Received December 30, 2018, accepted January 7, 2019, date of publication January 11, 2019, date of current version February 4, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2892046

TCP-Drinc: Smart Congestion Control Based on Deep Reinforcement Learning

KEFAN XIAO, (Student Member, IEEE), SHIWEN MAO, (Fellow, IEEE), AND JITENDRA K. TUGNAIT, (Life Fellow, IEEE)
Department of Electrical and Computer Engineering, Auburn University, Auburn, AL 36849-5201, USA

Corresponding author: Shiwen Mao (smao@ieee.org)

This work was supported in part by NSF under Grant CNS-1702957, and in part by the Wireless Engineering Research and Education Center (WEREC), Auburn University, Auburn, AL, USA.

ABSTRACT As wired/wireless networks become more and more complex, the fundamental assumptions made by many existing TCP variants may not hold true anymore. In this paper, we develop a model-free, smart congestion control algorithm based on deep reinforcement learning, which has a high potential in dealing with the complex and dynamic network environment. We present TCP-Deep ReInforcement learNing-based Congestion control (Drinc) which learns from past experience in the form of a set of measured features to decide how to adjust the congestion window size. We present the TCP-Drinc design and validate its performance with extensive ns-3 simulations and comparison with five benchmark schemes.

INDEX TERMS Congestion control, deep convolutional neural network (DCNN), deep reinforcement learning (DRL), long short term memory (LSTM), machine learning.

I. INTRODUCTION

The unprecedented growth of network traffic, in particular, mobile network traffic, has greatly stressed today's Internet. Although the capacities of wired and wireless links have been continuously increased, the gap between user demand and what the Internet can offer is actually getting wider. Furthermore, many emerging applications not only require high throughput and reliability, but also low delay. Although the brute-force approach of deploying wired and wireless links with a higher capacity helps to mitigate the problem, a more viable approach is to revisit the higher layer protocol design, to make more efficient use of the increased physical layer link capacity.

Congestion control is the most important networking function of the transport layer, which ensures reliable delivery of application data. However, the design of a congestion control protocol is highly challenging. First, the transport network is an extremely complex and large-scale network of queues. The TCP end host itself consists of various interconnected queues in the kernel. When the TCP flow gets into the Internet, it traverses various queues at routers/switches along the end-to-end path, each shared by cross-traffic (e.g., other TCP flows and UDP traffic) and served with some scheduling discipline. Significant efforts are still needed to gain good understanding of such a complex network to develop the queueing network theory that can guide the design of a congestion control protocol. Second, if following the end-to-end principle, agents at end hosts have to probe the network state and make independent decisions without coordination. The detected network state is usually error-prone and delayed, and the effect of an action is also delayed and depends on the actions of other competing hosts. Third, if to involve routers, the algorithm must be extremely simple (e.g., stateless) to ensure scalability, since the router may handle a huge amount of flows. Finally, as more wireless devices are connected, the lossy and capacity-varying wireless links also pose great challenges to congestion control design.

Many effective congestion control protocols have been developed in the past three decades since the pioneering work [1] (see Section II). However, many existing schemes are based on some fundamental assumptions. For example, early generation of TCP variants assume that all losses are due to buffer overflow, and use loss as indicator of congestion. Since such assumption does not hold true in wireless networks, many heuristics have been proposed for TCP over wireless to distinguish the losses due to congestion from that incurred by link errors. Moreover, many existing schemes assume a single bottleneck link in the end-to-end path, and the wireless last hop (if there is one) is always the bottleneck. Given the high capacity wireless links and the complex network topology/traffic conditions we have today [2], such assumptions are less likely to be true. The bottleneck could be at either the wired or wireless segment, it could move around, and there could be more than one bottlenecks. Finally, when there is a wireless last hop, some existing work [3] assumes no competition among the flows at the base station (BS), which, as shown in [4], may not be true due to coupled wireless transmission scheduling at the BS.

In this paper, we aim to develop a smart congestion control algorithm that does not rely on the above assumptions. Motivated by the recent success of applying machine learning to wireless networking problems [5], and based on our experience of applying deep learning (DR) and deep reinforcement learning (DRL) to 5G mmWave networks [6], edge computing and caching [7]–[9], and RF sensing and indoor localization [10]–[12], we propose to develop a model-free, smart congestion control algorithm based on DRL. The original methods that treat the network as a white box have been shown to have many limitations. To this end, machine learning, in particular, DRL, has a high potential in dealing with the complex network and traffic conditions by learning from past experience and extracting useful features. A DRL based approach also relieves the burden on training data, and has the unique advantage of being adaptive to varying network conditions.

In particular, we present **TCP-Drinc**, acronym for **D**eep **r**e**i**nforcement lear**n**ing based **c**ongestion control. TCP-Drinc is a DRL based agent that is executed at the sender side. The agent estimates features such as congestion window difference, round trip time (RTT), the minimum RTT over RTT ratio, the difference between RTT and the minimum RTT, and the inter-arrival time of ACKs, and stores historical data in an experience buffer. Then the agent uses a deep convolutional neural network (DCNN) concatenated with a long short term memory (LSTM) network to learn from historical data and select the next action to adjust the congestion window size. The contributions of this work are summarized as follows.

1) To the best of our knowledge, this is the first work that applies DRL to tackle the congestion control problem. Specifically, we propose a DRL based framework on (i) how to build an experience buffer to deal with the delayed environment, where an action will take effect after a delay and feedbacks are also delayed, (ii) how to handle the multi-agent competition problem, and (iii) how to design and compute the key components including states, action, and reward. We believe this framework could help to boost the future research on smart congestion control protocols.
2) The proposed TCP-Drinc framework also offers effective solutions to several long-existing problems in congestion control: delayed environment, partial observable information, and measurement variations. We apply DCNN as a filter to extract stable features from the rich but noisy measurements, instead of using EWMA as a coarse filter as in previous works. Moreover, the LSTM is utilized to handle the autocorrelation within the time-series introduced by delay and partial information that an agent senses.
3) We develop a realistic implementation of TCP-Drinc on the ns-3 [13] and TensorFlow [14] platforms. The DRL agent is developed with TensorFlow and the training and inference interfaces are built in ns-3 using TensorFlow C++. We conduct an extensive simulation study with TCP-Drinc and compare with five representative benchmark schemes, including both loss based and latency based TCP variants. TCP-Drinc achieves superior performance in throughput and RTT in all the simulations, and exhibits high adaptiveness and robustness under dynamic network environments.

TABLE 1. Notation.

Symbol	Description
$\mathbf{s}_t$	the state at time t
$\mathbf{a}_t$	the action at time t
$r(\mathbf{s}_t, \mathbf{a}_t)$	the reward for given state and action at time t
$V(\mathbf{s}_t, \mathbf{a}_t)$	the value function
$Q^\pi(\boldsymbol{s}_t, \boldsymbol{a}_t)$	the Q function for discounted accumulated reward
R_t	the discounted accumulated reward at time t
$\pi(t)$	the policy at time t
$\boldsymbol{\omega}$	the weights of the Deep Q-Network (DQN)
$J(\boldsymbol{\omega})$	the loss function for weight training
y_t	the target value in the loss function
τ_{RTT}	the round trip time
$\hat{\tau}_p$	the minimum RTT
τ_p	the propagation delay
τ_q	the queuing delay
T	the end of the episode
L	the horizon for the impact of an action
Δw	the congestion window size difference
v_{RTT}	the minimum RTT over RTT ratio
δ_{RTT}	the difference between RTT and the minimum RTT
τ_{ACK}	the inter-arrival time of ACKs
M	the number of time slots in a state
$\mathcal{S}$	the state space
$\mathcal{A}$	the action space
$x(t)$	the sending rate at time t
$w(t)$	the congestion window size at time t
$z(t)$	the impact of an action at time t on future goodput
$U(\cdot)$	the utility function
γ	the discount factor
β	the weight in the utility function
η	the decay rate of an action's impact on future goodput
N_{pre}	the duration of the pre-training process
N_{dis}	the duration of the distributed training process

The remainder of this paper is organized as follows. We first review related work in Section II and discuss preliminaries of DRL in Section III. The system model and problem statement are presented in Section IV. The proposed TCP-Drinc design is presented in Section V, and validated with ns-3 simulations in Section VI. We conclude this paper in Section VII. The math notation is summarized in Table 1.

II. RELATED WORK

Congestion control is a fundamental networking problem, which has drawn extensive attention and has been studied over the past three decades. In this section, we review the key related work and recent progress, as classified into three categories: end-to-end, router based, and smart schemes.

A. END-TO-END SCHEMES

This class of work can be further classified into loss-based or delay-based schemes. TCP Reno [15], TCP Tahoe [1] and TCP NewReno [16] are early protocols that are loss-based. TCP Vegas is the first delay-based protocol. Currently, most computer operation systems adopt TCP Cubic [17] or Compound TCP [18], which are mainly designed to deal with large capacity RTT products.

These protocols are originally designed for the wired Internet. However, recent measurements in 3G and 4G LTE networks reveals fast variation of channel capacity [3]. In addition, Zaki *et al.* [4] show that bursty scheduling and competing traffic will also influence the RTT and capacity utilization. The authors then propose new end-to-end protocols that observe packet arrival times to infer the uncertain network dynamics (i.e., Sprout [3]) and utilize delay measurements (i.e., Verus [4]) to cope with these challenges.

