Improving TCP Congestion Control with Machine Intelligence

Yiming Kong
School of ECE, Georgia Tech
Atlanta, GA
ymkong@gatech.edu

Hui Zang
Futurewei Technologies, Inc.
Santa Clara, CA
hui.zang@huawei.com

Xiaoli Ma
School of ECE, Georgia Tech
Atlanta, GA
xiaoli@gatech.edu

ABSTRACT

In a TCP/IP network, a key to ensure efficient and fair sharing of network resources among its users is the TCP congestion control (CC) scheme. Previously, the design of TCP CC schemes is based on hard-wiring of predefined actions to specific feedback signals from the network. However, as networks become more complex and dynamic, it becomes harder to design the optimal feedback-action mapping. Recently, learning-based TCP CC schemes have attracted much attention due to their strong capabilities to learn the actions from interacting with the network. In this paper, we design two learning-based TCP CC schemes for wired networks with under-buffered bottleneck links, a loss predictor (LP) based TCP CC (LP-TCP), and a reinforcement learning (RL) based TCP CC (RL-TCP). We implement both LP-TCP and RL-TCP in NS2. Compared to the existing NewReno and Q-learning based TCP, LP-TCP and RL-TCP both achieve a better tradeoff between throughput and delay, under various simulated network scenarios.

CCS CONCEPTS

• **Networks → Transport protocols**; • **Computing methodologies → Supervised learning**; **Reinforcement learning**; *Classification and regression trees*;

KEYWORDS

TCP congestion control, packet loss prediction, reinforcement learning, machine learning

ACM Reference Format:
Yiming Kong, Hui Zang, and Xiaoli Ma. 2018. Improving TCP Congestion Control with Machine Intelligence. In *NetAI'18: ACM SIGCOMM 2018 Workshop on Network Meets AI & ML , August 24, 2018, Budapest, Hungary.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3229543.3229550

1 INTRODUCTION

Designing TCP congestion control (CC) schemes to ensure efficient and fair use of the network resources has been a well-motivated and intensely studied topic for nearly three decades, resulting in a range of influential algorithms that are either entirely host-to-host [3–6, 9, 16, 28, 31, 32], or with in-net support [15, 27]. We focus on host-to-host CC schemes due to their flexibility and independence from the network.

Many of the existing host-to-host CC schemes target networks of high-bandwidth and low congestive packet loss rate (e.g., [6, 28]). To support high bandwidth, a rule of thumb is to have the buffer size at each link linearly scale with the link-rate, which causes negative side-effects such as "bufferbloat" (i.e., high latency as a result of excessive buffering of packets) and high hardware cost. Thus reducing buffer size is desirable. It is also shown to have negligible change in throughput when a large number of TCP connections coexist in a single backbone link [1]. However, when the number of coexisting TCP connections is small, an under-buffered (i.e., buffer size smaller than that suggested by the rule of thumb) bottleneck link can often be under-utilized by existing TCP flows, which reduce their congestion windows (cwnd) frequently upon packet losses.

Therefore, the first question we explore in this paper is: *Can a TCP CC scheme learn to predict congestive packet losses?* Heuristics based on the measured throughput or round-trip time (RTT) of a TCP flow [3, 11, 29] perform poorly in loss prediction [2]. A carefully-built loss predictor model [23] shows higher prediction accuracy, but requires sophisticated human design. Recently, capability of machines to learn and represent complex models is re-discovered and exploited to solve various problems in computer networks [17–20, 25]. Thus, we develop a loss predictor (LP) using supervised learning, and incorporate it into the TCP CC to predict and reduce congestive packet losses. With tuning of a decision threshold *th*, the loss predictor based TCP (LP-TCP) achieves a desired tradeoff between throughput and delay. Compared to NewReno [5], a single "always-on" LP-TCP connection shows 29% increase in throughput with similar RTT, in an extremely under-buffered bottleneck link (See Table 5, $L = 5$). Also, when four LP-TCP connections coexist in an under-buffered bottleneck link, their average throughput increases by $4-5\%$ with slightly increased RTT (See Tables 6 and 7).

However, LP-TCP works better when the network model remains more or less fixed. When the topology and parameters of a network change, a new LP needs to be learned. Thus, we explore the next question: *Can a TCP CC scheme adaptively learn to act in a dynamic network environment, given an objective?* We then develop a reinforcement learning (RL) based TCP CC (RL-TCP), with an objective to improve a function of throughput, delay, and packet loss rate. RL-TCP exhibits an excellent tradeoff between throughput and delay. Compared to NewReno and a Q-learning based TCP (Q-TCP) [16], a single "always-on" RL-TCP achieves $7-8\%$ decrease in RTT and at least 9% increase in throughput, in an under-buffered bottleneck link (See Table 5, $L = 50$). When four RL-TCP connections coexist in an under-buffered bottleneck link, their throughput increases by $4-5\%$ while maintaining similar RTT (See Tables 6 and 7).

NetAI'18, August 24, 2018, Budapest, Hungary
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-5911-5/18/08…$15.00
https://doi.org/10.1145/3229543.3229550

The rest of the paper is organized as follows. Section 2 presents the related work. Section 3 presents the architecture for the proposed learning-based TCP CC schemes, and introduces LP-TCP and RL-TCP. Section 4 evaluates the performance of LP-TCP and RL-TCP and compares them to NewReno and Q-TCP on NS2. Section 5 concludes the paper.

