J. Parallel Distrib. Comput. 71 (2011) 863–878

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

Learning-TCP: A stochastic approach for efficient update in TCP congestion window in ad hoc wireless networks

Venkataramana Badarla [a,*], C. Siva Ram Murthy [b]

[a] Indian Institute of Technology Rajasthan, Jodhpur 342 011, India

[b] Department of Computer Science and Engineering, Indian Institute of Technology Madras, Chennai 600 036, India

ARTICLE INFO

Article history:
Received 21 March 2010
Received in revised form
20 December 2010
Accepted 22 December 2010
Available online 20 January 2011

Keywords:
Learning automata
TCP over ad hoc networks
Experimental evaluation
Reliable data transport

ABSTRACT

In this work, we attempt to improve the performance of TCP over ad hoc wireless networks (AWNs) by using a learning technique from the theory of learning automata. It is well-known that the use of TCP in its present form, for reliable transport over AWNs leads to unnecessary packet losses, thus limiting the achievable throughput. This is mainly due to the aggressive, reactive, and deterministic nature in updating its congestion window. As the AWNs are highly bandwidth constrained, the behavior of TCP leads to high contentions among the packets of the flow, thus causing a high amount of packet loss. This further leads to high power consumption at mobile nodes as the lost packets are recovered via several retransmissions at both TCP and MAC layers. Hence, our proposal, here after called as Learning-TCP, focuses on updating the congestion window in an efficient manner (conservative, proactive, and finer and flexible update in the congestion window) in order to reduce the contentions and congestion, thus improving the performance of TCP in AWNs. The key advantage of Learning-TCP is that, without relying on any network feedback such as explicit congestion and link-failure notifications, it adapts to the changing network conditions and appropriately updates the congestion window by observing the inter-arrival times of TCP acknowledgments. We implemented Learning-TCP in *ns*-2.28 and Linux kernel 2.6 as well, and evaluated its performance for a wide range of network conditions. In all the studies, we observed that Learning-TCP outperforms TCP-Newreno by showing significant improvement in the goodput and reduction in the packet loss while maintaining higher fairness to the competing flows.

1. Introduction

Ad hoc wireless networks are formed dynamically by a collection of mobile nodes in the absence of any fixed infrastructure. In these networks, communication between any two nodes that are not within the radio range of each other takes place in a multi-hop fashion, with other nodes acting as routers. These networks are typically characterized by unpredictable and unrestricted mobility of the nodes and the absence of a centralized administration. The AWNs are considered as resource constrained networks because of the limited bandwidth in the network and limited processing and battery powers at the mobile nodes. However, these networks are very useful (or essential) in military and emergency rescue operations, where the existing infrastructure may be unreliable or even unavailable. These are also useful in commercial applications, such as on-the-fly conferences and electronic classrooms. Extensive research work on ad hoc wireless networking has been carried out on issues, such as medium access and routing [8]. In this paper, we focus on the issues related to reliable and adaptive data transmission over AWNs.

TCP [28] is the defacto protocol that provides reliable, end-to-end, connection-oriented transport service over the Internet. It uses a window, called the congestion window (*cwnd*) to deal with the congestion in the network. It has two phases namely, slow-start and congestion avoidance, for increasing the congestion window. In slow-start phase, on receipt of a TCP acknowledgment (Ack), it increases the *cwnd* by one Maximum segment size (MSS). In the slow-start phase, TCP doubles the *cwnd* for every round-trip time (RTT) until the *cwnd* reaches a threshold, called slow-start threshold. Once the *cwnd* exceeds this threshold, TCP enters into congestion avoidance phase, in which for each Ack, it increases the *cwnd* by a fraction of the MSS. The TCP receiver sends a duplicate Ack for every out-of-order packet it receives. On receiving three consecutive duplicate Acks, interpreting the packet loss due to congestion, the TCP sender invokes the congestion control mechanism, which halves the current *cwnd* and retransmits the lost packet. If the TCP sender does not receive an Ack before the retransmission timeout (RTO), it goes into slow-start phase and in this case *cwnd* is reset to one MSS.

* Corresponding author.
E-mail addresses: b.v.ramana@gmail.com, ramana@iitj.ac.in (V. Badarla), murthy@iitm.ac.in (C.S.R. Murthy).

0743-7315/$ – see front matter © 2010 Elsevier Inc. All rights reserved.
doi:10.1016/j.jpdc.2010.12.012

1.1. Motivation

Since TCP was designed for wired networks, using TCP in its present form for the AWNs leads to unnecessary packet losses and thereby affecting the achievable throughput. Several research works studied the behavior of TCP over the AWNs [13,16,27,2,14,3,12,7]. The main reasons for the poor performance of TCP over AWNs are explained in detail below.

- *Aggressive cwnd increase*: In AWNs, due to the aggressive nature of TCP, the average *cwnd* of TCP becomes much larger than that of optimal *cwnd* [11]. Because of this, the number of TCP packets in transit will be much higher than that of the one the network can handle, thereby leading to increased contentions among the packets of a flow at successive hops and causing a reduction in throughput. Hence, TCP should follow a conservative approach to increase its *cwnd*, thus achieving better spatial channel reuse in the network and improving its performance.
- *Deterministic cwnd update:* The deterministic approach used by TCP for updating the *cwnd* is not suitable for AWNs as the available bandwidth is low and the network conditions change frequently. For instance, say x is the smallest possible increment (for TCP it is $\frac{1}{cwnd}$) in congestion window. In some cases we may need to increase the *cwnd* by the amounts smaller than x. That is, we may need a more conservative increase in the congestion window. Hence, such fixed values make the protocol less responsive to the dynamic changes in network conditions.
- *Deterministic actions:* TCP increases the *cwnd* as long as it receives Acks and decreases the *cwnd* only when packet loss is detected via an RTO or triple duplicate Acks, thus it is reactive rather than proactive for avoiding congestion or contentions. However, as AWNs are resource constrained a proactive approach for avoiding losses is preferable. Further, the deterministic approach leads to the problem of *congestion window synchronization*.[1]

Due to these reasons, TCP often creates congestion in the network and in turn experiences a high amount of packet loss. As TCP needs to resend all these lost packets, this high packet loss not only reduces the throughput, but also consumes the resources of the AWNs, which are severely constrained by the bandwidth and battery power. Another problem is that TCP invokes its congestion control for both congestion and wireless losses as it cannot distinguish between them. This unnecessary reduction in *cwnd* for even wireless losses adversely affects the throughput.

