. Anonymized user data is used to calculate user and cell level performance metrics. Network metrics are calculated as aggregates over a control interval $\Delta t = 1$ minute. The start of timeslot t is the control decision point for our scheduler. Though raw performance logs contain over 200 variables, key features are:

- **Average User Throughput (B):** This is the mean datarate observed by users in the cell in slot t. Many applications have a minimum throughput requirement to deliver an acceptable user experience. We use Average User Throughput as a measure of cell performance. The HVFT scheduler should not deteriorate cell performance below a given throughput limit L. Cell throughput is influenced most by cell load and quality.
- **Cell Congestion (C):** Every cell has a maximum bandwidth that is divided amongst users. As the number of users rises, the bandwidth share of each user and hence the user throughput decreases. The effective number of users in the cell, denoted by C, is a measure of cell load.
- **Average Cell Efficiency (E):** The average cell quality takes into account a number of factors that affect user throughput like the total cell bandwidth, type of cellular technology deployed, distance of users from the cell tower, and possible obstructions such as buildings and trees that decrease wireless link strength.
- **Number of connections (N):** N_t represents the total number of connections at time t. Note that many of the connected devices may not be actively downloading data and hence may not contribute significantly to cell load.
- **Traffic volume (V):** The control action that the HVFT scheduler takes at each time t is the traffic rate a_t for IoT data, i.e., the fraction of the time-slot in which IoT data will be scheduled. Every time the HVFT controller is active it schedules traffic to M IoT devices. For simplicity, we assume a constant action multiplier M. Dynamically optimizing multiplier M is a second level of optimization that may further improve performance. The volume of IoT data transacted in time t is, hence, simply $a_t B_t M \Delta t$. Also, the total data downloaded by all users in the cell at time t is denoted by V_t. We will look at the fraction of IoT data scheduled, i.e., $\frac{\sum a_t B_t M \Delta t}{\sum V_t}$ over a day as a key performance metric of the HVFT scheduler.

The following table summarizes our terminology.

Variable	Description
B_t	Throughput (Kilobits/sec)
C_t	Congestion (Unitless)
N_t	Num. Users (Unitless)
E_t	Spectral Efficiency (bits/sec)
V_t	Traffic Volume (Kilobits)
$a_t \in [0, 1]$	IoT traffic rate (Control Action)
L	Throughput limit (Kilobits/sec)
M	Action multiplier (Unitless)

The scheduling algorithm (RL agent) will reside at the cell tower, where it can use a centralized view of congestion C, number of users N, and other relevant cell metrics to make informed control decisions. Practically, such an architecture will allow the controller to leverage established methods used today to distinguish conventional and IoT traffic by monitoring HTTP request headers for source IP addresses and checking TCP packets to see if they are from IoT sensors or conventional devices such as cell-phones. Further, the controller can be robust to applications which might want to misrepresent whether they are conventional or IoT traffic, since applications cannot lie on the TCP level as the destination has to be true otherwise the packet will not reach the desired endpoint.

Data-driven Network Model

We now formalize the *HVFT-RL* problem. RL formulates stochastic uncertainty within the framework of Markov decision processes (MDP) (Bellman 1957), and learns a control policy based on past observations of transition data (Sutton and Barto 1998). An MDP is a controlled stochastic process defined by state space $\mathcal{S}$, action space $\mathcal{A}$, transition probability function $\mathbb{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ and reward function $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$. We now represent HVFT-RL as a discrete-time, continuous state and action space MDP, by defining state $s \in \mathcal{S}$, action $a \in \mathcal{A}$, transition dynamics $\mathbb{P}$, and the reward function R.

Network state and action

We consider a single cell for daily HVFT control with a planning horizon of T minutes for variable working hours.

Each time step lasts a duration $\Delta t = 1$ minute. The horizon T is a single day to allow for episodic learning in RL and was chosen since IoT traffic (such as software updates to drones) typically has a slack of several hours, not days, to be sent. Thus, a finite horizon is natural to 'reset' the system and not allow IoT jobs to be scheduled too far in the future.

At each time t, the *current* network state is measured as $S_t = [C_t, N_t, E_t]$, where C is the congestion metric, N is the number of sessions, and E is the cell efficiency. To allow the RL agent to leverage useful temporal features and stochastic forecasts, we represent the full network state as

$$s_t = [S_t, \phi(S_0, \ldots, S_t, t, T)], \tag{1}$$

where S_t is the current state, and ϕ is a *temporal feature mapping* function for extracting relevant information from past measurements, the current time index, and control horizon T. To simplify notation, we define $\phi_t := \phi(S_0, \ldots, S_t, t, T)$ as the extracted temporal features.

