International Journal For Advanced Research In Science & Technology
A peer reviewed international journal www.ijarst.in
ISSN: 2457-0362

OPTIMIZING TCP CONGESTION CONTROL THROUGH MACHINE LEARNING

K. V.RAJESH [1], R. JYOSTHANA [2], SK. PARVEEN [3], SK. RIZWANA[4], SK. SEEMA[5]

[1]Assistant Professor,Department Of CSE,Malla Reddy Engineering College For Women,Hyderabad,Telangana,India

[2,3,4,5] UG Scholar,Department Of CSE,Malla Reddy Engineering College For Women,Hyderabad,Telangana,India

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.

KEYWORDS

TCP congestion control, packet loss prediction, reinforcement learn-ing, machine learning

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

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?* Heuris- tics 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 predic- tion accuracy, but requires sophisticated human design. Recently, capability of machines to learn and represent complex models is re-discovered and exploited to

International Journal For Advanced Research In Science & Technology
A peer reviewed international journal www.ijarst.in
ISSN: 2457-0362

solve various problems in computer networks [17–20, 25]. Thus, we develop a loss predictor (LP) us- ing supervised learning, and incorporate it into the TCP CC to predict and reduce congestive packet losses. With tuning of a deci- sion 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 rein- forcement 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).

The rest of the paper is organized as follows. Section 2 presents the related work. Section 3 presents the architecture for the pro- posed 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 avoid- ance. 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 scal- able 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 per- formance 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 perfor- mance 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.

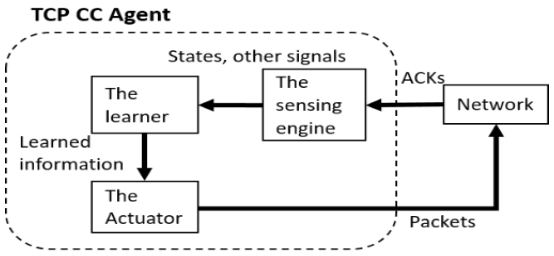
As we consider under- buffered networks with a small number of TCP connections, it is helpful to adopt more expressive features and redesign the

International Journal For Advanced Research In Science & Technology
A peer reviewed international journal www.ijarst.in
ISSN: 2457-0362

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 bottle neck 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, learn ing 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.



a)

International Journal For Advanced Research In Science & Technology
A peer reviewed international journal
www.ijarst.in
ISSN: 2457-0362

Figure 1: (a) Architecture of the proposed machine learning based TCP CC schemes.

We 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.

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.

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 con- tinuously 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 assign- ment of reward according to TCP dynamics. The inputs of the three components in Fig. 1a are as follows:

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

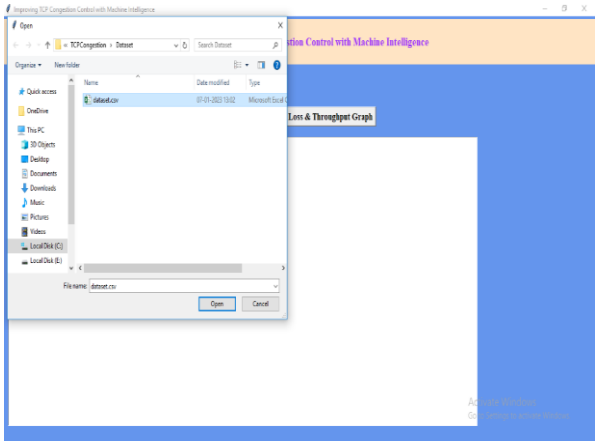
Table 1: RL-TCP (during congestion avoidance)

(S1)	$a_0 = a_1 = a_2 = 0, s_0 = s_1 = s_2 = 0, u_1 = u_2 = 0, t_0 = 0$
(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)	$Q(s_0, a_0) \xleftarrow{\alpha_{now}} r_2 + \gamma Q(s_1, a_1)$
(S9)	$a_0 = \epsilon$-greedy (s_2), $cwnd = cwnd + a_2/cwnd$
(S11)	**Else**
(S12)	$cwnd = cwnd + a_1/cwnd$

4 EXPERIMENTAL RESULTS

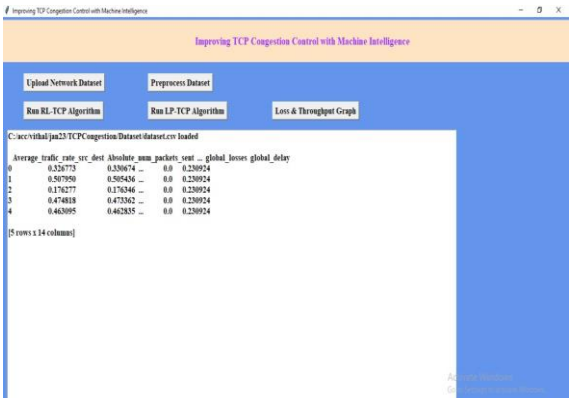
- To run project double click on 'run.bat' file to get below screen

- In above screen click on 'Upload Network Dataset' button to upload dataset and get below output

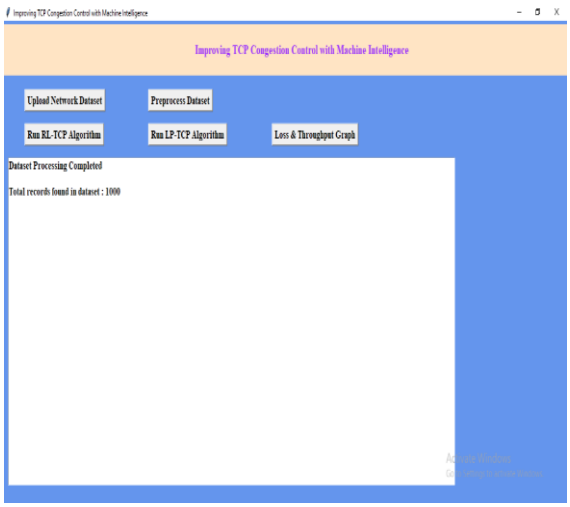


International Journal For Advanced Research In Science & Technology
A peer reviewed international journal
www.ijarst.in
ISSN: 2457-0362