Another recent progress on congestion control is in the paradigm of data center networks. Compared with general networks, the link capacity in data center networks is usually larger (e.g, Gigbits or 10s of Gigbits) and stable, and the latency is much smaller (e.g., in μs), while ACK is not sent for every packet. The Data Center TCP (DCTCP) [19] exploits ECN feedback from switches. Performance-oriented Congestion Control (PCC) proposes to continuously observe the connection between its actions and experienced performance, and to adopt actions that lead to good performance [20]. Jiang *et al.* [21] leverage sliding mode control theory to analyze and address the stability issue of quantized congestion notification (QCN) in data center Ethernet networks.

B. ROUTER BASED SCHEMES

This class of work involves switches or routers in the congestion control process. Explicit Congestion Notification (ECN) [22] is introduced as a substitution of loss as a congestion signal. In Active Queue Management (AQM) schemes, such as RED [23], CoDel [24], and MAQ [25], [26], routers mark or drop packets on incipient congestion. These approaches require modification of routers and intermediate devices, which may not be practical or may not scale up to large number of TCP flows.

C. SMART SCHEMES

With the recent success of machine learning/deep learning on image recognition, video analytics, and natural language processing, there is strong interest in applying machine learning to solving networking problems [5]. The results in [27] and [28] provide interesting insights into machine generated congestion protocols. However, these learning algorithms require offline training based on prior knowledge of the network and can only be adopted for limited situations. Xu *et al.* [29] propose an algorithm based on DRL to deal with the traffic engineering problem at intermediate nodes in the network layer. Li *et al.* [30] incorporate Q-learning into congestion control design and present QTCP. It is based on limited feature scales (discretized states) and uses Kanerva Coding, instead of a deep neural network, for Q-function approximation. The authors show considerable gains achieved by QTCP over the classical TCP NewReno scheme.

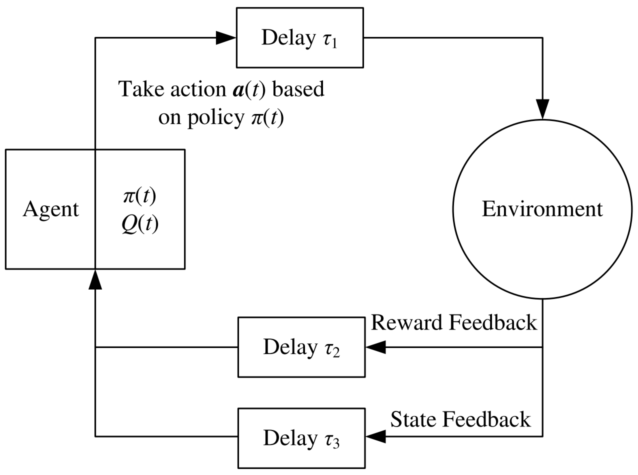


FIGURE 1. Reinforcement learning for congestion control.

III. PRELIMINARIES OF DRL

The general reinforcement learning consists of two entities: agent and environment, as shown in Fig. 1. The interactions between the two entities constantly influence the environment and trains the agent. Usually DRL is applied to solve Markov decision problems (MDP). During each episode, the agent receives a *state* tensor $\mathbf{s}_t$, takes an *action* $\mathbf{a}_t$ based on *policy* $\pi(\mathbf{s_t})$, and receives a scalar valued *reward* $r(\mathbf{s}_t, \mathbf{a}_t)$. There may be a delay τ_1 before action $\mathbf{a}_t$ starts to influence the environment, and there may also be a delay τ_2 for reward $r(\mathbf{s}_t, \mathbf{a}_t)$ to be received by the agent. Such delays are neglected in many DRL designs. However, for congestion control, such delays play an important role on the stability of the system as will be demonstrated later in this paper.

The *value function* $V(\mathbf{s}_t, \mathbf{a}_t)$ can be represented as the discounted, accumulated reward, which is given by

$$R_t = \sum_{i=t}^{T} \gamma^{i-t} \cdot r(\mathbf{s}_i, \mathbf{a}_i), \tag{1}$$

where $\gamma \in [0, 1]$ is the discount factor and T is the end of the episode. With Bellman-Ford equation, we rewrite (1) as

$$R_t = r(\mathbf{s}_t, \mathbf{a}_t) + \gamma \cdot R_{t+1}. \tag{2}$$

In general settings, *policy* $\pi(\mathbf{a}|\mathbf{s_t})$ or $\pi(\mathbf{s_t})$ generates the target action by mapping the state space to the action space: $\mathcal{S} \rightarrow \mathcal{A}$, stochastically or deterministically. For example, we can model the output action as a probability distribution over a discrete action space $\mathbf{p}(\mathbf{a})$, while the action to be taken could be either sampled with this distribution or the one with the largest probability. It is natural to use neural networks (NN) to approximate such policy and value functions. However, the approximation could be unstable, due to the strong correlation between sequential samples. In addition, the nonstationary target values could further degrade the stability. Therefore the shallow NN has a limited ability to extract important features.

DRL algorithms, first introduced in [31], are based on a similar structure but incorporate a deep neural network (DNN) to learn the value function and/or the policy function. The raw sensory data is treated as state $\boldsymbol{s}_t$ and input to the DNN. The DNN is utilized to approximate the Q-function $Q(\boldsymbol{s}_t, \boldsymbol{a}_t)$, termed Deep Q-Network (DQN), which calculates the optimal value for each possible action for given state $\boldsymbol{s}_t$. The Q value function $Q^{\pi}(\boldsymbol{s}_t, \boldsymbol{a}_t)$ for given policy π is defined as

$$Q^{\pi}(\boldsymbol{s}_t, \boldsymbol{a}_t) = \mathbb{E}[R_t | \boldsymbol{s}_t, \boldsymbol{a}_t], \tag{3}$$

where R_t is given in (1). The equation can be unrolled as

$$Q^{\pi}(\boldsymbol{s}_t, \boldsymbol{a}_t) = \mathbb{E}_{s_{t+1}}\left[r_t + \gamma Q^{\pi}(\boldsymbol{s}_{t+1}, \boldsymbol{a}_{t+1}) | \boldsymbol{s}_t, \boldsymbol{a}_t\right]. \tag{4}$$

Let $\boldsymbol{\omega}$ represent the weights of the DQN. Supposing $Q(\mathbf{s}_t, \mathbf{a}_t, \boldsymbol{\omega}) \approx Q^{\pi}(\mathbf{s}_t, \mathbf{a}_t)$, we can define a *loss function* as

$$J(\boldsymbol{\omega}) = \mathbb{E}\left[(r_t + \gamma Q(\mathbf{s}_{t+1}, \mathbf{a}_{t+1}, \boldsymbol{\omega}) - Q(\mathbf{s}_t, \mathbf{a}_t, \boldsymbol{\omega}))^2\right]. \tag{5}$$

The sum of the first two terms on the right-hand-side is defined as the *target value*, i.e., $y_t = r_t + \gamma Q(\mathbf{s}_{t+1}, \mathbf{a}_{t+1}, \boldsymbol{\omega})$.

To address the instability problem of the training process, Mnih *et al.* [31] proposed two strategies: *experience replay* and *target networks*. First, the history of transitional states and actions is stored in a buffer as *experience*. The DQN is trained with mini batches of data that are sampled randomly from the experience buffer, so as to break the correlation between sequential data samples. In fact, mini-batches are randomly drawn from the experience buffer in order to apply stochastic gradient decent (SGD) to further reduce the correlation. Second, an additional target network is introduced to calculate the target value $y_t = r_t + \gamma \cdot Q(\mathbf{s}_{t+1}, \mathbf{a}_{t+1}, \boldsymbol{\omega}')$ in the training process, whose weights $\boldsymbol{\omega}'$ are updated periodically with the running weights. The target value will become stable after certain amount of steps in the training process.

IV. SYSTEM MODEL AND PROBLEM STATEMENT

A. NETWORK ARCHITECTURE

In this paper, a very general setting of congestion control is considered, as shown in Fig. 2, where is a set of remote hosts serving a set of mobile users (in fact, the last hop could also be wired). Each end-to-end path, from a remote host to a mobile user, consists of multiple hops. The remote hosts apply a congestion window (cWnd) based protocol. The two features of the general setting are: competitions among different flows at bottleneck links and potentially multiple bottlenecks along the end-to-end path.

Most prior works assume that the last wireless hop is the single bottleneck. Under this assumption, the round-trip time (RTT) of a user, $\tau_{RTT}(t) = \tau_p + \tau_q(t)$, where τ_p is the *propagation delay* and $\tau_q(t)$ is the *queuing delay*, seems to be independent to other users, since the BS usually maintains separate queues for different users. However, even in this case, the multiple flows at the BS are still coupled due to the transmission scheduling algorithm used at the BS [4]. Furthermore, the flows may still compete with each other for transmission resources and buffer storage along the multihop path if they share a common link.

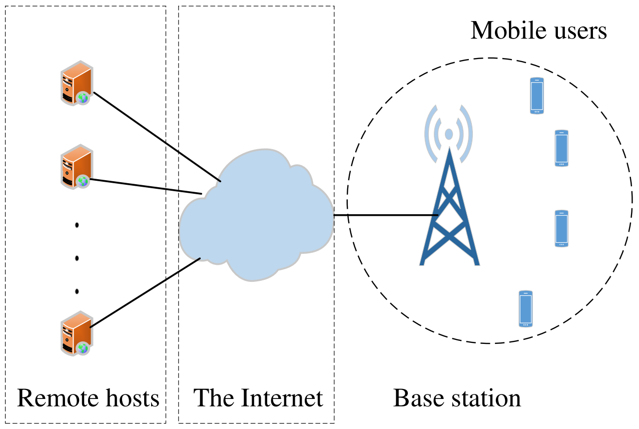


FIGURE 2. System architecture for TCP over wireless.