2 RELATED WORK

Since the internet congestion collapse in 1986, congestion control for multiuser packet-switched networks has remained an active research field. Jacobson, in his seminal work TCP Tahoe [9] and Reno [10], introduced three core phases in a CC algorithm (i.e., slow start, congestion avoidance, and fast recovery), which become the foundation most TCP CC schemes build upon. Many TCP CC schemes look for better ways to adjust cwnd at congestion avoidance. For instance, Vegas [3] treats increasing RTT as a congestion signal and adjust cwnd to keep RTT in a desired range. Cubic [6] modulates its cwnd according to a cubic function. Compound [28] reacts to delay signals and packet loss events, and adopts a scalable increasing rule on cwnd in response to changes in the RTTs. While having unique characteristics, the above mentioned TCP CC schemes share a similarity of hard-wiring of predefined operations on the cwnd in response to specific feedback signals. They do not learn and adapt from experience.

Machine learning has been used to indirectly improve the performance of TCP CC schemes. For example, it has been used to classify congestive and contention loss [13], and to give a better estimation of the RTT [22]. It has also been applied to accurately forecast TCP throughput [21]. Recently, many machine learning based TCP CC schemes have been proposed. Remy [31] formalizes the multiuser CC problem as the POMDP and learns the optimum policy offline. It needs intense offline computation and the performance of RemyCCs depends on the accuracy of the network and traffic models. PCC [4] adaptively adjusts its sending rate based on continuously carried out "micro-experiments", but it is rate-based and its performance depends on the accuracy of the clocking. The learnability of TCP CC is examined in [24], where RemyCCs are used to understand what imperfect knowledge about the target network would hurt the performance of TCP CC the most.

In [16], a Q-learning based TCP (Q-TCP) was proposed that uses RL to design a TCP CC scheme. However, Q-TCP is designed mostly with a single TCP connection in mind. As we consider under-buffered networks with a small number of TCP connections, it is helpful to adopt more expressive features and redesign the action space. We also propose a different credit assignment component which we believe better captures TCP dynamics.

3 THE PROPOSED LEARNING BASED TCP CC SCHEMES

In this section, we explore ways to improve the performance of TCP CC schemes in wired networks with an under-buffered bottleneck link using machine intelligence. Specifically, we propose two learning-based TCP CC schemes, one based on supervised learning, and the other based on RL. The two learning-based TCP CC agents share a common architecture, shown in Fig. 1a. It contains three components:

- A sensing engine, which processes signals from the network, combines them with variables in the TCP sender, and outputs an array representing the current state. It may also compute other quantities when required;
- A learner, which consists of an online learning engine or a learned model. It takes in the current state, and outputs certain "prediction";
- An actuator, which acts (i.e., adjusts cwnd) based on the "prediction" from the learner.

The sensing engine computes statistics that reflect how congestive the network may be. Such statistics may include the packet inter-sending time, acknowledgment (ACK) inter-arrival time, and RTTs [31]. The learner serves as the "brain" of the TCP CC agent, learning the complex relationship between a certain state and possible actions, and informs the actuator to act accordingly. Proper design and training of the learner remain the key to a well-performing learning-based TCP CC scheme. Though unnecessary, our learning-based TCP CC schemes are based on NewReno. This means that slow start, fast retransmit, and fast recovery of NewReno are still adopted.

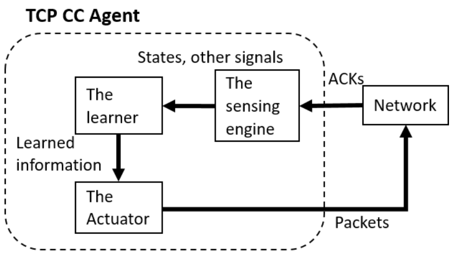
3.1 LP-TCP

Based on the architecture in Fig. 1a, we introduce our first learning-based TCP CC named LP-TCP. The intuition is simple. Since NewReno reduces sending rate (by halving cwnd) each time a packet loss occurs, and under-utilizes the bottleneck bandwidth in an under-buffered network, LP-TCP predicts and reduces packet loss events, lowers the frequency of sending rate reduction, and strives for a better throughput. Therefore, the learner in Fig. 1a is a packet loss predictor (LP), which tells the actuator how likely a packet will be lost if sent. If the probability of loss is higher than a threshold, the actuator does not send the packet (i.e., reduces cwnd by one). Otherwise, the actuator sends the packet. The inputs of the sensing engine, the learner, and the actuator are as follows:

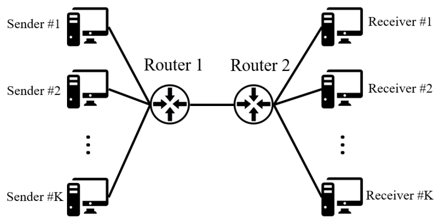
- *input to the sensing engine:* Received ACKs;
- *input to the learner:* cwnd size, the order of current packet in the cwnd, exponentially weighted moving average (EWMA), time series (TS), and minimum of ACK inter-arrival time, EWMA, TS, and minimum of packet inter-sending time, TS and minimum of RTT, TS of ratios of ACK inter-arrival time, TS of ratios of packet inter-sending time, TS of ratios of RTTs (TS of a variable includes 8 recent samples of that variable);
- *input to the actuator:* Estimated probability of loss of the current packet.

To reflect how congestive the network is, the sensing engine outputs a length-55 feature array as the state, based on received ACKs and variables in the TCP sender. Now we illustrate the process of building the loss predictor (the learner) using a supervised learning technique, called random forests [7, 8].