1.2. Our contribution

In order to improve the performance of TCP over AWNs, in this paper we propose a novel *cwnd* updating mechanism that uses learning automata [23] to learn the network's state and accordingly update the congestion window. Our proposal does not seek any explicit support from either the receiver or the intermediate nodes on the path. This is desirable because in reality one cannot expect changes at all the nodes in the network. The goal of Learning-TCP, is to improve the throughput while reducing the packet loss and maintaining the fairness among the competing flows. We achieve this goal through the following mechanisms.

- *Finer and flexible update in congestion window:* In Learning-TCP, the amount of increase or decrease in *cwnd* is not fixed as in TCP. Because of this, Learning-TCP overcomes the problems caused by the aggressive and deterministic nature of TCP in updating the congestion window. When required, it can increase its *cwnd* by more than one MSS and it can even decrease its *cwnd* by few bytes.
- *Stochastic action selection:* In Learning-TCP, the action selection (i.e., increasing or decreasing *cwnd*) is stochastic rather than deterministic. That is, when incipient congestion is detected, Learning-TCP can decrease the congestion window even on receiving Acks. This directly helps Learning-TCP to take proactive measures to counter the congestion and reduce the packet loss due to congestion. Further, due to randomization in the action selection, for a similar state of congestion, the nodes that use Learning-TCP behave differently, thereby preventing the *cwnd* synchronization.

The selection of learning automata for our problem is motivated by the following three reasons. First, the learning algorithms in learning automata are fairly simple and also the amount of state information to be maintained and the number of computations to be done are significantly low [23]. Second, learning automata does not require any modeling of the environment in which it operates because it does not need any knowledge about the environment, and also any analytical knowledge of the function to be optimized [1]. Finally, the solutions that use learning automata have been proven to *converge* to an optimal value [30].

The rest of the paper is organized as follows. Section 2 discusses the related work. Section 3 gives a brief introduction to Learning Automata and an overview of Learning-TCP. Section 4 presents the design details of Learning-TCP. Section 5 presents the performance evaluation of Learning-TCP. Finally, Section 6 summarizes our work.

2. Related work

Several proposals [16,5,18,20,33,32,29,25,10,22,31] have been made to address the issues related to reliable transport over AWNs. These proposals follow different approaches and can be classified broadly into network dependent and network independent approaches. The proposals [16,5,18,20,33,32,29] are network dependent since they rely on explicit feedback for situations such as congestion and link failure, from the network.

The proposals TCP-ELFN [16], TCP-F [5], TCP-BuS [18], ATCP [20], TCP-DOOR [32], and TCP-EPLN-BEAD [33] improve the TCP performance in AWNs by relying on explicit feedback from the network about route failure, congestion, and available bandwidth. ATP [29], a non-TCP variant, also relies on the explicit feedback from the network. However, such strong dependency on explicit feedback from the network, adversely affects their performance when the required feedback from the network is unavailable or unreliable which is more common in AWNs. In addition, these are not practically feasible as the deployment of these proposals require modification at all the nodes in the network. Hence, the focus of current research is on network independent approach, wherein the protocols, without any explicit feedback, infer the network conditions and take necessary actions via observing variations in RTTs or inter-arrival times of TCP Acks. The proposals [9,25,10,22,31] fall under this category.

Fixed-RTO [9] aims at faster recovery of losses due to path breaks. It argues that consecutive RTOs at the sender are due to route failure rather than congestion. So when two consecutive RTOs occur, it disables the regular exponential backoff mechanism of TCP and allows the sender to retransmit at regular intervals.

Fuzzy-TCP [25] employs a fuzzy logic based loss distinguishing mechanism which uses the rate of change in RTTs and the

[1] This is a well-known problem of TCP. All the competing flows keep increasing their *cwnd* and when the buffer at bottleneck node overflows, all the flows that experience loss due to this buffer overflow halve their congestion windows. This problem leads to alternate periods of over-utilization and under-utilization at the bottleneck link. This can be overcome by introducing randomization in the network. [6] proposes spacing of successive segment transmissions of TCP with a random interval of time, thus adding some randomization and reducing the effect of *cwnd* synchronization. Though Learning-TCP does not focus on transmission times of the segments, it introduces randomization in the *cwnd* updating mechanism, thus reducing the effect of *cwnd* synchronization.

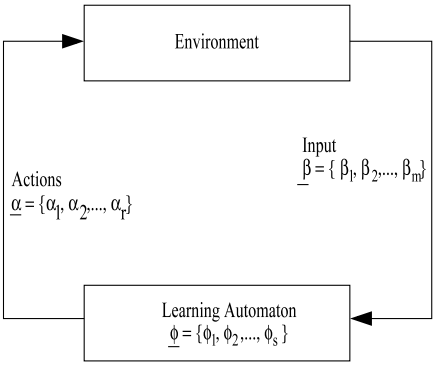


Fig. 1. Learning automaton interacts with the environment by executing an action α_i and learns the environment conditions through the responses from the environment β_i.

number of hops (obtained from the underlying routing protocol) information. It detects congestion when RTT increases n times successively, in which every increment is larger than a given α (n and α are predefined). The wireless losses are detected when the mean RTT is small.

TCP-AP [10] proposes rate based transmission of TCP packets while keeping the TCP intact. The rate is estimated from the RTTs and the hop length of TCP connection, which accounts for both congestion and spatial reuse constraint in multi-hop wireless networks. However, in its rate estimation, it assumes uniform bandwidth at all wireless nodes.

TCP-FeW [22] studies the impact of congestion and MAC contention on the interactions between TCP and on-demand routing protocols. It also proposes a fractional window increment scheme to limit the TCP's aggressiveness and to prevent its over-reaction on the routing protocols. FeW is effective only in the congestion avoidance phase and it increases the *cwnd* by a growth rate factor α (packets) for every RTT, where $0 < \alpha < 1$ is a tunable parameter. However, as it probes the network slowly, it will not show any performance gains in lightly loaded network conditions. Also for high packet loss or for short-lived flows, as it spends a longer time in the slow-start phase, it will not gain much from its conservative increase mechanism.