The general problem of congestion control should be treated as a *delayed, distributed decision-making problem*. Each sender needs to decide its own cWnd (or, sending rate). We focus on cWnd in this paper since it can be implicitly translated into sending rate according to the TCP clocking phenomenon [32]. The "delayed" feature means that the action that a sender takes will influence the bottleneck queue after a forward propagation delay; and the receiver feedback about the network condition is usually received by the sender an RTT later. If the global information of each user's strategy, network topology, and traffic condition were available, one could have made perfect decisions. But the global information is usually unavailable and the users do not cooperate on congestion control, which entails "distributed" decision making. Although these features have been considered in prior work, e.g., modeling TCP throughput as delayed differential equations [32], they all pose great challenges to a DRL based congestion control design.

B. PARTIALLY OBSERVABLE INFORMATION

In congestion control, the information each sender can acquire is partial and delayed. The measured RTT can only tell the total delay, but lacks details on the delay on each hop, on forward and backward directions, and on propagation and queueing components. One popular approach is to approximate propagation delay by the minimal RTT. In addition, the acquisition of RTT and the effect of action are both delayed. Therefore, congestion control is actually a partially observable Markov decision process (POMDP).

The problem is to develop a sender algorithm that explores the network environment and adjusts the congestion window to reliably transport data. This process is conducted by many other senders as well due to shared networking resources. Therefore, the environment one sender senses is also affected by other users' actions and is constantly changing. In this paper, we propose an integrated solution to deal with such "multi-agent" feature of the congestion control problem.

For the POMDP problem, it helps to utilize historical data to exploit the momentum and correlation in the process. However, utilizing historical data is non-trivial due to the associated time and space complexity. In machine learning, a promising solution is the recurrent neural network (RNN), which is effective for capturing the temporal, dynamic behavior in a time series.

C. BASIC ASSUMPTIONS

To make the design general, we do not assume the system is Markovian. Many TCP variants, such as [17] and [33], make control decisions based only on the current state, since the basic assumption is the next state only depends on the current state, no matter what congestion signal they use (i.e., delay or loss). In a recent work [27], the exponential average over historical delay is utilized. This is an intuitive solution to the problem of congestion control because the current state is delayed and partially observable. As discussed, the sending rate depends on the state of one RTT before, while the states of queues and latency depend on the state one propagation delay ago. In short, the Markovian assumption may be too simplistic and it will be helpful to exploit historical information for better decision making.

In real network environments, received signals are usually noisy, which can be mitigated by adopting an exponential window moving average to acquire the stable information. In this paper, we utilize a DCNN as an automatically tuned filter to extract stable information. To make the model mathematically tractable, the amount of previous states to be considered is limited to a window size of M.

We make no assumption on how the users compete for resources. The information that senders could acquire is very limited in practice. Many existing algorithms are based on certain assumptions of network conditions. For example, Sprout [3] assumes that the only dynamic component of delay is queuing delay, and more important, it is self-inflicted, meaning that it is solely determined by its sending rate and channel capacity. This assumption may be strong since the traffic flows are coupled by the transmission scheduling at the BS, and cross-traffic may also affect the queueing delay when queues and transmission capacity are shared in the wired hops. A practical scheme should not require the sender to know if the physical network is wired or wireless, how many bottlenecks are out there, and what scheduling algorithms are used at each hop. Therefore, it is better to make no extra assumptions on the physical network. We thus assume the information a sender could acquire is its own sending rate, congestion window cWnd, measured RTT, and interarrival times of received ACKs.

D. NETWORK MODEL

The traffic model adopted in this paper is based on TCP clocking [32]; the sending rate can be implicitly determined by the congestion window cWnd and RTT. CWnd is the maximum amount of TCP segments the sender can inject into the network without ACK. The network and receiver can control the sending pace by delaying or withholding ACKs. The benefit of focusing on cWnd is the reduced control action dimension.

To simplify notation, we omit the subscript i for sender i in the following presentation. Let the sender cWnd be $w(t)$. Recall the RTT is $\tau_{RTT}(t) = \tau_p + \tau_q(t)$, where τ_p is the propagation delay, and $\tau_q(t)$ is the total queuing delay incurred at one or more bottleneck links along the end-to-end path. The τ_p can be approximated by the minimum RTT within a time window. The instantaneous *sending rate*, $x(t)$, and its relationship with cWnd $w(t)$, is given by [32]

$$\int_{t-\tau_{RTT}(t)}^{t} x(\iota)d\iota = w(t). \tag{6}$$

Differentiating (6) and rearranging terms, we have [32]

$$x(t) = \frac{x(t-\tau_{RTT}(\tilde{t}))}{1+\dot{\tau}_{RTT}(t-\tau_{RTT}(\tilde{t}))} + \dot{w}(t), \tag{7}$$

where $\tilde{t} = t - \tau_{RTT}(t)$. Notice that the instantaneous sending rate at time t depends on not only the sending rate one RTT ago, but also the derivative of the window size and the derivative of RTT at one RTT ago.

Therefore, the congestion control process can be viewed as a "combined" sparse reward process. Combined means it is not a complete sparse process, since the current action can still be rewarded, while the previous actions would also influence the current reward. Moreover, the window size $w(t)$ is an integer, and the influence of increasing or decreasing $w(t)$ will be delayed until $w(t)$ packets are received.

E. UTILITY FUNCTION

We next define a *utility function* for training and evaluation of the proposed algorithm. For resource allocation/traffic engineering, the α-fairness function is widely adopted [27]. In this model, the utility function is defined as

$$U_\alpha(\iota) = \begin{cases} \log(\iota), & \text{if } \alpha = 1 \\ \dfrac{\iota^{1-\alpha}}{1-\alpha}, & \text{otherwise,} \end{cases} \tag{8}$$

where parameter α determines different types of fairness. For example, if $\alpha \to \infty$, maximizing the sum utility becomes a max-min problem. If $\alpha = 1$, maximizing the sum utility ensures proportional fairness. In FAST TCP [34], Wei *et al.* adopt this utility function and prove the fairness of the proposed congestion control algorithm.

In this paper, the goal is to trade-off between throughput and latency. For high throughput, the sending rate should be set high to prevent the bottleneck queue from being nonempty, so that the bottleneck capacity can be fully utilized. On the other hand, for low latency, the sending rate should be set low to ensure low occupancy level at the bottleneck queue, so the queueing delay will be low. We adopt the following utility function to trade-off the two design goals [27]

$$U(x(t), \tau_{RTT}(t)) = U_\alpha(x(t)) - \beta U_\alpha(\tau_{RTT}(t)). \tag{9}$$

where coefficient β indicates the relative importance of throughput and latency.

The goal of the congestion control algorithm is to optimize the expectation of (9) *by properly setting the congestion window cWnd.*

V. TCP-DRINC DESIGN

A. APPROACH FOR MULTI-AGENT COMPETITION

As discussed, we are dealing with a multi-agent competition problem. At a bottleneck link, multiple flows compete for network resources such as buffer and capacity. In addition, the exploration strategy of other users could be sensed as dynamics of the environment, which could lead to misinformed state feedback and result in an unstable training process and poor performance. We tackle this problem by integrating three methods: (i) feature selection, (ii) clipped rewards, and (iii) a modified training process.

Feature selection is presented in Section V-B. By hand-picking the features that are indicative of system state, we can deal with abnormal system dynamics caused by other users' unknown strategies. Reward clipping can reduce the reward variation in different environments and increase the learning stability, which is discussed in detail in Section V-D.

For training, we combine the suggestions from [35]–[37] to have: (i) a relatively small experience replay buffer, and (ii) concurrent experience replay trajectories (CERTs) for sampling training data. The first method can mitigate the non-stationary nature of the local experience sensed by an agent. The normal experience replay buffer size is multiples of 10K. In this work, we set the buffer size to 1600, meaning the algorithm only keeps the most recent 1600 samples. For CERTs, each agent takes the same samples for training. To ensure this, the only synchronization needed is to assign the same random seed to each agent at the beginning of training. It requires no synchronization of the agents afterward, while guaranteeing independent operation of each agent.

B. FEATURE ENGINEERING

During a TCP session, the sender acquires the following information: (i) congestion window size, (ii) RTT, and (iii) inter-arrival time of ACKs. It is necessary to pre-process the sensed data. The benefits are two-fold: accelerating the training process and better explanation of the model. According to (6) and (7), the current sending rate is determined by the cWnd difference and the sending rate one RTT ago. Therefore, cWnd difference, Δw, as measured by the difference in two consecutive time slots, and RTT τ_{RTT} should be collected as features.