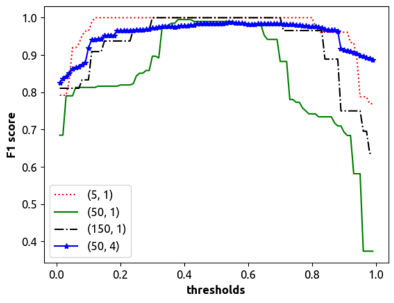
3.1.1 Training the LP. The learner is a loss predictor in LP-TCP. It takes the state as input, and predicts the probability of a packet being lost due to congestion should the packet be sent. We collect training data to train the learner through NewReno simulations on NS2. Whenever a packet gets sent, we record the state right before the packet goes into transmission as a feature vector. We



(a)



(b)



(c)

Figure 1: (a) Architecture of the proposed machine learning based TCP CC schemes. (b) A wired network with a dumbbell topology. (c) F1 score of the LP as a function of the decision threshold th.

want to mention that for LP-TCP, the packet inter-sending time (and thus the features that depend on it) is computed based on the packets the TCP sender is sending, instead of from the time-stamps in the received ACKs. If the packet is successfully delivered, this feature vector gets a corresponding label of 0; otherwise, the label is 1 (for loss). Since loss events are the minority events, we stop the collection when we have enough losses in the data. In this paper, the training data collection lasts 5000 seconds. A random forest model is then trained with this training set. For any feature vector representing a certain state, the model then outputs the estimated probability of loss should a packet get sent, which is the mean prediction of the decision trees in the forest.

3.1.2 Inferencing packet loss. After training, the sensing engine, the random forests LP, and the actuator work together as LP-TCP. During congestion avoidance, when a new ACK is received by the TCP sender, the $cwnd$ expands by $1/cwnd$, and the sensing engine updates the state. When the sender is about to send a packet, the state is computed again. The LP then takes in the state vector, and outputs a probability of loss of that packet. If the probability of loss is lower than a pre-determined threshold th, the actuator sends the packet. Otherwise, the actuator does not send the packet (i.e., reduces cwnd by one). For simulations in this paper, different ths are selected in different network setups. We will discuss the threshold selection process in detail in Section 4.1.

3.2 RL-TCP

A problem with the supervised learning based LP-TCP is that, when the topology and parameters of a network change, a new LP needs to be re-learned. Ideally, we would like the TCP CC scheme to continuously learn and adapt in a dynamic network environment, given an objective. This inspires us to formulate the TCP CC problem as an RL problem [26], where an agent with no prior knowledge learns to act by acting and receiving a reward (positive or negative) from the environment, with the goal of maximizing some kind of cumulative rewards. Doing so leads to our RL-based TCP CC (RL-TCP). Compared to Q-TCP [16], RL-TCP tailors the design of states and action space towards networks with under-buffered bottleneck links. More importantly, RL-TCP treats the temporal credit assignment of reward according to TCP dynamics. The inputs of the three components in Fig. 1a are as follows:

- *input to the sensing engine:* Received ACKs;
- *input to the learner:* the state (i.e., EWMA of the ACK inter-arrival time, EWMA of the packet inter-sending time, the ratio of current RTT and the minimum RTT, the slow start threshold, and cwnd size), and reward r from the network;
- *input to the actuator:* a value function of current state and actions, indicating how "good" each action is, at the current state.

The sensing engine updates a length-5 state vector representing the congestive condition whenever a new ACK is received. We want to mention that in RL-TCP, the packet inter-sending time (and thus the features that depend on it) is computed from the time-stamps in the received ACKs. At the beginning of each time step (generally one RTT), the sensing engine sends the current state vector to the learner. Each feature in the state vector is uniformly discretized into l levels over a range where most of its values lie in. In this paper, l is between 10 and 20. The sensing engine also computes a reward r from the network, based on the changes of utility U at two consecutive time steps. The utility U is a function of throughput tp, delay $d = RTT - RTT_{\min}$, and packet loss rate p, i.e.,

$$U = \log\left(\frac{tp}{B}\right) - \delta_1 \log(d) + \delta_2 \log(1-p), \tag{1}$$

where B is the bottleneck bandwidth, and δ_1 and δ_2 are adjustable coefficients.

At the beginning of each time step, the actuator selects an action. The actuator has an action space $\mathcal{A}$ containing four actions, i.e., $cwnd = cwnd + x$, where $x = -1, 0, +1, +3$. We choose small x to reduce the chances of incurring packet drops at an under-buffered bottleneck. Note that an action is actually "spread" into one RTT (i.e., $cwnd = cwnd + x/cwnd$, for each new ACK received during one RTT). Using value-based RL algorithms, the learner learns how "good" each action a is at a state s, called the Q-function $Q(s, a)$ in RL literature [26], defined as the cumulative rewards that the agent gets for performing action a at state s and acting optimally after that.

3.2.1 Learning $Q(s, a)$. To learn the Q-function $Q(s, a)$, we use SARSA [26], a popular on-policy temporal difference (TD) learning algorithm for value-based RL. At the beginning of time step $n + 1$, a SARSA agent updates the Q-function using

$$Q(s_n, a_n) \xleftarrow{\alpha_t} r_{n+1} + \gamma Q(s_{n+1}, a_{n+1}), \quad (2)$$

where $y_1 \xleftarrow{\alpha} y_2$ means $y_1 = (1-\alpha)y_1 + \alpha y_2$, s_i, a_i, r_i are state, action, and reward computed at the beginning of time step i, α_t is the learning rate as a function of time t (seconds), and γ is the discount factor. From [16], we understand that r_{n+1} is computed based on the difference $\Delta_{n+1} = U_{n+1} - U_n$, where U_i is the utility during time step $i-1$. However, in TCP CC, the effect of adjusting cwnd at time t on the utility is observed by the sender at $t + RTT$. Thus the effect of action a_n (that "spreads" into time step n) will be observed by the sender during time step $n+1$, and reflected in U_{n+2}. For this reason, we think r_{n+2} better captures the reward from the environment due to action a_n. Therefore, at the beginning of time step $n+1$, we modify the update rule in Eq. (2) to

$$Q(s_{n-1}, a_{n-1}) \xleftarrow{\alpha_t} r_{n+1} + \gamma Q(s_n, a_n), \quad (3)$$

where

$$r_{n+1} = \begin{cases} 10, & \Delta_{n+1} \geq 1 \\ 2, & 0 \leq \Delta_{n+1} < 1 \\ -2, & -1 \leq \Delta_{n+1} < 0 \\ -10, & \Delta_{n+1} < -1. \end{cases} \quad (4)$$