To provide the RL agent information about the time it has left to schedule traffic, we add current time index t to ϕ and add a fractional "horizon left" signal $H = \frac{T-t}{T}$ to accommodate different problem horizons. Temporal feature extractor ϕ_t is extremely general and can be optimized independently of the RL agent to incorporate stochastic timeseries forecasts from a variety of methods such as feed-forward neural networks, Long Short Term Memory (LSTM) networks (Hochreiter and Schmidhuber 1997), or traditional forecasters such as Autogressive Integrated Moving Average (ARIMA) techniques (Hyndman and Athanasopoulos 2014). Henceforth, $\phi_t = [S_{t-m}, \ldots, S_t, \hat{S}_{t+1}, \ldots, \hat{S}_{t+k}, t, H]$ indicates a feature extractor that uses a lookback of m past states, stochastic forecasts of k future states, current time t, and the horizon left. In our evaluation, we analyze several forms of extractor ϕ.

Our general framework allows for end-to-end learning with task-oriented features if we *directly* add historical network state to the current network state. Then, the RL agent's neural network can automatically learn key features for implicit forecasting in order to maximize its reward. However, separately optimizing forecasts in ϕ_t offers several practical benefits. Network operators can preserve privacy of users and simply provide anonymized cell forecasts to an external company building the controller. Then, an RL agent can ingest these forecasts and optimize for the control objective without ever accessing private data. Further, a separate forecaster is more general since it can be reused for other control applications that rely on future network state other than IoT scheduling.

Each control $a_t \in [0, 1]$ is a rate at which HVFT traffic can be served *on top of* conventional traffic, representing the fraction of control interval ΔT occupied by HVFT . An action $a_t = 0$ indicates no HVFT traffic is served, which allows the RL agent to yield to conventional traffic scheduling during congestion.

Time-variant transition dynamics

In an MDP, system dynamics f are Markovian, *i.e.*, $s_{t+1} = f(s_t, a_t, \epsilon_t)$ where ϵ_t is uncontrollable noise. We now formalize f for each component of *current, controlled* network state $S_t = [C_t, N_t, E_t]$, which includes the effect of IoT traffic. A controller must anticipate that HVFT traffic added in time t with rate a_t is *superimposed* on top of natural time-variant congestion trends from conventional traffic. To separate dynamics of conventional from IoT traffic, we denote $\tilde{S}_t = [\tilde{C}_t, \tilde{N}_t, \tilde{E}_t]$ to be the *uncontrolled*, natural state of the network at t, which we can forecast from historical timeseries. A standard technique, used in ARIMA forecasting, is to stationarize time-variant dynamics by computing the difference between successive samples $\Delta\tilde{S}_t = \tilde{S}_{t+1} - \tilde{S}_t$.

Congestion dynamics are piecewise: if IoT traffic is introduced, it compounds to the controlled state and is affected by trends in commute dynamics (term 1, $a_t > 0$):

$$C_{t+1} = \begin{cases} \underbrace{C_t + Ma_t}_{\text{controlled state}} + \underbrace{\Delta\tilde{C}_t}_{\text{historical commute}} + \epsilon_t & \text{if } a_t > 0 \\ \underbrace{\tilde{C}_t + \Delta\tilde{C}_t}_{\tilde{C}_{t+1}} + \epsilon_t & \text{if } a_t = 0 \end{cases} \tag{2}$$

where ϵ_t is zero-mean Gaussian noise with a standard deviation appropriate for congestion patterns.

Yet, in order to allow dynamics to follow realistic commute patterns observed in data, we cannot have the controlled state compound indefinitely. Thus, if no IoT traffic is added (term 2, $a_t = 0$), congestion dynamics yield to natural patterns allowing the agent to let the network settle back to historical congestion levels for time t. If we replace C_t and $\tilde{C}_t$ with N for number of sessions, we recover the dynamics for N_t. Cell efficiency E_t is uncontrolled, depending on whether mobile IoT devices move to lower channel quality locations, and hence we simply track measured state for E_t, which completes the dynamics for all components of S_t. We observed such piecewise dynamics for controlled variables (Eq. 2) in live network experiments where we introduced short file downloads to operational networks and such controlled traffic increased congestion for the subsequent interval, but the network settled to conventional traffic patterns after download completion (as shown in Figure 2).

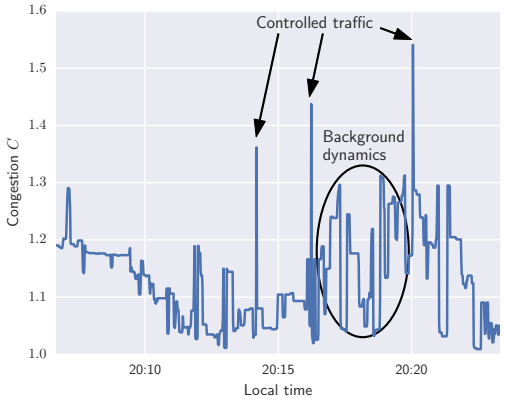


Figure 2: Congestion during our live network experiments.