Our earlier work, FALA based Learning-TCP (TCP-FALA) [31] uses finite action-set learning automata (FALA) for updating the congestion window. Here, the action-set consists of five actions: multiplicative increase/decrease, additive increase/decrease, and inaction. However, as the action-set size is limited, it is difficult to map the range of responses provided by the environment to the appropriate actions; also as these networks are highly bandwidth constrained, much finer update in the *cwnd* is required.

3. Overview of learning-TCP

In this section, we first give a brief introduction to Learning Automata followed by an overview of our protocol, Learning-TCP.

3.1. A brief introduction to learning automata

The theory of learning automata consists of a learning automaton which provides a simple model for adaptive decision making with unknown random environments [23]. The learning automaton interacts with the environment by selecting an action from a set of actions. When a specific action is performed, the environment provides either a favorable or an unfavorable response. The response can be either a binary value, a finite number of discrete values, or continuous values over a finite interval. However, in practice, it may be important to have finer distinctions (discretization) in the response, in order to perceive the system better. These finer distinctions would help in providing the extent of favorability of a response to a particular action. The selection of action could be either deterministic or stochastic. In the latter case, probabilities are maintained for each possible action, which are updated with the reception of each response from the environment. The objective in the design of the learning automaton is to determine how the previous actions and responses should affect the choice of the current action to be taken, and to improve or optimize some predefined objective function. Fig. 1 shows the interactions between the automaton and the environment.

A learning automaton can be formally described in terms of the following:

1. *State* of the automaton at any instant n, denoted by $\phi(n)$, is an element of the finite set $\underline{\Phi} = \{\phi_1, \phi_2, \ldots, \phi_s\}$, where s represents the number of states.
2. *Output* (or) *action* of an automaton at any time instant n, denoted by $\alpha(n)$, is an element of the finite set $\underline{\alpha} = \{\alpha_1, \alpha_2, \ldots, \alpha_r\}$, where r represents the number of actions.
3. *Input* (or) *Feedback* of an automaton at any time instant n, denoted by $\beta(n)$, is an element of the finite or infinite set $\underline{\beta} = \{\beta_1, \beta_2, \ldots, \beta_m\}$ or $\underline{\beta} = \{(a, b)\}$, where m is the number of discrete values in the response from the environment and a and b are real numbers.
4. *Transition function* $F(.,.)$ determines the state at the time instant $(n+1)$, in terms of the state and input at any time instant n and could be either deterministic or stochastic. $\phi(n+1) = F[\phi(n), \beta(n)]$.
5. *Output function* $G(.)$ determines the output of the automaton at any time instant n, in terms of the state at that instant and could be either deterministic or stochastic. $\alpha(n) = G[\phi(n)]$.

The automaton is called deterministic if both F and G are deterministic, and is called stochastic (variable-structure) otherwise. For mathematical simplicity, it is generally assumed that each state corresponds to one distinct action. Hence, the automaton can be represented by the triple $\{\underline{\alpha}, \underline{\beta}, A\}$, where A is called the *updating function*. In this paper, we use the terms *updating function* and *learning algorithm* interchangeably.

The objective of the updating function is to enable the automaton to *learn* the state of the environment based on the feedback obtained and choose the best possible action at any point of time. It should be able to efficiently guide the automaton to quickly adapt to the changes in the environment. The updating function needs to be simple, and yet efficient, especially when the environment is known to change rapidly.

The models of learning automata can be classified based on the number of actions in the action set ($\underline{\alpha}$). The FALA is one such model which contains a finite number of actions and each action corresponds to a range of responses provided by the environment. However, such discretization may not be possible in all the situations as the discretization may be either too coarse for the problem or a finer discretization may result in a large number of actions due to which the time to update the action probabilities increases and the decision making becomes difficult.

A natural choice in such a case, would be to use continuous action-set learning automata (CALA) [30] which has an infinite number of actions. It maintains an action probability distribution, which follows a normal distribution with mean μ and standard deviation σ, rather than action probability for each action. At any time instant n, the updating function of CALA has to update $\mu(n)$ and $\sigma(n)$ based on the response from the environment for its previously performed action. Then the automaton selects an action x, where x is a real number chosen from this action probability distribution. Unlike FALA, as CALA maintains only the action probability distribution, updating $\mu(n)$ and $\sigma(n)$, and decision making process are fairly simple. Details about the updating function used in CALA are provided in Section 4.

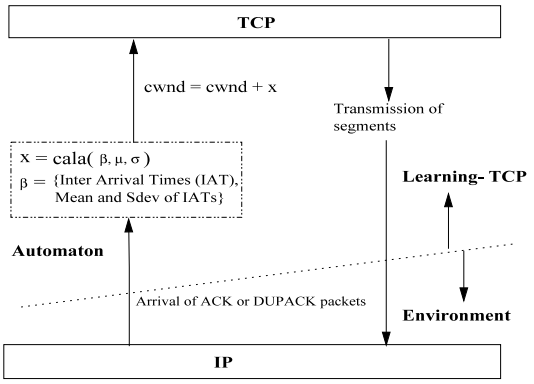


Fig. 2. Learning-TCP consists of TCP and a CALA learning automaton which infers the network conditions from the inter arrival times of TCP Acks and takes the necessary actions.

3.2. An overview of learning-TCP

Learning-TCP uses CALA as it allows finer discretization in the response which results in an accurate mapping of responses to the actions. Further, it contains an infinite number of actions, (i.e., actions are mapped to real line) because of which Learning-TCP can have flexible and finer updates in the congestion window. To avoid the reliance on any explicit feedback about the network conditions from either the receiver or the intermediate nodes on the path, it infers the network conditions by observing the inter-arrival times (iat)[2] of TCP Acks. As the network conditions are reflected in the arrival times of Acks, henceforth, in this paper, we use the term *network response* to represent the iat of the TCP Acks. The key observation is that an Ack which arrives late, generally, indicates congestion. Note that we do not use iat for estimating the effective bandwidth as in TCP-Westwood [21] which is proposed to enhance the TCP performance in last-hop wireless networks. Instead, we use iat of TCP Acks to know the trends in the changing network conditions, that is, whether the network is free from congestion or getting congested. Learning-TCP effectively captures these through the iat values and we validate the effectiveness of this mechanism in Section 5.1.