We adopt SARSA instead of the off-policy Q-learning [30], since SARSA was shown to achieve a better online performance than Q-learning (See [26], the "Cliff Walking" example).

3.2.2 Exploration and Exploitation. If the true Q-function is known, at the beginning of time step $n+1$, the actuator can act optimally by choosing the greedy action $a_{n+1} = \arg\max_{a \in \mathcal{A}} Q(s_{n+1}, a)$. But the learner can only estimate the Q-function. Also, in a dynamic environment, the optimal policy may change. Thus, the actuator uses the ϵ-greedy exploration-exploitation (E2) scheme, where it chooses a greedy action with probability $1-\epsilon$, and a random action with probability ϵ. ϵ is set to 0.1 by default. Constant exploration is necessary to maximize utility in a dynamic network environment [14]. Table 1 describes the outline of RL-TCP.

Table 1: RL-TCP (during congestion avoidance)

```
(S1)   $a_0 = a_1 = a_2 = 0, s_0 = s_1 = s_2 = \mathbf{0}, u_1 = u_2 = 0, t_0 = 0$
(S2)   For every new ACK received
(S3)       compute current state $s_2$, current utility $u_2$ (from $t_0$ up to now)
(S4)       If (slow start): use NewReno
(S5)       Else
(S6)           If (now $- t_0 \geq$ RTT)
(S7)               compute $r_2$ from $u_2, u_1$ based on Eq. (4)
(S8)               $Q(s_0, a_0) \xleftarrow{\alpha_{\text{now}}} r_2 + \gamma Q(s_1, a_1)$
(S9)               $a_2 = \epsilon$-greedy$(Q, s_2)$, $cwnd = cwnd + a_2/cwnd$
(S10)              $a_0 = a_1, a_1 = a_2, s_0 = s_1, s_1 = s_2, u_1 = u_2, t_0 =$ now
(S11)          Else
(S12)              $cwnd = cwnd + a_1/cwnd$
```

4 EXPERIMENTAL RESULTS

In this section, we evaluate the performance of our proposed learning-based TCP CC schemes in NS2. We adopt a dumbbell topology shown in Fig. 1b with K senders and K receivers. The bottleneck link is between router 1 and router 2 and has bandwidth B. The buffer size at router 1 is L (number of packets). A network setup with K senders/receivers and buffer size L is denoted by (L, K). The minimum RTT between each sender and receiver is $RTT_{\min}$. By default, B = 10Mb/s and $RTT_{\min}$ = 150ms. Thus, the bandwidth delay product (BDP) is 150 packets. The default traffic type for each sender is "always-on". We evaluate performance of TCP CC schemes in two main scenarios:

- Single sender with varying buffer size $L = 5, 50, 150$;
- Four senders sharing the bottleneck with buffer size $L = 50$.

The metrics we use include throughput tp of a flow (i.e., dividing the total amount of bytes received by the time during which the sender was active), average delay d of a flow (i.e., $d = RTT - RTT_{\min}$), and packet loss rate p of a flow (i.e., $p = \frac{\text{total packet loss}}{\text{total packet sent}}$). Unless stated otherwise, results are averaged over 100 iterations, each of which lasts 400 seconds. The unit for throughput is Mb/s, and for delay is millisecond (ms). $E(\cdot)$ represents expectation, and $V(\cdot)$ variance. Table 2 lists the TCP CC schemes[1] that we compare RL-TCP and LP-TCP with. For RL-TCP, $\alpha_t = 0.3 * 0.995^{\lfloor t/10 \rfloor}$, and $\gamma = 0.5$.

Table 2: TCP CC schemes for comparisons.

Name	Description
NewReno	See [5]
Q-TCP	Parameters from [16] are adopted. Action space = $\{-1, 0, 5, 10, 20\}$
Q_a-TCP	Same as Q-TCP, except for action space = $\{-1, 0, 1, 3\}$
Q-TCP$_{\text{ca}}$	Q-TCP using our proposed credit assignment
Q_a-TCP$_{\text{ca}}$	Q_a-TCP using our proposed credit assignment
RL-TCP$_{\text{no-ca}}$	RL-TCP using credit assignment from Eq. (2)

4.1 Setting Threshold th for LP-TCP

Decision threshold th serves as an important parameter of LP-TCP. If th is high, LP-TCP holds on to a packet only when it is highly confident that a packet loss will happen. If th is low, LP-TCP is more likely to hold on to packets to avoid potential packet losses. For each network setup (L, K), we first compute a threshold th_0 that maximizes the $F1$ score of the LP in Table 3. $(L, K)_{\text{mix}}$ means one LP-TCP and $K-1$ NewReno flows coexist in a bottleneck. Fig. 1c shows the $F1$ score of LP as a function of the decision threshold th, for various network setups.