In the transport process, the minimum RTT within a time window provides an estimation of the propagation delay. The reason why not using the minimum RTT of the entire transmission process, is to take into account the possibility of route changes. Let $\hat{\tau}_p$ denote the minimum RTT. When cWnd is smaller than the product of capacity and $\hat{\tau}_p$, the measured RTT will always converge to $\hat{\tau}_p$. Thus the minimum RTT $\hat{\tau}_p$ should also be treated as state information. In TCP-Drinc, we take the ratio $\nu_{RTT} = \hat{\tau}_p/\tau_{RTT}$ as a feature, which is indicative of the relative portions of propagation delay in the RTT. Furthermore, the difference between RTT and the minimum RTT, i.e., $\delta_{RTT} = \tau_{RTT} - \hat{\tau}_p$, is an estimate of the overall queuing delay and is indicative of the network congestion level. To track the minimum RTT, the estimation will be updated at 10 RTT intervals. The last feature we pick is the inter-arrival time of ACKs, denoted by τ_{ACK}, which is indicative of the goodput of the network.

C. DEFINITIONS OF STATES AND ACTIONS

We next define the state and action space of TCP-Drinc. The non-Markovian nature of the problem makes it necessary to take history into consideration. As discussed, we consider the following features of a TCP connection: (i) cWnd difference Δw, (ii) RTT τ_{RTT}, (iii) the minimum RTT over RTT ratio ν_{RTT}, (iv) the difference between RTT and the minimum RTT δ_{RTT}, and (v) the inter-arrival time of ACKs τ_{ACK}. The algorithm runs in slotted time (e.g., 10 ms per time slot). In each time slot t, we measure the above features and record them as state of the time slot $\mathbf{s}_t$. If there is no events (e.g., no ACK received or cWnd changed) in a time slot t, we have $\mathbf{s}_t = \mathbf{s}_{t-1}$. We then take the combination of the states of M consecutive past time slots as one system state, in the form of a 2nd-order tensor, which is denoted as $\mathcal{S}$ and given by

$$\mathcal{S} = [\mathbf{s}_0, \mathbf{s}_1, ..., \mathbf{s}_{M-1}], \tag{10}$$

where $\mathbf{s}_t = [\Delta w(t), \tau_{RTT}(t), \nu_{RTT}(t), \delta_{RTT}(t), \tau_{ACK}(t)]$.

The action space consists of five actions on adjusting cWnd. In order to trade-off between agility and robustness through the process, we adopt actions $w = w \pm 1$ as exponentially increase/decrease, and actions $w = w \pm \frac{1}{w}$ as linearly increase/decrease of w. The exponentially increase/decrease actions allow the agent to quickly adapt to fast variations of the environment. The linearly increase/decrease actions enhance robustness when the agent decides to doodle around a proper operating point. Moreover, the *no change* action should also be included. The action space is thus defined as

$$\mathcal{A} = \{w = w \pm 1;\ w = w \pm \frac{1}{w};\ \text{no change}\}. \tag{11}$$

D. REWARD CALCULATION

Reward design is also challenging for DRL based congestion control. The RTT measurements at the sender side is usually noisy. Factors such as the bursty sending process, varying processing times at the devices along the path, and recovery from packet losses in the physical layer all contribute to measurement noise. However, it is important to acquire accurate RTT measurements since a misinformed RTT might result in wrong reward to the agent's action, and affect the convergence of the training process. In this paper, we adopt a low-pass filter with a fixed window size, i.e., exponential window moving average (EWMA), for RTT measurements, which removes frequent, small latency jitters.

Goodput is measured by counting the ACKs received within a time window. A benefit of this method is we directly calculate the goodput, while random packet losses are excluded. The measured goodput usually varies over time due to factors such as bursty sending process and random distribution in intermediate queues, as well as packet losses caused by link errors or congestion. We also take EWMA with one RTT window size to process the measured goodput.

Recall that the action of a sender will take effect after one RTT, and the impact on future goodput is exponentially discounted. The overall impact of an action on future goodput, denoted as $z(t)$, can be estimated as

$$z(t) = \sum_{\iota=t}^{\infty} \hat{z}(\iota)(1-\eta)\eta^{\iota-t}, \tag{12}$$

where $\eta < 1$ is the decay rate of the action's influence in the future, and $\hat{z}(t)$ is the measured goodput at time t. In order to make the calculation tractable in time and storage, we cut off time to have a horizon of L. The approximation of $z(t)$ for an L-horizon is

$$z(t) = \sum_{\iota=t}^{t+L-1} \hat{z}(\iota) \cdot \frac{(1-\eta)}{1-\eta^{L+1}} \cdot \eta^{\iota-t} \tag{13}$$

For fairness, we compute the utility function as

$$U(z(t), \tau_{RTT}(t)) = U_\alpha(z(t)) - \beta U_\alpha(\tau_{RTT}(t)). \tag{14}$$

In the context of Q-learning, Q-value denotes the expectation of reward. If we only adopt the utility (14) in the reward function, the agent may keeps on choosing the same action, which returns a positive utility value but does not converge to the optimal operating point. Therefore, we define *reward function* as the utility difference, given by

$$r(t) = U(t + \tau_{RTT}(\tilde{t})) - U(t), \tag{15}$$

where $\tilde{t}$ is one RTT after t.

Furthermore, for different network environments (e.g., different bottleneck link capacities and propagation delays), the reward calculated using (15) may have a large range of variation. In order to make the TCP-Drinc design more general and adaptable to various network environments, we *clip* the reward value to the range $[-1, 1]$. With clipping, although the training process could be relatively longer in some cases, the chance of convergence of the training process could be greatly improved (i.e., by avoiding large oscillations) and the same TCP-Drinc design can be applied to varying network environments, as will be shown in our simulation results.

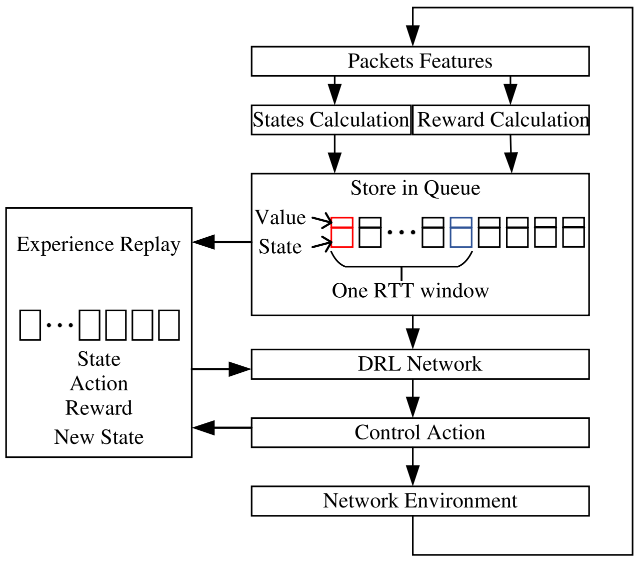


FIGURE 3. The proposed TCP-Drinc system architecture.

E. EXPERIENCE BUFFER FORMATION

The TCP-Drinc design is presented in Fig. 3, which takes the specifically designed structure of training samples to implicitly learn and predict the future. As shown in Fig. 1, it takes $\tau_1(t)$ for an action to take effect on the bottleneck queue(s). Thus the action that actually works should be $\mathbf{a}_s(t - \tau_1(t))$. Moreover, ACKs carry system state feedback with delay $\tau_2(t)$. Thus the currently sensed state $\mathbf{s}(t)$ is actually the system state $\mathbf{s}_s(t - \tau_2(t))$. To deal with such delays, we look into the future for $\tau_1(t) + \tau_2(t)$ to predict the system state based on the current state and action, and then choose the next action. However, such prediction is challenging since (i) the delays $\tau_1(t)$ and $\tau_2(t)$ are stochastic; (ii) the predictions are usually noisy and error-prone; and (iii) the computation could be intensive due to the high dimension of the system.

In this paper, we introduce a new strategy to address this problem. First of all, we use a buffer to store the running dynamics history in th form of three-tuples $\{\mathbf{s}(t), \mathbf{a}(t), r(t)\}$. Every time when a new tuple $\{\mathbf{s}(t), \mathbf{a}(t), r(t)\}$ is inserted, we look backward in the buffer to find the older tuple that was saved one RTT before. Then we combine these two tuples to obtain a complete training sample as a four-tuple $\{\mathbf{s}(t - \tau_{RTT}(t)), \mathbf{a}(t - \tau_{RTT}(t)), \mathbf{s}(t), r(t - \tau_{RTT}(t))\}$. This sample is then stored in the experience buffer to be used in the training process. This way, the DQN can better learn the reward based on current and future states.

F. TCP-DRINC AGENT DESIGN

With definitions of space and action, reward calculation, and experience buffer design, we are now ready to present the agent design, which is based on a DCNN as shown in Fig. 4. The agent takes the five features of 64 consecutive time slots as input and choose the next action from the action space.