Network reward function

Our control objective depends heavily on mapping state s_t to throughput B_t in order to quantify IoT traffic delivered. As the agent schedules HVFT traffic, congestion on the cell rises in subsequent states s'_t, leading IoT traffic to compete with conventional applications for fixed cell capacity, thereby decreasing the throughput share for all traffic. We use random forest (RF) regression to map state s_t to B_t using K fold cross-validation by holding out one test day and training on $K-1$ days to account for correlated throughput between successive timepoints. In the Melbourne network, our random forest regression model quite accurately fits real-world data, achieving 11.4% median error and 890 Kbps RMSE. We denote predicted throughput from the random forest model by $\hat{B}_t$.

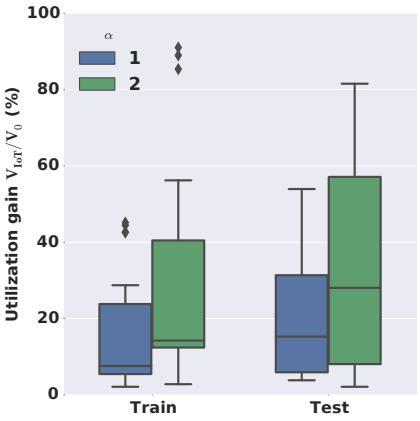


Figure 3: HVFT-RL can flexibly tradeoff IoT and regular traffic for significant gains. Outliers (diamonds) are underloaded days, such as train station weekends.

During each transition, a reward signal R is assigned to evaluate the control performance based on a weighted sum of IoT traffic served (V_t^{IoT}), byte loss to conventional applications due to introduction of IoT traffic (V_t^{loss}), and the amount of bytes served below a system-wide desired minimum throughput of L Kbps ($V_t^{\text{below limit}}$):

$$R(s_t, a_t) = \alpha V_t^{\text{IoT}} - \beta V_t^{\text{loss}} - \kappa V_t^{\text{below limit}}. \quad (3)$$

Operators can select a suite of control objectives by tuning α, β, κ to tradeoff relative importance between IoT traffic and conventional applications.

HVFT traffic volume depends on predicted throughput $\hat{B}'_t$, from the random forest model, *after* introduction of IoT content with control a_t to M IoT devices:

$$V_t^{\text{IoT}} = \hat{B}'_t M a_t \Delta t.$$

Traffic loss to a representative user sending at rate $\tilde{a}_t$ compares bytes served to that user at throughput $\hat{B}_t$ without IoT and at lower throughput $\hat{B}'_t$ after IoT traffic:

$$V_t^{\text{loss}} = \underbrace{\hat{B}_t \tilde{a}_t \Delta t}_{\text{without IoT}} - \underbrace{\hat{B}'_t \tilde{a}_t \Delta t}_{\text{with IoT}}.$$

Critically, the reward function in the above two equations uses predicted throughput $\hat{B}_t$ since it needs to evaluate throughput loss to existing users if no IoT traffic was scheduled, but in practice can only observe one cell throughput which is the joint realization of both IoT and conventional traffic. The random forest prediction model is necessary to evaluate the throughput impact $\hat{B}'_t$ of several possible control actions to compare to natural network throughput when there is no IoT .

The final term penalizes bytes served only if post IoT throughput dips below limit L *and* the agent added IoT traffic, since natural variation in throughput during peak-congested hours could fall below L despite judicious controls:

$$V_t^{\text{below limit}} = (L - \hat{B}'_t)_+ a_t \Delta t.$$

The reward is a complex function since predicted throughputs $\hat{B}_t$ depend on a random forest model, rendering closed-form analytical solutions infeasible. We use RL to find a *stationary* control policy $\mu : \mathcal{S} \rightarrow \mathcal{A}$ that maximizes cumulative discounted reward across a full day. We restrict our search to stationary policies for tractability of RL training, which is justified since the state encodes temporal features ϕ_t to capture time-variant trends. Our evaluation shows that stationary policies with accurate forecasts in ϕ_t perform quite well relative to an upper bound, non-stationary policy.

Evaluation

We now evaluate HVFT-RL on three major criteria:

1. robust performance on diverse cell-day pairs;
2. reward gap relative to an upper bound, "oracle" scheme which has access to perfect congestion forecasts; and
3. ability to use a variety of temporal features and stochastic forecasts to enhance scheduling quality.