Fig. 2 shows the relationship between various components of Learning-TCP and the interactions of Learning-TCP with the environment. Learning-TCP consists of TCP and a continuous-action learning automaton; it is compatible with TCP as it does not modify the semantics and header format of TCP. It only changes the *cwnd* updating mechanism. The objective of the learning automaton is to determine the number of bytes by which the *cwnd* should be increased or decreased at each time instant by learning the congestion level in the network. The automaton has been implemented as a separate module within the TCP. Like the other TCP variables, such as receiver advertised window (snd_wnd) and sequence number of next segment to be sent (snd_nxt), Learning-TCP maintains μ and σ of action probability distribution and other related information for each session (or flow) separately.

The following steps briefly describe the working mechanism of our protocol. They are explained in detail in Section 4.

1. **Capturing the network conditions into a parameter** γ: As we do not seek any explicit feedback about the level of congestion and contention[3] in the network, we use iat of TCP Acks to infer the current network conditions. On receipt of an Ack, the learning automaton at the sender obtains the current level of congestion in the path from the current iat and the history of most recent n iat values, into a parameter γ.
2. **Mapping of** γ **to** β: As γ consists of network conditions and these conditions are influenced by the previous actions taken by the automaton, the input to the CALA learning algorithm, β can be obtained from γ. The notation $\beta(n)$ corresponds to a β value at a time step n. $\beta(n) = 1$ indicates the highest reward and $\beta(n) = 0$ indicates the lowest reward (i.e., penalty).
3. **Updating** μ **and** σ **of action probability distribution:** The learning algorithm uses $\beta(n)$ for updating *mean*, $\mu(n)$ and *standard deviation*, $\sigma(n)$ of the action probability distribution which is assumed to follow a normal distribution.
4. **Action selection:** Using the updated action probability distribution, the automaton selects an action which is essentially an amount of increment or decrement in the congestion window.

Note that, unlike TCP, Learning-TCP does not distinguish between Acks and duplicate Acks while updating the congestion window. The congestion control mechanism is guided strictly by the learning algorithm in Learning-TCP. That is, it does not use loss-based congestion control. However, when a loss is indicated via triple duplicate Acks or retransmission timeout, it retransmits the corresponding segment by keeping *cwnd* intact. In this paper, we use the term *ack* to refer both Acks and duplicate Acks.

4. Design of learning-TCP

4.1. Capturing the network conditions

Since we do not seek any explicit feedback about current congestion and contention levels in the network, along with current iat, we maintain a history of n most recent iat values, to estimate the current network conditions. Then, we compute the average ($mean_n$) and standard deviation ($sdev_n$) of these n iat values. Further, we also compute the average ($mean_k$) over the most recent k iat values, where $k < n$. Shortly, we discuss the need for the $mean_k$. As iat is purely computed over the acks, we may infer the condition wrongly when TCP data and ack packets take different paths. Hence, in this work, similar to [17,26], we assume symmetric paths for the communication between TCP end points. Making such an assumption is a valid one in IEEE 802.11 DCF based AWNs. The reasons are the following. Although wireless links are known for link-asymmetry, bi-directional links should exist to realize the use of RTS-CTS-DATA-ACK handshaking in IEEE 802.11 based AWNs. Hence, when there exist bi-directional links, it is always possible to set up symmetric paths. Further, as symmetric paths are composed of a smaller number of wireless links than asymmetric paths, having the symmetric paths for the communication reduces the probability of route failures for a TCP connection [2]. However, even in the case of symmetric paths, the measured iat values may fluctuate due to transient congestion or high delays between the traffic bursts (i.e., time difference between the last ack in the previous burst and the first ack in the current burst) or MAC level retransmission mechanisms. Since these fluctuations adversely affect the learning mechanism, we reduce the effect of these fluctuations by resetting the iat values that do not fall in the range [*mean-sdev, mean+sdev*] to the appropriate bounds, when *sdev* is high (i.e., $sdev > \frac{mean}{2}$). This mechanism also reduces the effect of confined acks, which arrive with a low iat at the time of congestion.

[2] Time difference between the arrival times of any two successive TCP Acks of a TCP session.

[3] The term congestion refers to a situation of buffer overflow at the intermediate nodes in the network. The term contention refers to a situation at MAC layer which occurs when there are too many nodes trying to access the shared medium in wireless networks. For both congestion and contention situations the response needed is the same. That is, when there is a rise in congestion or contention, the *cwnd* should be decreased and vice versa. Hence, hereafter in this paper, we do not explicitly mention the term contention. Also the term network conditions refers to the level of congestion or contention in the network.

Another problem is that the $mean_n$ sharply rises to a high value and remains over there for a long-lasting congestion in the network. However, in order to learn the network conditions better, the proposed learning mechanism requires $mean_n$ when there is no congestion in the network. Hence, we use a parameter *safeMean*, which is updated by the $mean_n$ when the network state is found to be uncongested. The key idea is that, using the *safeMean* as a reference point we first identify the degree of deviation in the current iat, and then prefer an action for increasing (decreasing) the *cwnd* when the iat is below (above) the *safeMean*. As *safeMean* is an important parameter, first we discuss how to update the *safeMean* with the $mean_n$ when there is no congestion. As mentioned earlier, along with $mean_n$, we also maintain the average of recent k iat values, where $k < n$, in $mean_k$. As $mean_k$ is computed over a smaller number of iat values than $mean_n$, any sudden change in the network conditions reflects immediately in $mean_k$. That is, $mean_k$ rises (or falls) when there is a sudden increase (or decrease) in the congestion in the network. When we take the ratio of these two means (i.e., $\frac{mean_k}{mean_n}$), the ratio will be much above to the value of 1 when the congestion is rising. So, when this ratio is much below to the value of 1, we can infer minimal or no congestion in the network. Therefore, we update *safeMean* with $mean_n$ when this ratio is less than a threshold 0.75; further we take the values for k and n as *20* and *100*, respectively, which are derived via empirical studies. We validate the updating of *safeMean* with $mean_n$ in Section 5.