In prior works such as [27], the EWMA is utilized to process the noisy state information. However, this method

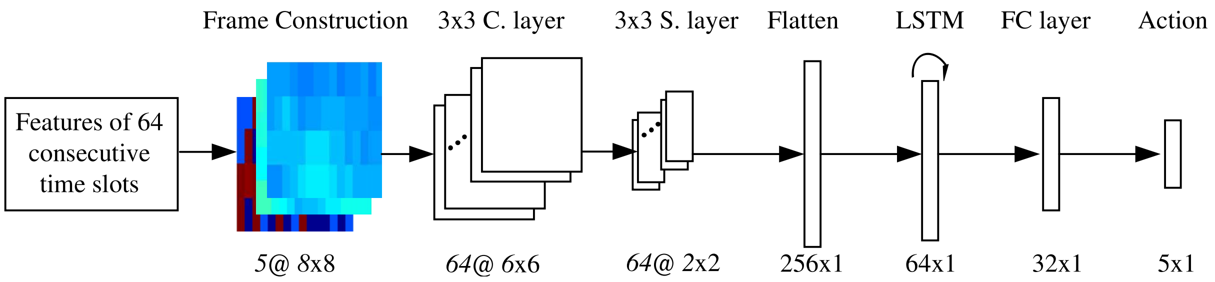


FIGURE 4. Design of the proposed DCNN (in the figure, "C." represents the convolutional layer, "S." represents the down sampling (pooling) layer, "FC" means fully connected).

suffers from the long tail effect when used as a filter, and it could miss the rich information in feedback. In this paper, we proposed a new approach to fully utilize the rich feedback information by adopting a DCNN. We use the combined state of M consecutive time slots including and before the current time t and divide them in to frames as inputs. For instance, we set $M = 64$ and obtain 5 frames. Each frame has a size 8×8 and consists of data on one of the five features, as shown in Fig. 4. With the convolutional layers and pooling layer, the more stable, higher level features can be abstracted from the raw state information.

Furthermore, the non-Markovian nature of the problem should also be accounted for, as well as the action-taking-effect delay τ_1 and feedback delay τ_2 (see Fig. 1). A promising approach is to incorporate an LSTM to handle the correlation in the time series [38]. Through back-propagation through time (BPTT), this structure exploits memory to extract system features. As in Fig. 4, the LSTM layer is placed after the DCNN extracts the stable features. Hausknecht and Stone [38] proposed two training methods: Bootstrapped Sequential Updates and Bootstrapped Random Updates. The former executes the training by randomly selecting an episode and then starting from the beginning of the episode. The latter, however, picks a random timestep in an episode and proceeds for a fixed amount of timesteps. In this paper, we adopt the Bootstrapped Random Updates method to train the DQN by combining with experience replay.

In Fig. 4, the fully connected layer is used to compute the Q-value for each action. We adopt the Exponential Linear Unit (ELU) *activation function*, defined as

$$ELU(\iota) = \begin{cases} \iota, & \iota > 0 \\ \lambda \cdot (e^{\iota} - 1), & \iota \leq 0, \end{cases} \tag{16}$$

instead of Rectified Linear Units (ReLU). Since we have to deal with negative rewards, it is beneficial not to kill the nodes with negative outputs. In the last layer, which is to output the Q-value for actions, no activation function is used.

VI. SIMULATION STUDY

A. SIMULATION SETUP

In this paper, the DCNN is implemented with TensorFlow [14]. The training is executed at a reasonable speed on a PC with an I5-8600k CPU and Nvidia GeForce 1060 3GB GPU. The dropout layer is applied with a 0.2 dropout probability to provide both regularization and ensemble effect. The LSTM layer with 64 units is applied after the DCNN layers. One fully connected layer is used with activation function *ELU*. The output layer is a linear combination of the previous output with five outputs, one for each action. The control action is applied every 10 ms. If there is no ACK received during a time slot, we simply copy the previous state. The agent measures RTT using timestamps in received ACKs. The minimum RTT is updated every 10 RTTs.

In the simulations, we assume there are N remote hosts that serve N wired or wireless users. Each remote host runs a DRL agent independently, i.e., the DRL agents do not share/exchange information. The simulation is based on the classical *dumbbell* topology with two routers in the middle, between the remote hosts and users (see Fig. 2).

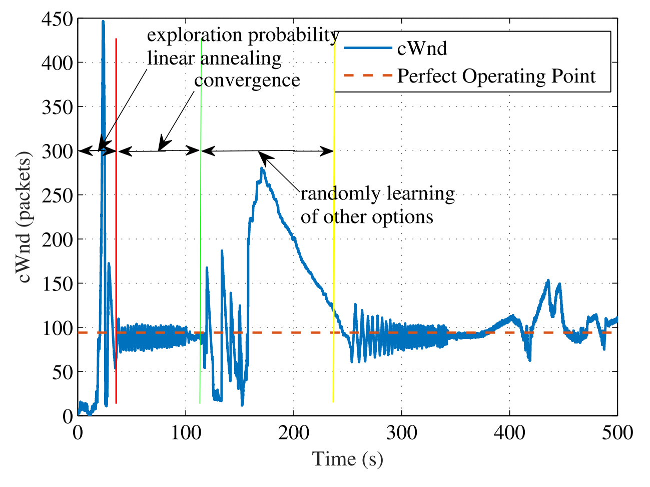
For fair and comprehensive performance comparison, we choose the following five benchmark schemes from existing representative algorithms, including: (i) TCP-NewReno [16], (ii) TCP-Cubic [17], (iii) TCP-Hybla [39], (iv) TCP-Vegas [40], and (v) TCP-Illinois [41]. We will mainly focus on their performance on throughput and RTT in this study.

B. TRAINING PROCESS

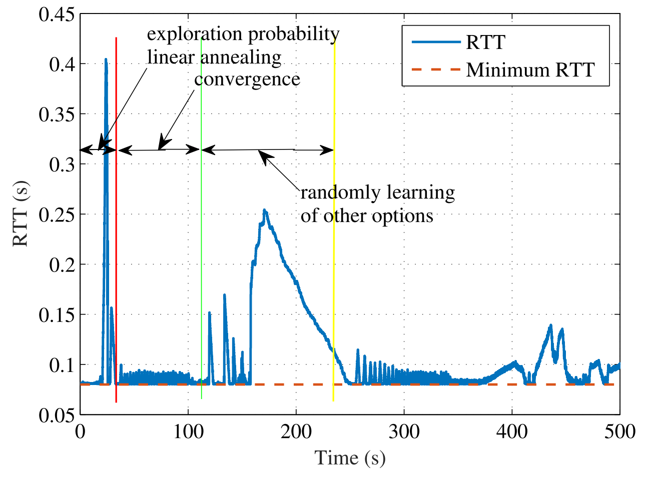
The training process consists of two phases: pre-training and distributed training. In pre-training, we train each agent for N_{pre} episodes with the same setup except that only one user is served to build the baseline behavior for the agent. Then in distributed training, the competition strategy of each agent will be trained independently for N_{dis} episodes.

1) PRE-TRAINING

In this simulation, we have $N_{pre} = 20$. The bottleneck capacity is 10 Mbps and the propagation delay is 80 ms. Each training episode lasts for 500 s. Fig. 5 shows the cWnd and RTT traces in the 10th training episode. It can be seen that the learning process consists of three phases: the exploration probability linear annealing phase, the convergence phase, and the phase of randomly learning of other options. In Fig. 5(a), before the vertical red line is the exploration probability annealing process, which involves a large probability of random choice of actions. With the exploration probability decreased to 0.1, which is the end of the first

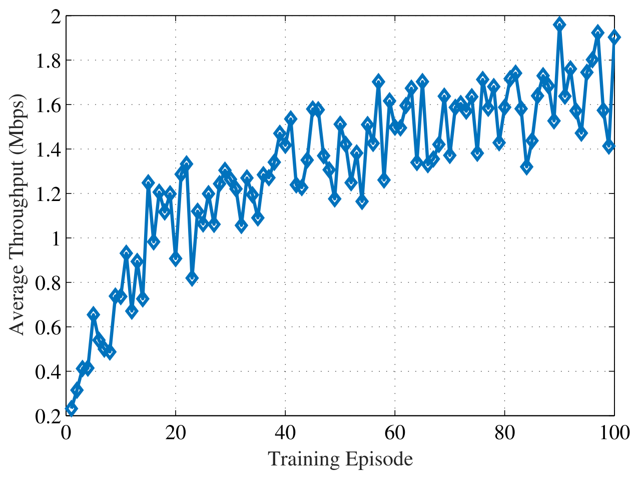


(a)

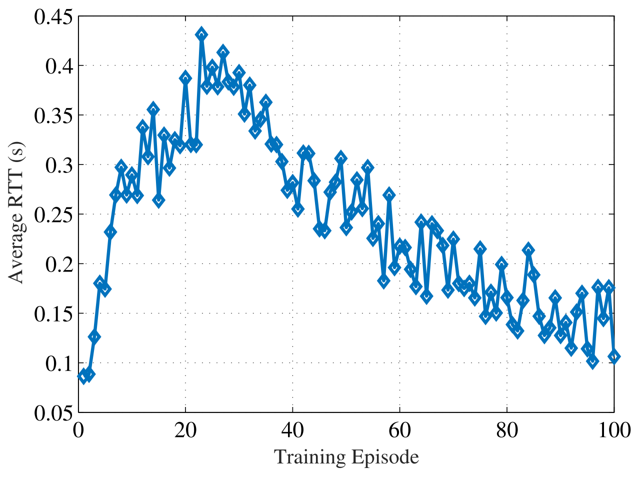


(b)

FIGURE 5. The cWnd and RTT traces of pre-training with a single agent obtained in the 10th training episode. (a) cWnd dynamics. (b) RTT dynamics.



(a)



(b)

FIGURE 6. The throughput and RTT dynamics of distribute training of five agents. (a) Average throughput. (b) Average RTT.

phase, the agent can adjust the action to converge to the *perfect operating condition*, i.e., the cWnd that can fully utilize the bottleneck link capacity without introducing extra queuing delay (the horizontal red dashed line in Fig. 5(a)). Then with a 0.1 probability, the agent will randomly deviate from the perfect operating point to explore the state space. This process will be repeated to train the agent with different network conditions.