But on top of loss prediction accuracy, network users often care more about the performance of throughput and delay. Therefore, we introduce a throughput-delay tradeoff metric M_e, defined as

$$M_e = \log(E(tp)) - 0.1 \log(E(d)). \quad (5)$$

Directly computing th that maximizes M_e is hard (if not impossible). Thus, we make a first attempt by sampling th as 0.1, 0.3, 0.5, 0.7, 0.9. Among these samples and th_0, we choose one th that maximize M_e. For various network configurations (L, K), we compute M_e for our th candidates in Table 3. The selected ths (corresponding to the bold-faced value of M_e) are used for simulations in the remaining sections.

[1]Since its code is not publicly available, we implemented Q-TCP according to the "TCPLearning" algorithm in [16] to our best ability.

Table 3: Throughput-delay tradeoff metric M_e as a function of the decision threshold th of LP, at various network configurations.

		$M_e(th)$					
(L, K)	th_0	th = 0.1	th = 0.3	th = 0.5	th = 0.7	th = 0.9	$th = th_0$
(5, 1)	0.12	**2.1191**	2.1174	2.1174	2.1153	1.9539	2.1170
(50, 1)	0.38	1.8816	1.8817	1.9056	1.9058	**1.9065**	1.8817
(150, 1)	0.3	1.7970	1.7970	1.7970	1.7970	**1.8529**	1.7970
(50, 4)	0.54	0.5242	0.5235	0.5501	0.5540	**0.5624**	0.5537
$(50, 4)_{\text{mix}}$	0.54	1.1024	1.2940	1.3112	1.2908	0.9029	**1.3354**

4.2 Single Sender

In this section, we evaluate the performance of LP-TCP and RL-TCP, when there is only a single sender with "always-on" traffic in the dumbbell network. First, we demonstrate the importance of credit assignment in RL-based TCP CC schemes.

4.2.1 Effect of credit assignment in RL-based TCP CC schemes. In Table 4, we compute the throughput and delay of a single "always-on" sender using various TCP CC schemes, over 10 iterations. For RL-TCP, $\delta_1 = 0.01, \delta_2 = 0.1$. We observe that with the proposed credit assignment, throughput and/or delay are improved, and a better M_e is achieved. Fig. 2a shows cwnd of Q-TCP, Q-TCP$_{ca}$, RL-TCP$_{\text{no-ca}}$, and RL-TCP during one simulation. With action space tailored to under-buffered networks and the proposed credit assignment, RL-TCP learns quickly, while Q-TCP encounters frequent time-out due to large cwnd modifications in its original action space, which is not suitable for under-buffered situations. With our proposed credit assignment, Q-TCP$_{ca}$ learns and rescues from the consequences of unfavorable actions at some states, but still encounters time-out from time to time.

Table 4: Effect of credit assignment for Q-TCP, Q$_a$-TCP and RL-TCP. Buffer size L is 50 packets.

	$E(tp)$	$V(tp)$	$E(d)$	$V(d)$	M_e
Q-TCP	6.176	0.2674	16.26	4.6624	1.541
Q-TCP$_{\text{ca}}$	**9.597**	8.7210e-3	**20.31**	3.6909	**1.960**
Q_a-TCP	9.658	0.0191	14.8	2.818	1.998
Q_a-TCP$_{\text{ca}}$	**9.857**	8.100e-5	**3.74**	3.240e-2	**2.156**
RL-TCP$_{\text{no-ca}}$	9.723	9.3010e-3	13.87	3.1521	2.011
RL-TCP	**9.869**	7.490e-4	**3.86**	3.240e-2	**2.154**

4.2.2 Varying buffer size. Varying the buffer size L at router 1, we compare the performance of various TCP CC schemes in Table 5. It shows that LP-TCP benefits most from small L: when compared with NewReno at $L = 5$, LP-TCP shows 29% increase on throughput with RTT close to RTT_{min}. But when L increases, LP-TCP selects large th (see Table 3) for better M_e, and performs similar to NewReno. When $L = 50$, RL-TCP achieves $7-8\%$ decrease in RTT and at least 9% increase in throughput, compared to NewReno and Q-TCP. Table 5 suggests that to fully utilize a 10Mb/s bottleneck link, LP-TCP only needs buffer size as small as 5 packets. RL-TCP needs larger buffer size (e.g., $L = 50$), but it also fully utilizes the bandwidth while keeping RTT close to RTT_{min}. However, NewReno needs buffer size of 150 packets, while having delay as high as 82ms.

Table 5: Average performance of a single TCP sender using NewReno, Q-TCP, Q$_a$-TCP, LP-TCP, or RL-TCP, with varying buffer size L.

	L	$E(tp)$	$V(tp)$	$E(d)$	$V(d)$	p	M_e
NewReno	5	7.490	0.	1.900	0.	1.4e-4	1.949
Q-TCP	5	0.652	1.32e-2	3.101	1.14e-2	4.8e-2	-0.541
Q_a-TCP	5	5.262	1.08e-1	2.504	6.38e-3	1.05e-3	1.568
LP-TCP	5	**9.675**	2.5e-5	**4.5**	9.0e-2	2.319e-5	**2.119**
RL-TCP	5	**7.666**	8.03e-2	**2.514**	5.80e-3	2.9e-4	**1.944**
NewReno	50	8.910	0.	15.98	1.60e-3	2.6e-4	1.910
Q-TCP	50	6.198	3.38e-1	17.24	5.750	3.11e-3	1.539
Q_a-TCP	50	9.503	3.68e-2	16.10	4.113	2.1e-4	1.973
LP-TCP	50	**8.91**	0.	**16.54**	3.04e-2	2.645e-4	**1.906**
RL-TCP	50	**9.762**	9.96e-4	**3.885**	4.807e-2	1.8e-4	**2.142**
NewReno	150	9.920	0.	82.50	0.	7.8e-4	1.853
Q-TCP	150	8.873	8.62e-2	71.91	40.53	1.46e-3	1.755
Q_a-TCP	150	9.909	4.74e-4	29.58	91.52	7.6e-4	1.954
LP-TCP	150	**9.920**	0.	**82.78**	7.6e-3	7.867e-4	**1.852**
RL-TCP	150	**9.919**	4.75e-6	**4.041**	4.92e-2	7.5e-4	**2.154**