Upon receiving an ack at time step n, using *safeMean* and current iat, as shown in Eq. (1), the automaton captures the network conditions into a parameter *timeRatio*. From Eq. (1), we can observe that the *timeRatio* value approaches its maximum value of 1 when the iat values are negligible compared to the *safeMean*. Hence, the upper bound for *timeRatio* will be 1. The *timeRatio* approaches a lower bound (δ) when iat values are higher than *safeMean* (i.e., iat $>$ $c \cdot$ *safeMean*, for any c > 1). The selection of a value for c affects the detection of incipient congestion in the path. We prefer a proactive decrease in *cwnd* when iat values are higher than $c \cdot$ *safeMean*. However, as taking a large value for c does not give an indication of incipient congestion in the path, the automaton may not perform a proactive decrease of *cwnd*, and thus experiencing an increasing amount of packet loss and also degradation in throughput. On the other hand, taking smaller values for c leads to an unnecessary reduction in *cwnd* even for smaller fluctuations in iat values. Hence, we take a moderate value for c as 3. Thus, the lower bound of *timeRatio* (δ) becomes -2. When the iat value is higher than $3 \cdot$ *safeMean* then the *timeRatio* is set to -2.

$$\text{timeRatio} = \begin{cases} \dfrac{\textit{safeMean} - \text{iat}}{\textit{safeMean}} & \textit{if } \text{iat} < 3 \cdot \textit{safeMean}, \\ -2 \quad \text{otherwise} \end{cases} \tag{1}$$

$$\gamma = \frac{\textit{timeRatio} - \delta}{1 - \delta}. \tag{2}$$

As shown in Eq. (2), the *timeRatio* is normalized on a linear scale of 0 and 1, and stored as a parameter γ. To summarize the above mechanism, when iat values are negligible compared to *safeMean*, γ gets the values close to 1. The ack that resulted in such values for γ is called *early ack*. When the iat values are significantly higher compared to *safeMean*, we call such ack as *late acks*. In this case, γ gets a value close to 0.

The next step is to map this normalized network response parameter γ to the input parameter of the CALA learning algorithm β. This is done based on the state of the learning automata. We define two states for the automaton namely, *increase* and *decrease* states. As this mapping requires an idea of the CALA learning algorithm, we postpone the discussion on this mapping until Section 4.3.

4.2. The CALA learning algorithm

In this section, we provide the details of the CALA learning algorithm that is used to update the μ and σ of the action probability distribution. In CALA, the number of actions is infinite. However, instead of maintaining action probabilities for each action separately, the action probability distribution which is assumed to follow a normal distribution, is maintained. The functions for updating the action probability distribution are simple, and they do not require any discretization in the network response. Further, there exists proof for the convergence of CALA learning algorithm that follows a normal distribution. The learning algorithm given in [30] is based on two reinforcements from the environment, $\beta_{x(n)}$ and $\beta_{\mu(n)}$, which represent the reinforcement obtained from the environment at time step n, when actions selected are $x(n)$ and $\mu(n)$, respectively. Since we obtain only one reinforcement $x(n)$, we use variants of the original equations. As mentioned in [30], we substitute $\beta_{\mu(n)} = 0$ and obtain the Eqs. (3) and (4) which correspond to the updating functions for $\mu(n)$ and $\sigma(n)$ of the action probability distribution at any time step n, respectively. In our problem, $\mu(n)$ and $\sigma(n)$ represent the mean and deviation of the effective amount of update in the congestion window.

$$\mu(n+1) = \mu(n) + \lambda \frac{\beta_{x(n)}}{\phi(\sigma(n))} \frac{x(n) - \mu(n)}{\phi(\sigma(n))}, \tag{3}$$

$$\begin{aligned} \sigma(n+1) \;=\; & \sigma(n) + \lambda \frac{\beta_{x(n)}}{\phi(\sigma(n))} \left[\left(\frac{x(n) - \mu(n)}{\phi(\sigma(n))} \right)^2 - 1 \right] \\ & - \lambda \cdot K \cdot (\sigma(n) - \sigma_L), \end{aligned} \tag{4}$$

where

$$\phi(\sigma) = \begin{cases} \sigma_L & \text{if } \sigma \le \sigma_L, \\ \sigma & \text{if } \sigma > \sigma_L > 0. \end{cases}$$

In the above equations, $x(n)$ is the action taken at any time step n, λ is the learning parameter controlling the step size ($0 < \lambda < 1$), K is a sufficiently large positive constant, and σ_L is the lower bound on σ.

4.3. Discussion about the learning mechanism

The idea behind the updating functions of the action probability distribution is as follows. It essentially shifts $\mu(n)$ towards $x(n)$ which is the amount of increment or decrement made in *cwnd* at time step n. The shift in $\mu(n)$ towards $x(n)$ should be high when $\beta(n)$ is close to 1. When $\beta(n)$ is close to 0, there will not be any significant change in $\mu(n)$ or $\mu(n)$ is marginally shifted towards $x(n)$. The learning algorithm increases $\sigma(n)$ when $|x(n) - \mu(n)| > \sigma(n)$, otherwise it decreases $\sigma(n)$. There is a reduction term in Eq. (4) that depends on λ and K. The purpose of this reduction term is to shift $\sigma(n)$ towards σ_L.

We define two states, *increase* and *decrease*, for the automaton. When $x(n) - \mu(n) > 0$, the automaton is said to be in the *increase* state. Otherwise it is in the *decrease* state. The normalized network response, γ is treated differently in these two states. In *increase* state, an *early ack* corresponds to positive feedback (γ close to 1) for a faster increase in congestion window. Similarly, for a *late ack* there should not be any increase in μ or the magnitude of increase in μ should be low; thus ensuring no significant change in the effective amount of update in congestion window. Hence, we directly take γ as β ($\beta = \gamma$) when the automaton is in the *increase* state. In the *decrease* state, we map γ to β based on the congestion level in the network. When the network is getting congested (i.e., $\gamma < 0.5$), a significant reduction in μ is required to reduce the *cwnd* drastically. Hence, we take β as 1 and $\mu = \frac{\mu}{2}$. Otherwise, the degree of reduction in μ should be decided based on the extent to which γ is close to 0. Hence, we take $(1 - \gamma)$ as β.

4.4. Convergence proof and selection of parameters

The convergence proof for CALA learning algorithm is well-studied in the literature [30]. The outline of the convergence proof is as follows. The convergence proof is divided into two steps. The first step consists of obtaining an ordinary differential equation (ODE) that approximates the behavior of CALA learning algorithm. Approximating stochastic algorithms, such as CALA learning algorithm by an ODE to understand its long-term behavior is a well studied method in [4,19]. The second step analyzes the asymptotic property of the algorithm based on this approximating ODE to infer the long-term behavior of the algorithm. The convergence proof concludes by showing for small values of σ_L and λ and for a sufficiently high K value, the CALA learning algorithm converges to an optimal value.