2) DISTRIBUTED TRAINING

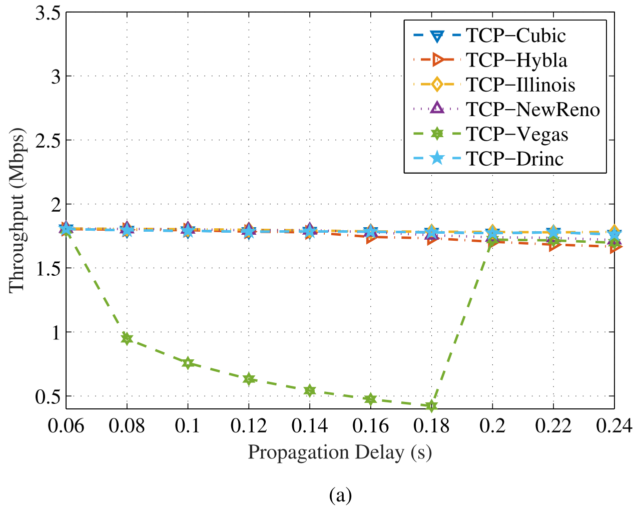
After pre-training, we copy the model to five different senders (i.e., there are $N = 5$ hosts serving 5 wired users), and let them run the training process independently. We set the random seed to 17 for all the senders. The basic environment setting is the same and the distributed training runs for $N_{dis} = 100$ episodes. In Fig. 6, the average throughput and RTT in the distributed training process are presented. We can see that as the distributed training proceeds, the average throughput is increasing in general. And the average RTT eventually converges toward the minimum RTT of 80 ms.

C. CONGESTION CONTROL PERFORMANCE

In this section, we present our experimental results with different network parameters to test the performance of TCP-Drinc, specifically, over situations that deviate from that TCP-Drinc was originally trained. As in Section VI-B, TCP-Drinc is trained with five TCP sessions with 10 Mbps bottleneck capacity and 80 ms propagation delay. The following three testing cases are simulated in this section.

1) Case I: four users; the propagation delay is varied in the range [60, 240] ms; and the bottleneck bandwidth is 10 Mbps;
2) Case II: the number of users is varied in the range [3, 9]; the propagation delay is 80 ms; and the bottleneck bandwidth is 8 Mbps;
3) Case III: four users; the propagation delay is 100 ms; the bottleneck bandwidth is varied within [5, 20] Mbps.

The bottleneck buffer size is set to 100 packets. Each packet consists of a 1000-Byte payload plus a 40-Byte header.



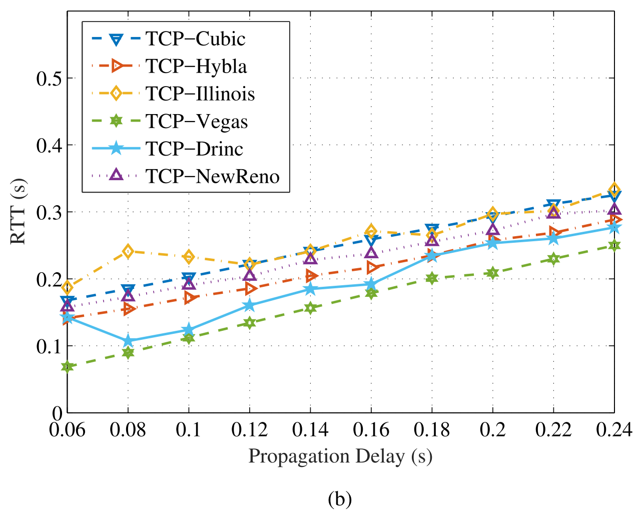
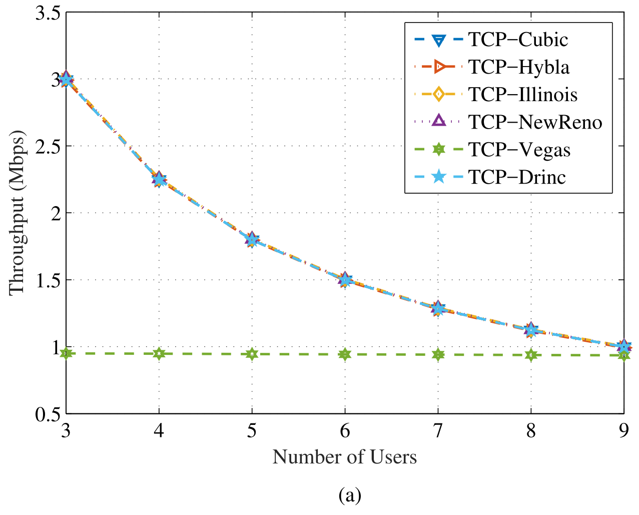


FIGURE 7. Throughput and RTT statistics for increased propagation delay (Case I). (a) Average throughput. (b) Average RTT.



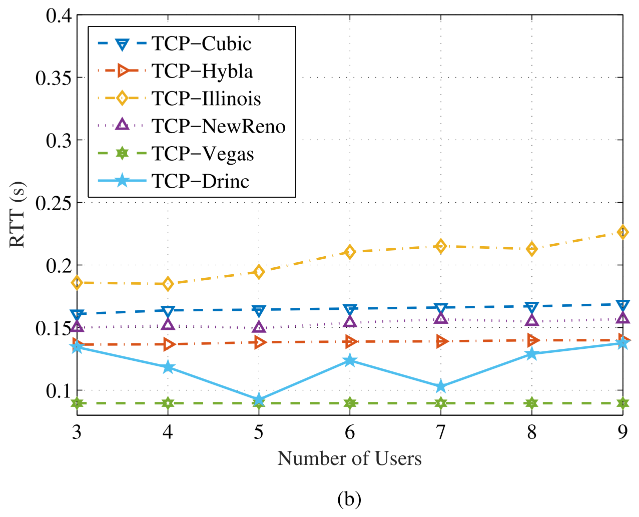


FIGURE 8. Throughput and RTT statistics for increased number of users (Case II). (a) Average throughput. (b) Average RTT.

As shown in Figs. 7, 8, and 9, the well-trained TCP-Drinc model achieves a promising performance under all the scenarios. In Fig. 7, we present the throughput and RTT for increased propagation delay. We can see that all the RTTs increase with propagation delay, while the throughput is relatively stable (except for TCP-Vegas). TCP-Drinc achieves the highest throughput and the second smallest RTT for the entire range of propagation delay. Although TCP-Vegas achieves a lower RTT than TCP-Drinc, its throughput is the lowest, indicating that the low RTT is achieved by keeping the sending rate low and the bottleneck buffer almost empty (i.e., sacrificing the utilization of the bottleneck link capacity). Unlike the several loss based protocols, TCP Vegas is a latency based scheme that linearly adjusts its cWnd according to the difference between the current RTT and the minimum RTT. It is effective in keeping RTT low, but is sensitive to network condition changes and suffers from relatively low throughput.

In Fig. 8, the throughput and RTT are presented for increased number of users. We can see the capacity for each user decreases with increased number of users. Again, TCP-Drinc achieves the highest throughput and the second lowest RTT. TCP-Vegas outperforms TCP-Drinc with a lower RTT, but at the cost of a low throughput. Fig. 9 presents the throughput and RTT achieved under different bottleneck capacity. For increased bottleneck capacity, the throughput increases and the RTT decreases as expected (except for TCP Vegas). Again, TCP-Drinc achieves the highest throughput and the second lowest RTT for the entire range of bottleneck capacity. TCP Vegas can always achieve the lowest RTT, but at the cost of low throughput. TCP-Drinc achieves a comparable throughput as loss based protocols (i.e., TCP-Cubic or TCP-NewReno), but only introduces a 20% higher RTT than TCP-Vegas. This is because the TCP-Drinc algorithm can always find the operation point that is close to the perfect operating point. We also find that the performance of TCP-Drinc is the best when the network parameters are close to the training scenarios.

We next examine the performance of the schemes under dynamic network settings. In particular, the simulation is

TABLE 2. Jain's Fairness Index Achieved by the Congestion Control Schemes.

	TCP-Cubic	TCP-Hybla	TCP-Illinois	TCP-NewReno	TCP-Vegas	TCP-Drinc
Average fairness index	0.7873	0.8025	0.8125	0.7562	0.8214	0.8058
95% Confidence Interval	[0.7005, 0.8741]	[0.7008, 0.9042]	[0.7114, 0.9136]	[0.6284, 0.8839]	[0.7115, 0.9313]	[0.7233, 0.8882]
Confidence Interval Span	0.1736	0.2034	0.2022	0.2555	0.2198	0.1649