Before describing HVFT-RL's performance on several cells, we first provide implementation details. We implement a variant of the DDPG RL algorithm with Google's TensorFlow (Abadi et al. 2016) and build a novel network simulator using openAI's *gym* environment (Brockman et al. 2016). The simulator code is publicly available at `https://bitbucket.org/sandeep_chinchali/aaai18_deeprlcell`. After an extensive literature search, we concluded comparable network simulators do not already exist, since the IoT problem is new and we worked with network operators over months to collect the novel dataset. Existing simulators are unsuitable since they focus on wireless channel models at extremely fast timescales and not city-level congestion patterns relevant to the problem addressed in this paper.

The simulator initializes an episode at the start of a day for a single cell, updates the dynamics according to Equation 2 based on the RL agent's selected IoT data rate a_t, and provides the agent a reward R_t. At the end of the day, our simulator resets to a new cell-day pair for a new training episode.

We use standard parameters for DDPG. The neural networks have two hidden layers of sizes 400 and 300. The actor and critic networks have learning rates 0.0001 and 0.001,

and L_2-norm regularization weights 0 and 0.001, respectively. The architecture of networks was tuned using validation days and control performance did not improve past two hidden layers. The discount factor is 0.99 and the minibatch size is 32.

The experiments are conducted on 27 Melbourne cell-day pairs (19 train, 8 test days). For each cell, we fit a separate RF throughput model *unknown to the agent* and train DDPG using a reward function where throughput limit L is set approximately as the median of B in training data. Each training episode is a simulation of a certain cell-day pair, and we observe stable convergence within 200 episodes in all cases.

The TensorFlow RL agent only has access to state s_t and reward R_t, and is unaware of the simulator dynamics, historical commute patterns $\tilde{S}_t$, throughput predictions $\hat{B}_t$ or desired throughput limit L. Nevertheless, we see extremely stable convergence across cells of differing capacities, for several realistic scheduling policies that balance the importance of HVFT and conventional traffic.

HVFT-RL generalizes to several cell-day pairs

Figure 3 illustrates HVFT-RL creates significant traffic volume gains compared to normal network patterns. Each point in the boxplot represents performance for a certain cell-day pair for an aggressive, IoT traffic favoring policy ($\alpha = 2, \beta = 1, \kappa = 1$) or conservative policy equally weighing IoT traffic and throughput cost terms ($\alpha = 1, \beta = 1, \kappa = 1$).

Figure 3 indicates HVFT-RL flexibly responds to the operator policy of favoring IoT traffic by creating statistically larger volume gains ($V_{IoT}\,/V_0$) for $\alpha = 2$ compared to $\alpha = 1$, where V_0 is normal traffic carried during a day. The Wilcoxon paired signed rank-test p value shows the distribution of $\alpha = 1$ and $\alpha = 2$ gains per cell-day pair are indeed different at the .05 significance level ($p < .002$) (Wilcoxon 1945).

The median gain across all days for the $\alpha = 2$ policy was 14.7 %, which is extremely significant for operators. Since the cells in our work use 10 MHz of radio spectrum, costing roughly \$4.5B (Reardon 2015), a utilization gain of 14.7% means the operator is saved about \$661 million that it would have otherwise spent on acquiring new spectrum to support the additional traffic. Interestingly, the largest outlier in Figure 3 represents a Melbourne railway cell Saturday, which faces 66% less congestion compared to 8 weekdays in our data. Thus, RL correctly learns to heavily schedule IoT traffic during uncongested times.

Figure 3 illustrates the utilization gain $V_{IoT}\,/V_0$ is marginally higher on the testing set than on the training set. In accordance with established practices, we randomly selected the test-train split and ensured they both have underloaded weekend days and are representative. However, the test set had a slightly different fraction of uncongested timepoints, leading to higher gain, which was unavoidable since we had a finite dataset. As we collect more data, this gap should decrease.