In this work, in order to maintain the convergence property, we have taken the values for the parameters λ, σ_L, and K as 0.001 (a small value), $\frac{MSS}{100}$, and 20 000 (a large constant with respect to σ_L), respectively. The intuition behind choice of the values for these parameters is as follows. In all the cases, the rate of increase or decrease in the μ value depends on λ and the degree of favorable response (γ) obtained from the network. In all our simulations, we have fixed the λ value as 0.001. The selection of the learning parameter has a trade-off between the accuracy of action selection and speed of learning. Lower values of λ improve the accuracy of learning. They avoid the unnecessary reduction in μ for the short term fluctuations in the network. Higher values of λ cause the automaton to adapt to the changes in the network rapidly. However, in this case the accuracy is low, further any short term fluctuations in the network immediately cause either a sudden rise or fall in the *cwnd* which adversely affects the achievable goodput. The value of K affects σ. For a specific λ value, the smaller values for K shift σ to σ_L very slowly. As a result, it takes several loss cycles, where a loss cycle begins with slow-start phase and ends with an RTO, to shift σ towards σ_L. In order to shift σ to σ_L in fewer cycles, thus reducing losses, for the λ value of 0.001, we take the K value as 20 000. As σ_L represents the lower bound on σ, a larger value for σ_L results in larger increments and decrements in the congestion window. However, as we prefer a conservative approach for updating the *cwnd*, such larger updates in *cwnd* leads to either an increase in packet loss or a decrease in achievable throughput. Hence, we take the value for σ_L as $\lceil \frac{MSS}{100} \rceil$, thus enabling the learning automaton to make finer updates in the congestion window. For instance, the amount of update in *cwnd* can be as low as 15 bytes for an MSS value of 1500 bytes.

4.5. Detailed algorithm of learning-TCP

The working mechanism of Learning-TCP (which is executed by the sender for each *ack* it receives from the receiver) is given in Algorithm 1. Typically, at the connection start-up, the learning algorithm must be provided with a sufficiently large action set. Hence, it initializes the mean (μ) and standard deviation (σ) of the probability distribution with 1 MSS and 0.00068 MSS (i.e., one byte when MSS = 1460 bytes), respectively. It will be in the loop until the end of the TCP session or abrupt connection close due to 12 successive retransmission timeouts.

5. Performance evaluation of learning-TCP

We implemented Learning-TCP in *ns*-2.28 [24] and also in Linux kernel 2.6, and evaluated its performance for varying link bit error rate, hop length, number of flows, and mobility. In our simulations, we have considered a grid topology of sizes 5×5 and 11×11, chain topology, and random topology and for experimental study we have taken a 5-hop chain topology. The transmission and carrier sense ranges of the nodes are set to 250 m and 550 m, respectively. The application, routing, and MAC protocols used are FTP, AODV, and IEEE 802.11 DCF with 2 Mbps, respectively. The nodes use TwoRayGround as a propagation model and drop-tail queues with maximum size of 50 packets. The MSS is taken as 1460 bytes. All the simulation and experimental studies are conducted for 25 and 10 random seed values, respectively, and all the results are obtained at 95% confidence level.

The performance of Learning-TCP is compared with that of the TCP-FeW, TCP-FALA, and TCP-Newreno. The metrics used for comparison are

- *Packet loss percentage* is a function of number of data packets transmitted and retransmitted by the TCP sender. It is computed as $\frac{number\ of\ packets\ retransmitted}{number\ of\ packets\ transmitted} \times 100$.
- *Goodput* for a flow is computed as $\frac{amount\ of\ actual\ data}{flow\ completion\ time}$. Note that the *amount of actual data* does not include retransmitted data.
- *Number of RTOs* is the total number of retransmission timeouts experienced by a flow.
- *Average cwnd size* is the average congestion window, which is sampled at every 500 ms.
- *Fairness* of the competing flows in a group is obtained using Jain's fairness formula [11], which is expressed as, fairness index $= \frac{[\sum_{i=1}^{n} x_i]^2}{n \sum_{i=1}^{n} x_i^2}$, where n is the number of flows and x_i represents the throughput of ith flow.

In order to measure the performance of Learning-TCP for short-lived flows we conducted two different studies wherein we varied the mean pause time between the flow arrivals and the density of the number of short-lived flows around the mean file size. Except in the studies related to short-lived flows, in all other studies mentioned in this section, similar to the one in [10], we conducted steady state simulations by discarding packets transmitted in first 50 s of the simulation time and then collected statistics over the next 600 s. For the short-lived flow studies, without discarding any packets, we ran the simulations for 600 s.

Algorithm 1 The working mechanism of Learning-TCP

```
Initialize μ and σ as 1 and 0.00068, respectively.
repeat
    iat ← time now - arrival time of previous ack
    Get network response into timeRatio (Eq. 1)
    Get normalized network response into γ (Eq.2)
    if x(n) > μ(n) then
        β(n) ← γ
    else if x(n) < μ(n) and γ < 0.5 then
        β(n) ← 0, μ(n) ← μ(n)/2
    else
        β(n) ← (1 − γ)
    end if
    Update the μ(n) and σ(n) of the probability distribution (Eqs. 3 and 4)
    Generate a normal random number from this distribution and take it as current action x(n)
    cwnd ← cwnd + x(n)
    Transmit one or more TCP segments based on the updated cwnd
    if sdev > mean/2 and iat ∉ [mean + sdev, mean − sdev] then
        Reset iat to the appropriate bound
    end if
    Compute mean, mean_k, and meanRatio
    if meanRatio < threshold then
        safeMean ← mean
    end if
until (the end of TCP session OR abrupt connection close due to 12 successive RTOs)
```

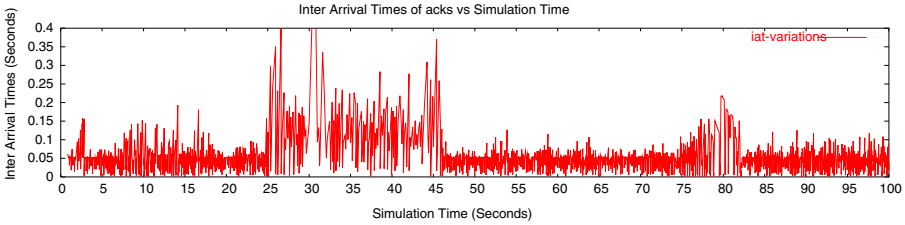


Fig. 3. A 6-hop FTP flow sharing path with a 2-hop CBR flow which operates at 400 Kbps between 25–45 s in a chain topology.