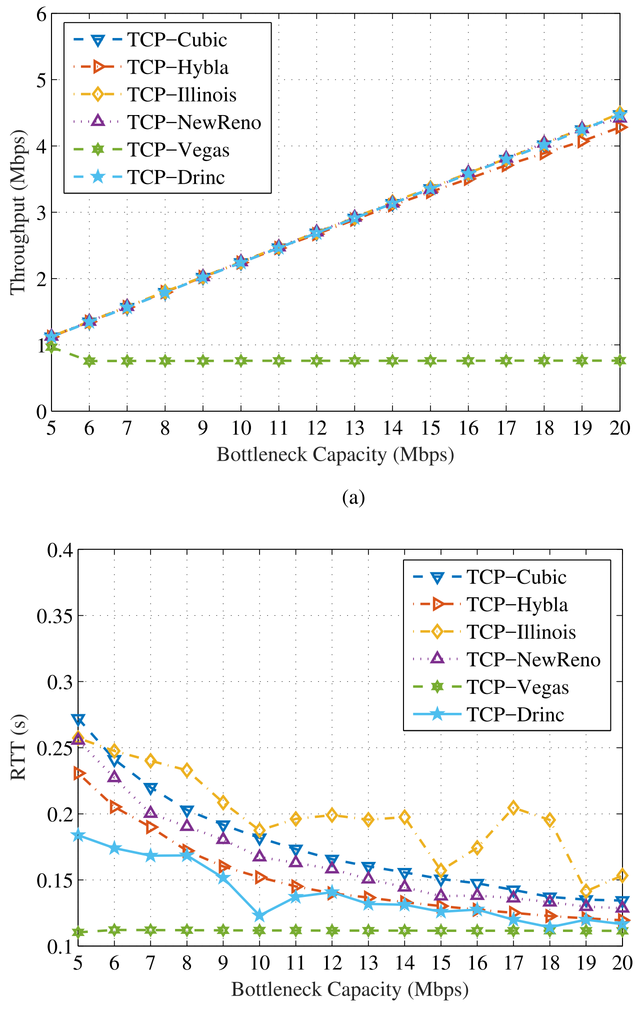


FIGURE 9. Throughput and RTT statistics for increased bottleneck capacity (Case III). (a) Average throughput. (b) Average RTT.

executed 100 times, each lasts for 500s. The number of users is 5. The bottleneck capacity is varied at a frequency of 10 Hz; each capacity is randomly drawn from a uniform distribution in [5, 15] Mbps. The propagation delay is also varied at a 10 Hz frequency and each value is randomly drawn from a uniform distribution in [0.06, 0.16]s. In Fig. 10, we plot the combined RTT (x-axis) and throughput (y-axis) results in the form of of 95% confidence intervals. That is, we are 95% sure that the throughput and RTT combination of each scheme are located within the corresponding oval area. We find that TCP-Drinc achieves a comparable throughput performance with the loss based protocols, e.g., TCP-Cubic and

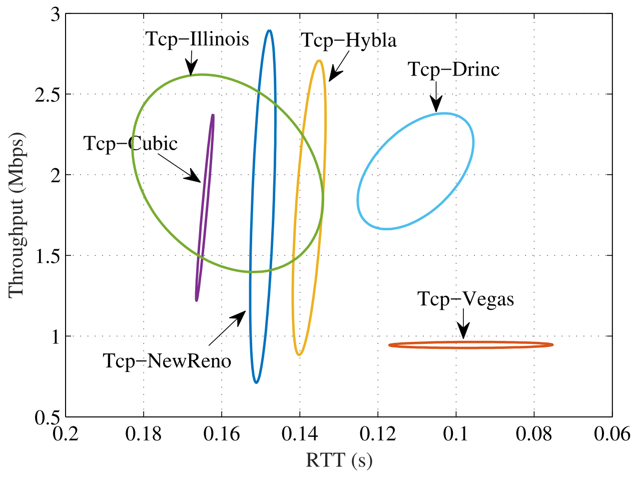


FIGURE 10. Throughput and RTT of the TCP variants under randomly varied network parameters. Each oval area represents the 95% confidence interval.

TCP-NewReno. Furthermore, TCP-Drinc achieves a much lower RTT performance than the loss based protocols, e.g., at least 46% lower than TCP-NewReno and 65% lower than TCP-Cubic. Furthermore, TCP-Drinc achieves an over 100% throughput gain than TCP-Vegas at the cost of a only 15% higher RTT.

To study the fairness performance of the algorithms, we evaluate the Jain's index they achieve in the simulation. The average fairness index and the corresponding 95% confidence intervals are presented in Table 2. TCP-Vegas and TCP-Illinois achieve the best fairness performance among all the algorithms. TCP-Drinc can still achieve a considerably high fairness index (only 1.9% lower than the best). Note that the best fairness performance of TCP Vegas is achieved at the cost of a much poorer throughput performance. It is also worth noting that the 95% confidence interval of TCP-Drinc is the smallest among all the schemes, which is indicative of its robustness under varying network conditions.

Finally, we evaluate the TCP variants with a mobile network scenario. The same network setting is used, except the last hop is now an LTE network with five mobile users. The users move in a disk area of 800 m radius, following a random walk mobility model with 1 m/s speed. The LTE network is simulated using the built-in LTE model in ns-3 [13]. The LTE BS uses a deep and separate queue for each user. Therefore, we do not consider TCP-NewReno in this simulation, since it will introduce a very large RTT under this setting. The wired part has the same configuration as in previous simulations. The simulation results are presented in Fig. 11. The proposed

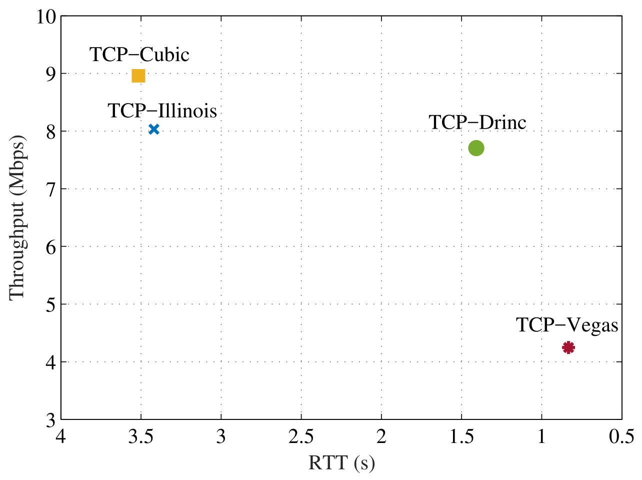


FIGURE 11. Throughput and delay results for the dumbbell topology with five mobile users served by an LTE network.

algorithm still performs well in the wireless network setting. Its throughput is comparable to the two loss based protocols, with a greatly reduced RTT. Its throughput is much higher than TCP Vegas while the RTT is only slightly higher.

VII. CONCLUSIONS

In this paper, we developed a framework for model-free, smart congestion control based on DRL. The proposed scheme does not require accurate models for network, scheduling, and network traffic flows; it also does not require training data, and is robust to varying network conditions. The detailed design of the proposed TCP-Drinc scheme was presented and the trade-offs were discussed. Extensive simulations with ns-3 were conducted to validate its superior performance over five benchmark algorithms.

REFERENCES

[1] V. Jacobson, "Congestion avoidance and control," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 18, no. 4, pp. 314–329, 1988.

[2] Y. Zhao, B. Zhang, C. Li, and C. Chen, "ON/OFF traffic shaping in the Internet: Motivation, challenges, and solutions," *IEEE Netw.*, vol. 31, no. 2, pp. 48–57, Mar./Apr. 2017.

[3] K. Winstein, A. Sivaraman, and H. Balakrishnan, "Stochastic forecasts achieve high throughput and low delay over cellular networks," in *Proc. USENIX NSDI*, Lombard, IL, USA, Apr. 2013, pp. 459–471.

[4] Y. Zaki, T. Pötsch, J. Chen, L. Subramanian, and C. Görg, "Adaptive congestion control for unpredictable cellular networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, pp. 509–522, Oct. 2015.

[5] Y. Sun, M. Peng, Y. Zhou, Y. Huang, and S. Mao. (Sep. 2018). "Application of machine learning in wireless networks: Key techniques and open issues." [Online]. Available: https://arxiv.org/abs/1809.08707

[6] M. Feng and S. Mao, "Dealing with limited backhaul capacity in millimeter wave systems: A deep reinforcement learning approach," *IEEE Commun.*, to be published.

[7] X. Chen, H. Zhang, C. Wu, S. Mao, Y. Ji, and M. Bennis, "Optimized computation offloading performance in virtual edge computing systems via deep reinforcement learning," *IEEE Internet Things J.*, to be published. [Online]. Available: https://ieeexplore.ieee.org/document/8493155

[8] Y. Sun, M. Peng, and S. Mao, "Deep reinforcement learning based mode selection and resource management for green fog radio access networks," *IEEE Internet Things J.*, to be published. [Online]. Available: https://ieeexplore.ieee.org/document/8468000

[9] Z. Chang, L. Lei, Z. Zhou, S. Mao, and T. Ristaniemi, "Learn to cache: Machine learning for network edge caching in the big data era," *IEEE Wireless Commun.*, vol. 25, no. 3, pp. 28–35, Jun. 2018.

[10] X. Wang, X. Wang, and S. Mao, "RF sensing in the Internet of Things: A general deep learning framework," *IEEE Commun.*, vol. 56, no. 9, pp. 62–69, Sep. 2018.

[11] X. Wang, L. Gao, S. Mao, and S. Pandey, "CSI-based fingerprinting for indoor localization: A deep learning approach," *IEEE Trans. Veh. Technol.*, vol. 66, no. 1, pp. 763–776, Jan. 2017.

[12] W. Wang, X. Wang, and S. Mao, "Deep convolutional neural networks for indoor localization with CSI images," *IEEE Trans. Netw. Sci. Eng.*, to be published. [Online]. Available: https://ieeexplore.ieee.org/document/8468057

[13] G. F. Riley and T. R. Henderson, "The *ns*-3 network simulator," in *Modeling and Tools for Network Simulation*, K. Wehrle, M. Günes, and J. Gross, Eds. Berlin, Germany: Springer, 2010, pp. 15–34.