4.3 Multiple Senders

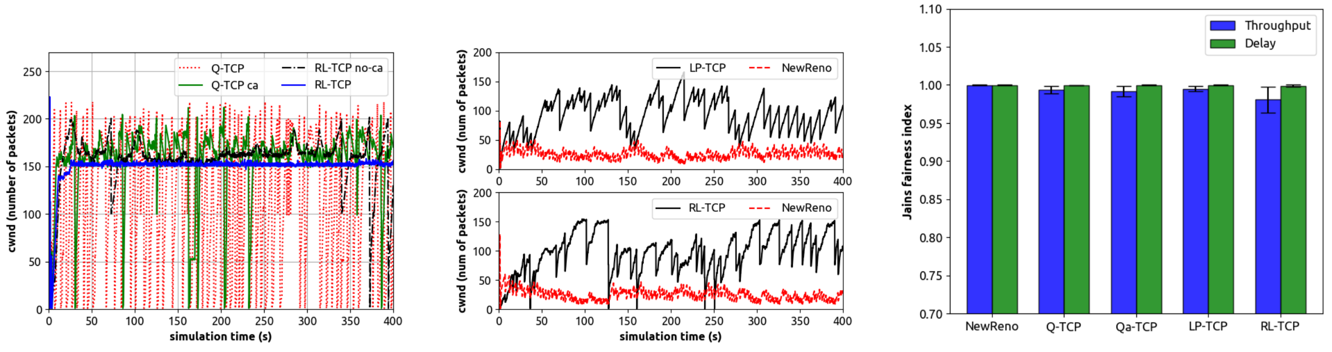
Now we consider the situation when there are multiple senders in the dumbbell network.

4.3.1 "Always-on" senders. First we let the four senders be "always-on". Table 6 shows the performance per sender when all senders use the same TCP CC scheme. For RL-TCP, $\delta_1 = 0.2, \delta_2 = 0.1$. Compared to NewReno, LP-TCP improves throughput by 5% with slightly increased delay, and RL-TCP increases throughput by 5% with a similar delay. RL-TCP gives the best throughput and delay tradeoff in terms of M_e, albeit having a slightly higher variance of its performance.

Table 6: Average performance per sender when four "always-on" senders adopting the same CC scheme coexist in the bottleneck link. $L = 50$.

	$E(tp)$	$V(tp)$	$E(d)$	$V(d)$	p	M_e
NewReno	2.296	3.58e-4	17.34	9.673e-3	1.43e-3	0.545
Q-TCP	1.836	2.96e-2	20.37	8.094e-1	1.888e-2	0.306
Q_a-TCP	2.290	4.32e-2	20.30	7.483e-1	2.33e-3	0.527
LP-TCP	**2.412**	2.931e-2	**24.04**	2.189e-1	1.33e-3	**0.562**
RL-TCP	**2.409**	1.17e-1	**17.67**	8.183	8.2e-4	**0.592**

4.3.2 On-Off senders. In this case, all four senders adopt the on-off traffic model. When a sender is on, it sends a number of bytes according to an exponential distribution with a mean of 10MB. It then turns off for a time period according to an exponential distribution with a mean of 0.5s. We show average performance per sender when all senders adopt the same TCP CC scheme in Table 7. Buffer size is of 50 packets. For RL-TCP, $\delta_1 = 0.4, \delta_2 = 0.1$. It can be seen that Q-TCP does not perform well in this dynamic situation. Q$_a$-TCP, however, performs noticeably better than Q-TCP, with a throughput comparable to that of NewReno, and a slightly increased delay. LP-TCP achieves a higher throughput and lower packet loss rate than NewReno, also at the expense of slightly increased delay. RL-TCP achieves the highest M_e, with slightly larger variance in its performance.



(a) (b) (c)

Figure 2: (a) The changes of cwnd for Q-TCP, Q-TCP$_{ca}$, RL-TCP$_{no\text{-}ca}$, and RL-TCP. $L = 50$ and $K = 1$. (b) cwnd of LP-TCP/RL-TCP and average cwnd of three NewReno senders. $L = 50$. (c) Jain's fairness index of throughput and delay of TCP senders.

Table 7: Average performance per sender when four on-off senders adopting the same CC scheme coexist in the bottleneck link. $L = 50$.

	$E(tp)$	$V(tp)$	$E(d)$	$V(d)$	p	M_e
NewReno	2.305	6.18e-3	16.42	1.003e-1	1.43e-3	0.555
Q-TCP	1.837	2.722e-2	19.91	6.668e-1	1.813e-2	0.309
Q_a-TCP	2.296	2.767e-2	19.20	5.945e-1	2.194e-3	0.535
LP-TCP	**2.394**	3.126e-2	**21.56**	3.677e-1	1.343e-3	**0.565**
RL-TCP	**2.399**	4.234e-2	**20.21**	5.871	8.277e-4	**0.574**

4.4 Fairness

In this section, we evaluate intra-protocol and inter-protocol fairness of the learning-based TCP CC schemes.