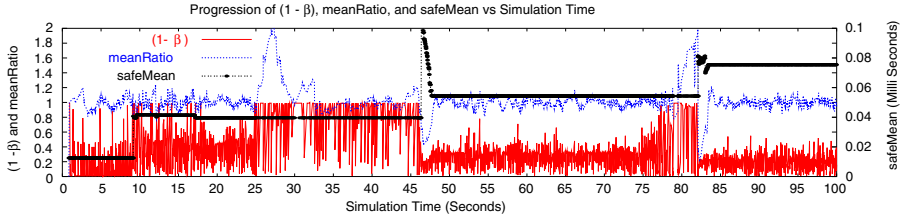


Fig. 4. The corresponding progression of $1 - \beta$, meanRatio, and safeMean.

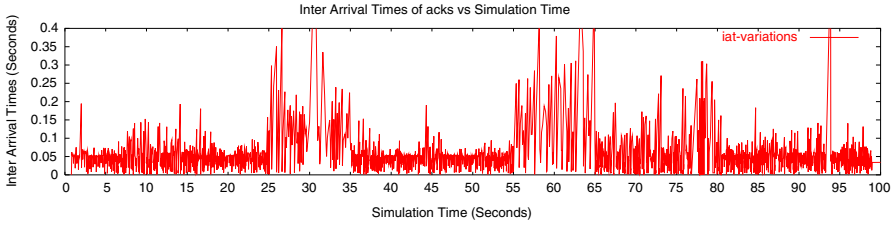


Fig. 5. A 6-hop FTP flow sharing path with a 2-hop CBR flow which operates at 400 Kbps and 200 Kbps between 25–35 and 55–65 s, respectively.

5.1. Validating the behavior of learning-TCP

In this section, with the help of simulations, we show β is able to capture the congestion in the network. We consider a 6-hop FTP flow and a 2-hop CBR flow over a 6-hop ad hoc network. Figs. 3 and 4, show the inter-arrival times vs simulation time and the progression of $1 - \beta$ vs simulation time, respectively, for an FTP flow in the presence of CBR traffic. The CBR traffic is introduced at 25 s for a duration of 20 s. In Fig. 3, we clearly observe the presence of background traffic (CBR traffic) as the iat values of TCP session are very high from 25 to 45 s. The corresponding progression of $1 - \beta$ is shown in Fig. 4, which matches the trends shown by the progression of iat values. We use $1 - \beta$, which is directly proportional to iat, in the graphs, instead of β, for clarity. We observe a clear increase in $1 - \beta$ with the increasing congestion in the network. When Learning-TCP increases the rate of transmission by increasing the *cwnd*, the iat values also increase, which in turn increases $1 - \beta$.

Fig. 4 also shows the progression of the *meanRatio*, which is the ratio of $mean_k$ and $mean_n$. Note that k and n (where $k < n$) define the size of the history that needs to be maintained to take appropriate actions. In all the results presented in this paper, we used the values, which are derived empirically, for k and n as *20* and *100*, respectively. The ratio of $mean_k$ and $mean_n$, *meanRatio*, is close to 1 in presence or absence of congestion in the network for an extended period of time. But *meanRatio* reacts sharply to sudden changes in congestion in the network as these changes get immediately reflected in $mean_k$. That is, $mean_k$, and in turn *meanRatio* rises (or falls) when there is a sudden increase (or decrease) in the congestion. Fig. 4 clearly shows the reactions of *meanRatio* to the congestion level in the network. At the beginning and ending of the CBR flow the changes in *meanRatio* are clearly visible. The figure also shows the changes in *safeMean*. As soon as the network becomes congestion-free, we notice a sudden increase in *safeMean*. This causes β getting the values close to 1, thereby helping the learning mechanism for a rapid increase in congestion window.

Figs. 5 and 6 show similar trends for the same 6-hop FTP flow in the presence of a CBR flow operating at 400 Kbps and 200 Kbps between 25–35 s and 55–65 s, respectively. These results also confirm that β captures the congestion state of the network and helps the learning mechanism to take appropriate action.

In Figs. 7 and 8, we show the progression of *cwnd* vs time, for the above discussed two cases. These plots show how the learning algorithm updates *cwnd* in response to the variations in $1 - \beta$. The trends shown in updating *cwnd* are in conformance with the above discussion. As the learning algorithm favors increasing the *cwnd* for higher values of β (i.e., for lower $1 - \beta$), we observe a clear and steady increase in the congestion window. For lower β values, that is during congestion in the network due to CBR traffic or self traffic, we observe either a small increase, a continuous decrease, or no change in the congestion window. We also notice that, as soon as congestion ends, the learning algorithm is steadily increasing the congestion window.

5.2. Performance of learning-TCP for varying packet error rates

Unlike over wired links, the packet error rate (PER) over wireless links in AWNs can be as high as 10^{-2}. The end-to-end PER will be even higher as the paths are composed entirely of wireless links. Hence, we study the performance of Learning-TCP for a range of PERs. Here, we take a single Learning-TCP flow running over a 8-hop chain topology and vary PER over the wireless links between 0.0001 and 0.01. As we take only a single flow, there will be fewer losses due to congestion/contention in the network. We consider the uniform error model available in *ns*-2.28. As suggested in [15], in order to allow the packets to utilize the link bandwidth before dropping, we introduce the error model at the end of the link

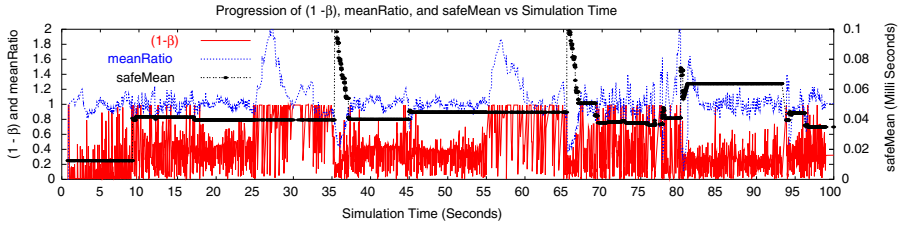


Fig. 6. The corresponding progression of $1-\beta$, meanRatio, and safeMean.