Upper bound on performance

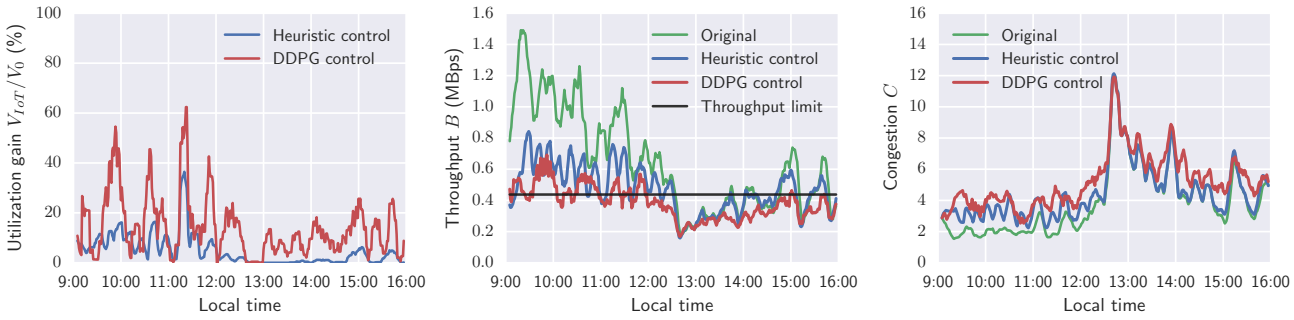
Having illustrated the performance of HVFT -RL on several cell-day pairs, we now investigate its reward relative to an "oracle" scheme with perfect, full-day congestion forecasts using offline dynamic programming (DP). The reward obtained by offline optimal DP serves as an upper bound, since it has complete knowledge of the exact MDP dynamics, congestion traces, reward function and throughput model. However, such a scheme is naturally unrealizable due to inevitable uncertainty in congestion forecasts.

A principal challenge in computing an upper bound is that our problem has both continuous state and action spaces. Further, the reward function employs a nonlinear random forest throughput model and dynamics are affected by time-variant commute patterns learned from data. Hence, analytical solutions for the upper bound are infeasible.

Instead, we closely approximate the solution to our continuous state and action space MDP by uniformly discretizing both spaces until the reward changes negligibly. We denote the discretized state space by $\bar{\mathcal{S}}$ and action space by $\bar{\mathcal{A}}$. After discretization, we compute a Q function $Q_t(s, a)$, which represents the future expected reward for a controller that takes action a in state s at time t, and acts optimally thereafter until the end of the horizon T. Given a full day's trace of T steps, we start from the end of the day at $t = T$ and iteratively solve for prior Q functions $Q_{t-1}(s, a)$ from $Q_t(s, a)$ using Bellman's equations (Bellman 1957). Then, executing an optimal policy amounts to measuring state s_t at time t and taking the optimal action given by $a_t = \arg\max_{a \in \bar{\mathcal{A}}} Q_t(s_t, a)$. Since the state and action spaces are finite, we can compute a set of Q tables for all timesteps of size $T \times |\bar{\mathcal{S}}| \times |\bar{\mathcal{A}}|$.

The complexity of computing all Q tables scales with discretization granularity as well as horizon length T. To find a suitably granular state and action space discretization beyond which the discretized MDP reward does not change beyond a predefined tolerance (chosen at 0.5% for our problem), we measured the reward for increasingly fine, uniform discretizations on a representative validation day. Figure 5(a) shows the optimal cumulative reward, which is normalized to 1.0 for the most dense state and action discretizations of $|\bar{\mathcal{S}}| = 240$ and $|\bar{\mathcal{A}}| = 60$. The cumulative reward is larger for coarse discretizations since high continuous congestion values are often binned to lower discrete congestion boundaries where the RL agent is given a high reward for the lower congestion discrete bin. As the state discretization gets finer, the difference between the true continuous congestion and discretized bin is minimal so we correctly see a lower reward accounting for the true congestion state. Even for coarse discretizations like $|\bar{\mathcal{S}}| < 50$, Figure 5(a) shows that increasing action granularity correctly leads to higher reward since the RL agent has finer control to maximize performance. Notably, for $|\bar{\mathcal{S}}| = 240$, increasing from $|\bar{\mathcal{A}}| = 40$ to $|\bar{\mathcal{A}}| = 60$ only changes the reward by 0.226%, showing the reward does not change beyond our predefined tolerance of 0.5%.

Since our numerical experiments took several hours on a modern multicore server, we used a uniform state space discretization of $|\bar{\mathcal{S}}| = 240$ and $|\bar{\mathcal{A}}| = 40$, which provides a close approximation to the continuous MDP solution.



(a) IoT traffic (b) Throughput (c) Network state

Figure 4: HVFT -RL smooths throughput variation and exploits transient dips in utilization in a Melbourne Cell.