4.4.1 Intra-protocol fairness. To evaluate intra-protocol fairness achieved by LP-TCP and RL-TCP ($\delta_1 = 0.2, \delta_2 = 0.1$), we use Jain's fairness index [12],

$$J(x_1, x_2, ..., x_K) = \frac{\left(\sum_{i=1}^{K} x_i\right)^2}{K \sum_{i=1}^{K} x_i^2}, \tag{6}$$

where x_i, $i = 1, ..., K$ represents the throughput (or delay) of the i-th sender during one iteration. The buffer size $L = 50$ and $K = 4$. We compute 100 samples of fairness index from 100 iterations, and plot the mean and variance in Fig. 2c. We see that NewReno achieves the best fairness (close to one). LP-TCP and RL-TCP achieve less optimal fairness index, and RL-TCP shows to be the least fair. This implies that fairness of LP-TCP and RL-TCP in under-buffered wired networks has room for improvement and worth further investigations.

4.4.2 Inter-protocol fairness. We evaluate the competitiveness of each learning-based TCP CC by letting it coexist with three other NewReno senders. With buffer size of 50 packets, we show the cwnd of different senders in Fig. 2b, and their performance in Table 8. We see that both LP-TCP and RL-TCP ($\delta_1 = \delta_2 = 0$) take more than three times of bandwidth compared to that of NewReno, which makes them extremely competitive in coexisting with NewReno flows. This fact indicates a potentially strong incentive for rolling out these new transport protocols. Both Q-TCP and Q_a-TCP are also able to take more bandwidth than NewReno, however with a much smaller margin. It is worth mentioning that the action space tailored to under-buffered networks helps with Q-TCP in improving throughput and reducing delay in this network configuration, hence resulting in a better throughput-delay-tradeoff (from 0.554 to 0.643).

Table 8: Average performance of NewReno and Q-TCP/Q_a-TCP/LP-TCP/RL-TCP when three NewReno and one Q-TCP/Q_a-TCP/LP-TCP/RL-TCP coexist in the bottleneck link. $L = 50$.

	$E(tp)$	$V(tp)$	$E(d)$	$V(d)$	p	M_e
NewReno	1.748	5.178e-3	12.947	6.998e-1	2.339e-3	0.302
Q-TCP	2.342	1.826e-2	19.358	6.902e-1	1.376e-2	0.554
NewReno	2.178	7.650e-3	17.762	4.007e-1	1.571e-3	0.491
Q_a-TCP	2.553	4.874e-2	19.002	4.409e-1	2.164e-3	0.643
NewReno	1.450	9.131e-3	23.65	7.558e-1	3.037e-3	0.055
LP-TCP	**5.275**	9.175e-2	**26.45**	1.659	3.764e-4	**1.335**
NewReno	1.466	1.45e-1	26.504	5.217	3.03e-3	0.054
RL-TCP	**5.238**	1.0192	**29.254**	10.72	3.9e-4	**1.318**

5 CONCLUSIONS

In this paper, we propose two learning-based TCP congestion control schemes for wired networks, one based on supervised learning (LP-TCP), and another based on reinforcement learning (RL-TCP). We evaluate the performance of both schemes in NS2 and compare them to NewReno, Q-TCP, and Q_a-TCP. By adjusting a decision threshold, LP-TCP provides a better tradeoff on throughput and delay compared to NewReno. RL-TCP, using our proposed credit assignment, learns effectively in dynamic network environments, and achieves better throughput and/or delay when compared to Q-TCP, Q_a-TCP and NewReno at various network configurations.

This paper also raises awareness on several issues during the design of learning-based TCP CC schemes. First, the performance of learning-based TCP CC schemes may be parameter-sensitive. For example, the action space of RL-TCP may need a different design depending on the network configurations. It also impacts the competitiveness of RL-TCP over NewReno. Second, both learning-based TCP CC schemes can be improved further in achieving fairness among multiple senders in networks with an under-buffered bottleneck link.

REFERENCES

[1] Guido Appenzeller. 2005. *Sizing router buffers*. Ph.D. Dissertation. Stanford University, Palo Alto, CA.

[2] Saad Biaz and Nitin H Vaidya. 1998. Distinguishing congestion losses from wireless transmission losses: A negative result. In *Proc. 7th International Conference on Computer Communications and Networks*. IEEE, Lafayette, LA, 722–731.

[3] Lawrence S. Brakmo and Larry L. Peterson. 1995. TCP Vegas: End to end congestion avoidance on a global Internet. *IEEE Journal on selected Areas in communications* 13, 8 (1995), 1465–1480.

[4] Mo Dong, Qingxi Li, and Doron Zarchy. 2015. PCC: Re-architecting Congestion Control for Consistent High Performance. In *Proc. 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI'15)*. ACM, Oakland, CA, 395–408.

[5] Sally Floyd and Tom Henderson. 1999. The NewReno modification to TCP's fast recovery algorithm. *RFC 2582* (1999).

[6] Sangtae Ha, Injong Rhee, and Lisong Xu. 2008. CUBIC: a new TCP-friendly high-speed TCP variant. *ACM SIGOPS Operating Systems Review* 42, 5 (2008), 64–74.

[7] Tin Kam Ho. 1995. Random decision forests. In *Proc. the 3rd international conference on Document analysis and recognition*. IEEE, Montreal, Que., Canada, 278–282.

[8] Tin Kam Ho. 1998. The random subspace method for constructing decision forests. *IEEE transactions on pattern analysis and machine intelligence* 20, 8 (1998), 832–844.

[9] Van Jacobson. 1988. Congestion avoidance and control. In *Proc. ACM SIGCOMM*. ACM, Stanford, CA, 314–329.