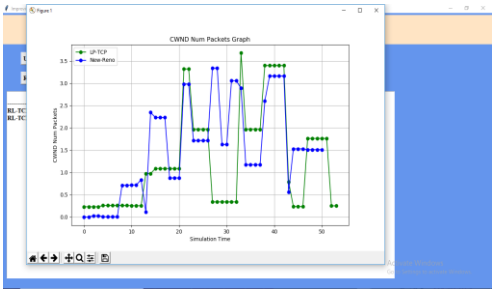
- In above screen selecting and uploading dataset and then click on 'Open' button to load dataset and get below output



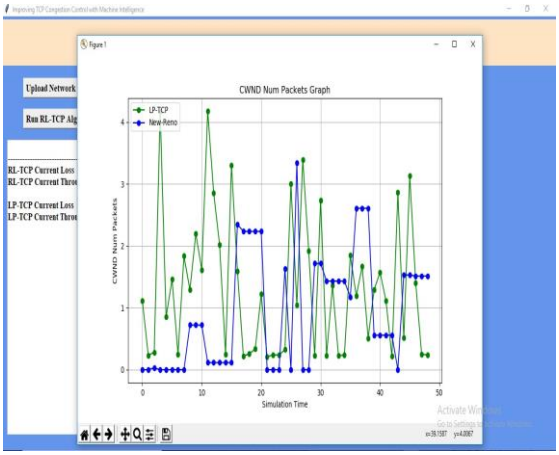
- In above screen dataset loaded and now click on 'Preprocess Dataset' button to remove missing values and get below output



- In above screen Preprocessing completed and dataset contains 1000 records and nowclick on 'Run RL-TCP Algorithm' button to train RL-TCP and get below output

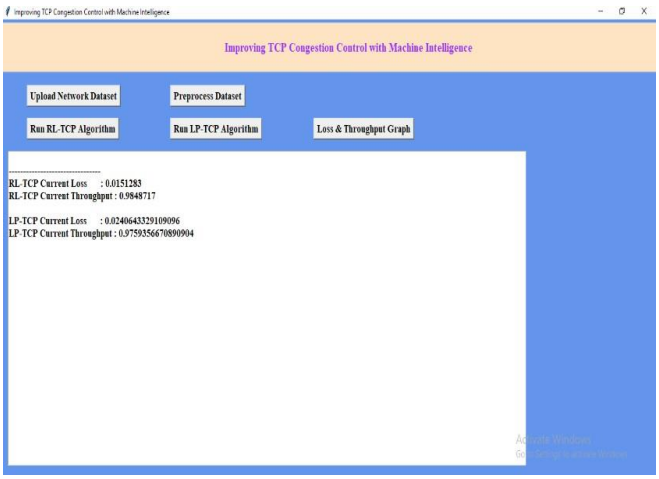


- In above graph x-axis represents Simulation Time and y-axis represents CWND window size and for each packet sending we got learning or prediction rate for existing New-Reno (blue colour line) and propose RL-TCP (green colour line) and in above graph we can see RL-TCP got more packet prediction compare to existing New-Reno.

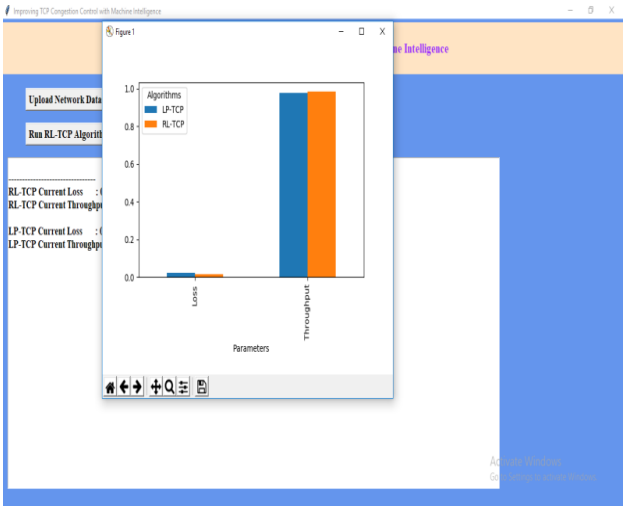


International Journal For Advanced Research In Science & Technology
A peer reviewed international journal www.ijarst.in
ISSN: 2457-0362

- In above screen RL-TCP loss is 0.015 and its throughput is 0.98% as it is processing more packets due to less time in prediction so its throughput will be high and now click on



- In above screen with RL-TCP we got throughput as 0.98 and with LP-TCP we got 0.97 so RL-TCP is better than all other algorithms and now click on 'Loss &Throughput Graph' button to get below outpu



- In above screen we can see LP-TCP (green line) is also better than existing New-Reno to handle congestion and now close above graph to get below screen

- In above graph x-axis represents algorithm names and y-axis represents LOSS and throughout where orange bar is for RL-TCP and blue bar is for LP-TCP and in both algorithms RL-TCP got high throughput and less LOSS. So we can say with RL-TCP we can improve congestion to get less loss and high throughput.

5. 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 wire- less 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).