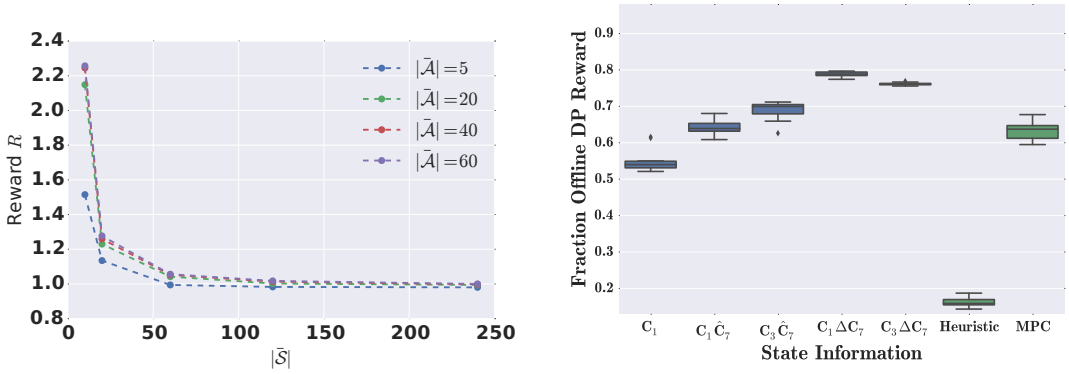


Figure 5: (Left, (a)) Optimal reward saturates for increasingly fine discretizations of our continuous MDP. (Right, (b)) HVFT -RL outperforms stochastic Model Predictive Control (MPC) and a heuristic relative to an upper bound oracle solution.

LSTM congestion forecasts

Since implementable controllers cannot be clairvoyant like the oracle solution, we build stochastic forecasts of congestion using Long Short Term Memory (LSTM) neural networks (Hochreiter and Schmidhuber 1997). LSTMs are a natural choice for our problem since their long term memory is well suited to time series forecasting with replicable commute patterns.

We trained several LSTM models to use the past $m = 9$ samples of congestion patterns and current minute t to predict a vector of the next $k = 2, 5, 7$ congestion values, $\hat{C}_k$. We also trained LSTMs to predict the changes in commute patterns ΔC_k, which are relevant to the time-variant congestion dynamics in Equation 2, to allow controllers to anticipate uncontrolled, exogeneous variation in congestion.

Using grid search to tune hyperparameters by measuring performance on a validation set, we experimented with the lookback m (number of past samples), neural network dropout to reduce overfitting, and various stacked LSTM architectures. Empirically, we observed that predicting more than a vector of 7 future congestion levels led to unacceptably high errors. Our final LSTM model for $k = 7$ congestion predictions had 14.8% median percentage error, RMSE of 0.951 for C, no dropout, and had a stacked two-layer architecture with 50 units per layer.

Comparing RL with Benchmark Controllers

We now couple LSTMs with two benchmark control schemes. After a thorough literature search, we concluded that there are no existing benchmarks for comparable problems and data, since we collected the data ourselves and posed a new control problem. Hence, we turn to *stochastic* Model Predictive Control (MPC) since it is a natural competitor to RL and has been deployed with widespread success in domains ranging from industrial process control (Camacho and Alba 2013) to cellular networking (Yin et al. 2015). Unlike some simpler forms of MPC that do not exploit probabilistic forecasts, our *stochastic* MPC approach uses an LSTM forecast of the next k congestion levels $\hat{C}$ to compute a sequence of Q-tables using DP. To ensure a fair comparison with RL, we used the same LSTM model as the RL agent with the same horizon $k = 7$. MPC implements only the first optimal action given by DP, measures the subsequent next state, and re-computes a new controller using an updated LSTM forecast in a receding horizon manner. Unlike fast evaluation of an RL agent's neural network policy, MPC is extremely computationally expensive for our task since we must compute $k \times |\bar{\mathcal{S}}| \times |\bar{\mathcal{A}}|$ finely discretized

Q-tables at each timestep t.

A simple heuristic policy consists of a scheduler that uses LSTM congestion forecasts and the throughput model to predict future throughputs $\hat{B}_k$ and choose an action proportional to the "headroom" $(\hat{B}_k - L)_+$ or an action of zero if natural throughput variation is already predicted to be below the desired minimum limit of L. Though simple, discussions with network operators indicate such approaches are common techniques used in the field.

Rich temporal information enhances scheduling

In this section, we combine offline DP, LSTM forecasts, and our benchmark controllers to investigate how the quality of temporal information affects performance. Figure 5(b) shows how various schemes perform on a representative testing day relative to the offline DP solution. RL schemes (in blue) use temporal feature extractors ϕ ranging from extremely simple such as the past m states to more complex schemes involving the past m states augmented with an LSTM forecast of k future congestion values or commute deltas. In the plot label, the variable C_m indicates the past m states, and ΔC_k or $\hat{C}_k$ indicate LSTM *future* forecasts for k steps. In addition, we include temporal information such as the current time t and the normalized horizon left $H \in [0, 1]$ for all RL agents.