[10] Van Jacobson. 1990. Modified TCP congestion avoidance algorithm. *note sent to end2end-interest mailing list* (1990).

[11] Raj Jain. 1989. A delay-based approach for congestion avoidance in interconnected heterogeneous computer networks. *ACM SIGCOMM Computer Communication Review* 19, 5 (1989), 56–71.

[12] Raj Jain, Dah-Ming Chiu, and William R Hawe. 1984. *A quantitative measure of fairness and discrimination for resource allocation in shared computer system*. Vol. 38. Eastern Research Laboratory, Digital Equipment Corporation Hudson, MA.

[13] A Jayaraj, T Venkatesh, and C Siva Ram Murthy. 2008. Loss classification in optical burst switching networks using machine learning techniques: Improving the performance of TCP. *IEEE Journal on Selected Areas in Communications* 26, 6 (2008).

[14] Junchen Jiang, Shijie Sun, Vyas Sekar, and Hui Zhang. 2017. Pytheas: Enabling Data-Driven Quality of Experience Optimization Using Group-Based Exploration-Exploitation. In *Proc. 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI'17)*. ACM, BOSTON, MA, 393–406.

[15] Dina Katabi, Mark Handley, and Charlie Rohrs. 2002. Congestion control for high bandwidth-delay product networks. *ACM SIGCOMM computer communication review* 32, 4 (2002), 89–102.

[16] Wei Li, Fan Zhou, Waleed Meleis, and Kaushik Chowdhury. 2016. Learning-based and Data-driven TCP Design for Memory-constrained IoT. In *Proc. International Conference on Distributed Computing in Sensor Systems (DCOSS)*. IEEE, Washington, DC, 199–205.

[17] Dapeng Liu, Youjian Zhao, Kaixin Sui, Lei Zou, Dan Pei, Qingqian Tao, Xiyang Chen, and Dai Tan. 2016. FOCUS: Shedding light on the high search response time in the wild. In *Proc. The 35th Annual IEEE International Conference on Computer Communications (INFOCOM)*. IEEE, San Francisco, CA, 1–9.

[18] Dapeng Liu, Youjian Zhao, Haowen Xu, Yongqian Sun, Dan Pei, Jiao Luo, Xiaowei Jing, and Mei Feng. 2015. Opprentice: Towards practical and automatic anomaly detection through machine learning. In *Proc. the Internet Measurement Conference*. ACM, Tokyo, Japan, 211–224.

[19] Hongzi Mao, Mohammad Alizadeh, Ishai Menache, and Srikanth Kandula. 2016. Resource management with deep reinforcement learning. In *Proc. the 15th ACM Workshop on Hot Topics in Networks*. ACM, Atlanta, GA, 50–56.

[20] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. 2017. Neural adaptive video streaming with Pensieve. In *Proc. ACM SIGCOMM*. ACM, Los Angeles, CA, 197–210.

[21] Mariyam Mirza, Joel Sommers, Paul Barford, and Xiaojin Zhu. 2010. A machine learning approach to TCP throughput prediction. *IEEE/ACM Transactions on Networking* 18, 4 (2010), 1026–1039.

[22] Bruno Astuto Arouche Nunes, Kerry Veenstra, William Ballenthin, Stephanie Lukin, and Katia Obraczka. 2014. A machine learning framework for TCP round-trip time estimation. *EURASIP Journal on Wireless Communications and Networking* 1 (2014), 47.

[23] Lopamudra Roychoudhuri and Ehab S Al-Shaer. 2005. Real-time packet loss prediction based on end-to-end delay variation. *IEEE transactions on Network and Service Management* 2, 1 (2005), 29–38.

[24] Anirudh Sivaraman, Keith Winstein, Pratiksha Thaker, and Hari Balakrishnan. 2014. An experimental study of the learnability of congestion control. In *Proc. ACM SIGCOMM*. ACM, Chicago, IL, 479–490.

[25] Yi Sun, Xiaoqi Yin, Junchen Jiang, Vyas Sekar, Fuyuan Lin, Nanshu Wang, Tao Liu, and Bruno Sinopoli. 2016. CS2P: Improving video bitrate selection and adaptation with data-driven throughput prediction. In *Proc. ACM SIGCOMM*. ACM, Florianópolis, Brazil, 272–285.

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

[27] C-H Tai, Jiang Zhu, and Nandita Dukkipati. 2008. Making large scale deployment of RCP practical for real networks. In *Proc. The 27th Conference on Computer Communications*. IEEE, Phoenix, AZ, 2180–2188.

[28] Kun Tan, Jingmin Song, Qian Zhang, and Murad Sridharan. 2006. A compound TCP approach for high-speed and long distance networks. In *Proc. IEEE INFOCOM*. IEEE, Barcelona, Spain, 1–12.

[29] Zheng Wang and Jon Crowcroft. 1991. A new congestion control scheme: Slow start and search (Tri-S). *ACM SIGCOMM Computer Communication Review* 21, 1 (1991), 32–43.

[30] Christopher John Cornish Hellaby Watkins. 1989. *Learning from delayed rewards*. Ph.D. Dissertation. King's College, Cambridge, Cambridge, UK.

[31] Keith Winstein and Hari Balakrishnan. 2013. TCP ex machina: Computer-generated congestion control. In *Proc. ACM SIGCOMM*. ACM, Hong Kong, China, 123–134.

[32] Yasir Zaki, Thomas Pötsch, Jay Chen, Lakshminarayanan Subramanian, and Carmelita Görg. 2015. Adaptive congestion control for unpredictable cellular networks. In *Proc. ACM SIGCOMM*. ACM, London, United Kingdom, 509–522.