[14] M. Abadi *et al.*, "Tensorflow: A system for large-scale machine learning," in *Proc. USENIX OSDI*, Savannah, GA, USA, Nov. 2016, pp. 265–283.

[15] D. R. Cox, "Long-range dependence: A review," in *Statistics: An Appraisal*, H. A. David and H. T. David, Eds. Ames, IA, USA: Iowa State Univ. Press, 1984, pp. 55–74.

[16] S. Floyd, A. Gurtov, and T. Henderson, *The NewReno Modification to TCP's Fast Recovery Algorithm*, document RFC 3782, IETF, Apr. 2004.

[17] S. Ha, I. Rhee, and L. Xu, "CUBIC: A new TCP-friendly high-speed TCP variant," *ACM SIGOPS Operat. Syst. Rev.*, vol. 42, no. 5, pp. 64–74, 2008.

[18] K. Tan, J. Song, Q. Zhang, and M. Sridharan, "A compound TCP approach for high-speed and long distance networks," in *Proc. IEEE INFOCOM*, Barcelona, Spain, Apr. 2006, pp. 1–12.

[19] M. Alizadeh *et al.*, "Data center TCP (DCTCP)," in *Proc. ACM SIGCOMM*, New Delhi, India, Aug./Sep. 2010, pp. 63–74.

[20] M. Dong, Q. Li, D. Zarchy, P. B. Godfrey, and M. Schapira, "PCC: Re-architecting congestion control for consistent high performance," in *Proc. USENIX NSDI*, Oakland, CA, USA, May 2015, pp. 395–408.

[21] W. Jiang, F. Ren, R. Shu, Y. Wu, and C. Lin, "Sliding mode congestion control for data center Ethernet networks," *IEEE Trans. Comput.*, vol. 64, no. 9, pp. 2675–2690, Sep. 2015.

[22] S. Floyd, "TCP and explicit congestion notification," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 24, no. 5, pp. 8–23, Oct. 1994.

[23] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Trans. Netw.*, vol. 1, no. 4, pp. 397–413, Aug. 1993.

[24] K. Nichols and V. Jacobson, "Controlling queue delay," *Commun. ACM*, vol. 55, no. 7, pp. 42–50, Jul. 2012.

[25] K. Xiao, S. Mao, and J. K. Tugnait, "Congestion control for infrastructure-based CRNs: A multiple model predictive control approach," in *Proc. IEEE GLOBECOM*, Washington, DC, USA, Dec. 2016, pp. 1–6.

[26] K. Xiao, S. Mao, and J. K. Tugnait, "MAQ: A multiple model predictive congestion control scheme for cognitive radio networks," *IEEE Trans. Wireless Commun.*, vol. 16, no. 4, pp. 2614–2626, Apr. 2017.

[27] K. Winstein and H. Balakrishnan, "TCP ex machina: Computer-generated congestion control," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 123–134, 2013.

[28] A. Sivaraman, K. Winstein, P. Thaker, and H. Balakrishnan, "An experimental study of the learnability of congestion control," in *Proc. ACM SIGCOMM*, Chicago, IL, USA, Aug. 2014, pp. 479–490.

[29] Z. Xu *et al.* (Jan. 2018). "Experience-driven networking: A deep reinforcement learning based approach." [Online]. Available: https://arxiv.org/abs/1801.05757

[30] W. Li, F. Zhou, K. R. Chowdhury, and W. M. Meleis, "QTCP: Adaptive congestion control with reinforcement learning," *IEEE Trans. Netw. Sci. Eng.*, to be published. [Online]. Available: https://ieeexplore.ieee.org/document/8357943

[31] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, Feb. 2015.

[32] K. Jacobsson, L. L. H. Andrew, A. Tang, K. H. Johansson, H. Hjalmarsson, and S. H. Low, "ACK-clocking dynamics: Modelling the interaction between windows and the network," in *Proc. IEEE INFOCOM*, Phoenix, AZ, USA, Apr. 2008, pp. 2146–2152.

[33] C. Jin, D. X. Wei, and S. H. Low, "FAST TCP: Motivation, architecture, algorithms, performance," in *Proc. IEEE INFOCOM*, Hong Kong, Mar. 2004, pp. 2490–2501.

[34] D. X. Wei, C. Jin, S. H. Low, and S. Hegde, "Fast TCP: Motivation, architecture, algorithms, performance," *IEEE/ACM Trans. Netw.*, vol. 14, no. 6, pp. 1246–1259, Dec. 2006.

[35] J. N. Foerster, Y. M. Assael, N. de Freitas, and S. Whiteson. (2016). "Learning to communicate to solve riddles with deep distributed recurrent Q-networks." [Online]. Available: https://arxiv.org/abs/1602.02672
[36] J. Foerster *et al.* (2017). "Stabilising experience replay for deep multi-agent reinforcement learning." [Online]. Available: https://arxiv.org/abs/1702.08887
[37] S. Omidshafiei, J. Pazis, C. Amato, J. P. How, and J. Vian. (2017). "Deep decentralized multi-task multi-agent reinforcement learning under partial observability." [Online]. Available: https://arxiv.org/abs/1703.06182
[38] M. Hausknecht and P. Stone. "Deep recurrent Q-learning for partially observable MDPs." [Online]. Available: https://arxiv.org/abs/1507.06527
[39] C. Caini and R. Firrincieli, "TCP hybla: A TCP enhancement for heterogeneous networks," *Int. J. Satellite Commun. Netw.*, vol. 22, no. 5, pp. 547–566, Sep./Oct. 2004.
[40] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson, "TCP Vegas: New techniques for congestion detection and avoidance," in *Proc. SIGCOMM*, London, U.K., Aug./Sep. 1994, pp. 24–35.
[41] S. Liu, T. Ba ar, and R. Srikant, "TCP-Illinois: A loss-and delay-based congestion control algorithm for high-speed networks," *Perform. Eval.*, vol. 65, no. 6, pp. 417–440, 2008.

KEFAN XIAO (S'14) received the B.E. degree in electronic engineering from Xi'an Jiaotong University, Xi'an, China, in 2011, the M.S. degree in electronic engineering from Shanghai Jiao Tong University, Shanghai, China, in 2014, and the Ph.D. degree in electrical and computer engineering from Auburn University, Auburn, AL, USA, in 2018. He is currently a Software Engineer with Google, Inc. His research interests include TCP congestion control, video streaming, and transmissions.

SHIWEN MAO (S'99–M'04–SM'09–F'19) received the Ph.D. degree in electrical and computer engineering from Polytechnic University (now New York University), Brooklyn, NY, USA. He is currently the Samuel Ginn Distinguished Professor with the Department of Electrical and Computer Engineering, and the Director of the Wireless Engineering Research and Education Center, Auburn University, Auburn, AL, USA.

His research interests include wireless networks, multimedia communications, and smart grid. He is a Distinguished Speaker of the IEEE Vehicular Technology Society. He is on the Editorial Board of the IEEE TRANSACTIONS ON NETWORK SCIENCE AND ENGINEERING, the IEEE TRANSACTIONS ON MOBILE COMPUTING, the IEEE TRANSACTIONS ON MULTIMEDIA, the IEEE INTERNET OF THINGS JOURNAL, the IEEE WIRELESS NETWORKING LETTERS, the IEEE MULTIMEDIA, and *ACM GetMobile*.

He is a Fellow of the IEEE. He received the NSF CAREER Award, in 2010, the 2013 IEEE ComSoc MMTC Outstanding Leadership Award, the 2015 IEEE ComSoc TC-CSR Distinguished Service Award, the 2017 IEEE ComSoc ITC Outstanding Service Award, and the Auburn University Creative Research & Scholarship Award, in 2018. He was a co-recipient of the 2004 IEEE Communications Society Leonard G. Abraham Prize in the Field of Communications Systems, IEEE ICC 2013, IEEE WCNC 2015, the Best Paper Awards from IEEE GLOBECOM 2015and 2016, the Best Demo Award from IEEE SECON 2017, and the IEEE ComSoc MMTC Best Conference Paper Award in 2018.

JITENDRA K. TUGNAIT (M'79–SM'93–F'94–LF'16) received the B.Sc. degree (Hons.) in electronics and electrical communication engineering from the Punjab Engineering College, Chandigarh, India, in 1971, the M.S. and E.E. degrees from Syracuse University, Syracuse, NY, USA, and the Ph.D. degree from the University of Illinois at Urbana–Champaign, in 1973, 1974, and 1978, respectively, all in electrical engineering.

From 1978 to 1982, he was an Assistant Professor of electrical and computer engineering with the University of Iowa, Iowa, IA, USA. He was with the Long Range Research Division, Exxon Production Research Company, Houston, TX, USA, from 1982 to 1989. He joined the Department of Electrical and Computer Engineering, Auburn University, Auburn, AL, USA, in 1989, as a Professor, where he currently holds the title of James B. Davis Professor. His current research interests include statistical signal processing, wireless communications, and multiple target tracking.

Dr. Tugnait is a past Associate Editor of the IEEE TRANSACTIONS ON AUTOMATIC CONTROL, the IEEE TRANSACTIONS ON SIGNAL PROCESSING, the IEEE SIGNAL PROCESSING LETTERS, and the IEEE TRANSACTIONS ON WIRELESS COMMUNICATIONS, and a past Senior Area Editor of the IEEE TRANSACTIONS ON SIGNAL, and a past Senior Editor of IEEE WIRELESS COMMUNICATIONS LETTERS.