Figure 5(b) shows the performance of the RL agent (in blue) relative to an oracle solution steadily increases as LSTM forecasts are included into the overall state vector, reaching just below 80% of the upper bound for feature set $C_1 \Delta C_7$. This promising result suggests that, in practice, network operators should choose a feature set using one past state and a high-quality longer LSTM forecast of 7 steps for best performance. Notably, Figure 5(b) indicates performance marginally drops if we incorporate more *past* states like $C_3 \Delta C_7$, since the RL agent has too large a state space to efficiently learn. We experimented with a wide array of past states m and horizons k, but those shown are most representative. In particular, configurations with more past states like $m = 5$ or shorter LSTM horizons $k = 2, 5$ performed worse than the pictured results and could not be displayed due to space limits.

The heuristic benchmark (green) performs extremely poorly since it does not directly optimize the complete reward function and simply schedules proportionally to throughput limit L. To ensure a fair comparison, stochastic MPC (green) uses the same LSTM model as the RL agent for forecasts of congestion C and performs fairly well at about 62% of the optimal solution. We believe the RL agent outperforms MPC since MPC is extremely sensitive to throughput forecasts, while RL can implicitly weight uncertainty in forecasts with past state measurements in its neural network control policy based on iteratively optimizing the reward function in its training procedure. Overall, our results show HVFT -RL can exploit rich temporal information from LSTM forecasts to perform quite well in the context of an offline DP oracle and realistic benchmarks.

HVFT -RL learns interpretable control policies

We now visualize HVFT -RL's control policies on a median load Melbourne cell. We do not compare with the MPC benchmark for this long day, since we wish to visualize the continuous solution and it was too computationally expensive to compute a finely discretized MPC policy. Figure 4(a) illustrates HVFT -RL schedules $2.06\times$ as much IoT traffic than the heuristic, only backing off with $a_t = 0$ during peak lunch congestion hours. As a result, Figure 4(b) shows DDPG correctly smooths throughput variation compared to original levels in order to effectively use excess cell resources, especially before noon. We see DDPG is more aggressive than the heuristic, exploiting natural drops in congestion throughout the day (Figure 4(c)), illustrating why it will do significantly better than simple schemes operators might want to deploy such as only scheduling IoT traffic during off-peak hours. Further, it sometimes causes throughput to drop below limit L, because its reward function tolerates throughput degradation to maximize IoT traffic due to similar weights α, β, κ. It is infeasible to always strictly keep throughput above L since even natural variation breaks this threshold, seen especially during peak hours.

The principal benefits of RL above the heuristic are that RL can implicitly capture temporal patterns to better forecast transient utilization dips. Unlike the heuristic's proportional scheme, RL dynamically optimizes the relative importance of IoT traffic to directly maximize reward.

Conclusion

In this paper, we analyze weeks of historical network data across heterogeneous cells to model time-variant dynamics and build a faithful network simulator. We then show that RL controllers allow operators to simply express high-level reward functions and automatically generate adaptive controllers that create significant gains. Such an adaptive approach is necessary since modern networks are constantly evolving due to shifts in city-scale commute patterns from urban population growth, coupled with new IoT datastreams. Such trends will in turn lead to operators deploying more cells with expensive spectrum, making our RL approach especially appealing to generalize to new cells.

Future work centers around deploying an RL agent in an operational network to validate gains. We also plan to differentiate global from cell-specific features learned by our deep neural networks.

Acknowledgements

We gratefully acknowledge access to network data from Telstra Corporation Ltd.

References

Abadi, M.; Barham, P.; Chen, J.; Chen, Z.; Davis, A.; Dean, J.; Devin, M.; Ghemawat, S.; Irving, G.; Isard, M.; et al. 2016. Tensorflow: A system for large-scale machine learning. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation 2016. Savannah, Georgia, USA*.

Amodei, D.; Olah, C.; Steinhardt, J.; Christiano, P.; Schulman, J.; and Mané, D. 2016. Concrete problems in AI safety. *CoRR* abs/1606.06565.

Bellman, R. 1957. A markovian decision process. *Journal of Mathematics and Mechanics* 679–684.

Benbrahim, H., and Franklin, J. A. 1997. Biped dynamic walking using reinforcement learning. *Robotics and Autonomous Systems* 22(3-4):283–302.

Bhorkar, A. A.; Naghshvar, M.; Javidi, T.; and Rao, B. D. 2012. Adaptive opportunistic routing for wireless ad hoc networks. *IEEE/ACM Transactions On Networking* 20(1):243–256.

Brockman, G.; Cheung, V.; Pettersson, L.; Schneider, J.; Schulman, J.; Tang, J.; and Zaremba, W. 2016. Openai gym. *arXiv preprint arXiv:1606.01540*.

Camacho, E. F., and Alba, C. B. 2013. *Model predictive control*. Springer Science & Business Media.

Chu, T.; Qu, S.; and Wang, J. 2016. Large-scale traffic grid signal control with regional reinforcement learning. In *American Control Conference*, 815–820. IEEE.

Gerla, M.; Lee, E.-K.; Pau, G.; and Lee, U. 2014. Internet of vehicles: From intelligent grid to autonomous cars and vehicular clouds. In *2014 IEEE World Forum on Internet of Things*, 241–246. IEEE.

Glaubius, R.; Tidwell, T.; Gill, C.; and Smart, W. D. 2012. Real-time scheduling via reinforcement learning. *arXiv preprint arXiv:1203.3481*.

Gombolay, M.; Jensen, R.; Stigile, J.; Son, S.-H.; and Shah, J. 2016. Decision-making authority, team efficiency and human worker satisfaction in mixed human-robot teams. In *International Joint Conference on Artificial Intelligence*.

Hochreiter, S., and Schmidhuber, J. 1997. Long short-term memory. *Neural computation* 9(8):1735–1780.

Hyndman, R. J., and Athanasopoulos, G. 2014. *Forecasting: principles and practice*. OTexts.

Kakade, S., and Langford, J. 2002. Approximately optimal approximate reinforcement learning. In *Int. Conf. on Machine Learning*, 267–274. Morgan Kaufmann Publishers Inc.

Konda, V., and Tsitsiklis, J. 1999. Actor-critic algorithms. In *Advances in Neural Information Processing Systems*, volume 13, 1008–1014.

Konidaris, G.; Osentoski, S.; and Thomas, P. S. 2011. Value function approximation in reinforcement learning using the fourier basis. In *Proc. AAAI Conf. on Artificial Intelligence*, volume 6, 7.

Lillicrap, T.; Hunt, J.; Pritzel, A.; Heess, N.; Erez, T.; Tassa, Y.; Silver, D.; and Wierstra, D. 2015. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.

Lin, L.-J. 1992. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning* 8(3-4):293–321.

Mao, H.; Alizadeh, M.; Menache, I.; and Kandula, S. 2016. Resource management with deep reinforcement learning. In *ACM Workshop on Hot Topics in Networks*, 50–56. ACM.

Marbach, P.; Mihatsch, O.; and Tsitsiklis, J. N. 1998. Call admission control and routing in integrated services networks using reinforcement learning. In *IEEE Conf. on Decision and Control*, volume 1, 563–568. IEEE.

McDonagh, P.; Vallati, C.; Pande, A.; Mohapatra, P.; Perry, P.; and Mingozzi, E. 2011. Quality-oriented scalable video delivery using h. 264 svc on an lte network. In *International Symposium on Wireless Personal Multimedia Communications*.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; and Riedmiller, M. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

Nevmyvaka, Y.; Feng, Y.; and Kearns, M. 2006. Reinforcement learning for optimized trade execution. In *Int. Conf. on Machine Learning*, 673–680. ACM.

Parikh, P. P.; Kanabar, M. G.; and Sidhu, T. S. 2010. Opportunities and challenges of wireless communication technologies for smart grid applications. In *IEEE Power and Energy Society General Meeting*, 1–7. IEEE.

Reardon, M. 2015. Fcc rakes in 45 billion from wireless spectrum auction. Available at `https://www.cnet.com/news/fcc-rakes-in-45-billion-from-wireless-spectrum-auction`.

Reddy, P. P., and Veloso, M. M. 2011. Strategy learning for autonomous agents in smart grid markets. *International Joint Conference on Artificial Intelligence*.

Silver, D.; Lever, G.; Heess, N.; Degris, T.; Wierstra, D.; and Riedmiller, M. 2014. Deterministic policy gradient algorithms. In Jebara, T., and Xing, E. P., eds., *Int. Conf. on Machine Learning*, 387–395. JMLR Workshop and Conference Proceedings.

Sundqvist, L., et al. 2015. Cellular controlled drone experiment: Evaluation of network requirements. Master's thesis, Aalto University.

Sutton, R., and Barto, A. 1998. *Reinforcement learning: an introduction*. MIT Press.

Sutton, R.; McAllester, D.; Singh, S.; Mansour, Y.; et al. 1999. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems*, volume 99, 1057–1063.

Szepesvári, C. 2010. Algorithms for reinforcement learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning* 4(1):1–103.

Vengerov, D.; Bambos, N.; and Berenji, H. R. 2005. A fuzzy reinforcement learning approach to power control in wireless transmitters. *IEEE Transactions on Systems, Man, and Cybernetics* 35(4):768–778.

Wilcoxon, F. 1945. Individual comparisons by ranking methods. *Biometrics bulletin* 1(6):80–83.

Yin, X.; Jindal, A.; Sekar, V.; and Sinopoli, B. 2015. A control-theoretic approach for dynamic adaptive video streaming over http. *ACM SIGCOMM Computer Communication Review* 45(4):325–338.