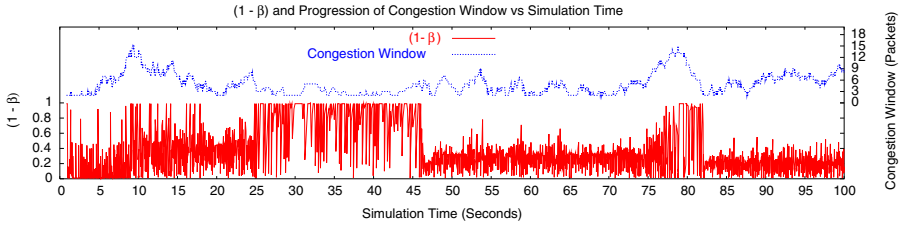


Fig. 7. A 6-hop FTP flow sharing path with a 2-hop CBR flow which operates at 400 Kbps between 25–45 s in a chain topology.

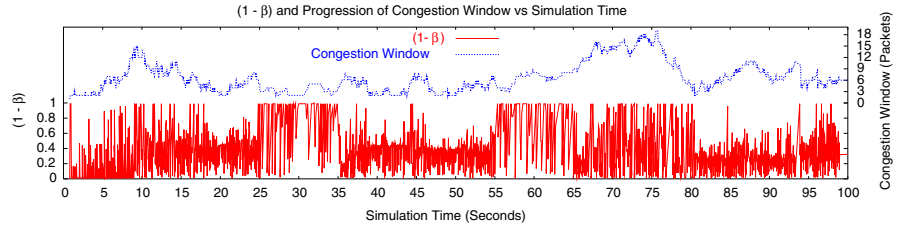
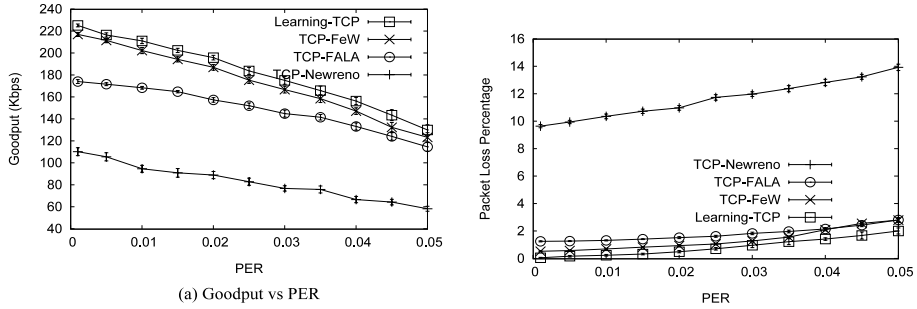


Fig. 8. A 6-hop FTP flow sharing path with a 2-hop CBR flow which operates at 400 Kbps and 200 Kbps between 25–35 and 55–65 s, respectively.



(a) Goodput vs PER. (b) Packet loss percentage vs PER.

Fig. 9. Uniform Error Model: A single flow running over a 8-hop chain.

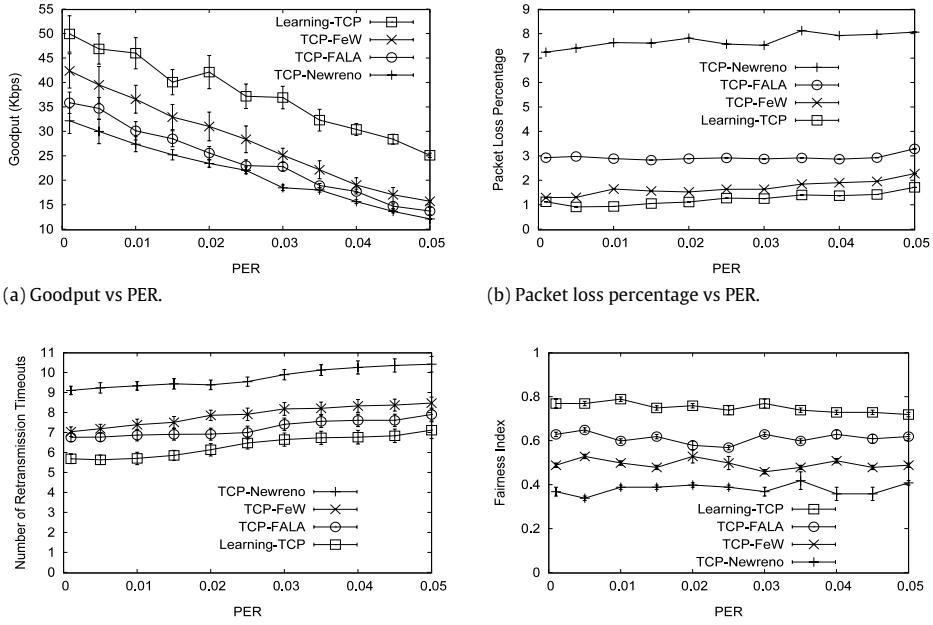
(i.e., over incoming wireless link in NS2). Note that as we introduce this error model at every node, the effective end-to-end packet loss rate increases for increasing hop length. For instance, for a PER (p) of 0.001, the end-to-end packet loss rate ($1-(1-p)^n$, n is number of hops) for a 6-hop and 8-hop chain networks would be 0.0039 and 0.0079, respectively. We do not make any changes to the retransmission mechanism of IEEE 802.11 MAC. Although, this mechanism at MAC may reduce the effect of error model, as the error model is introduced at every node, it shows a significant impact on the performance of a flow.

Fig. 9 shows the goodput and packet loss percentage[4] of Learning-TCP, TCP-FeW, TCP-FALA, and TCP-Newreno for varying PER. We can observe that Learning-TCP performs the best among the four by showing up to 110% higher goodput over TCP-Newreno and up to 6% and 30% higher goodput over TCP-FeW and TCP-FALA, respectively. Learning-TCP, TCP-FALA, and TCP-FeW protocols show significant reduction in packet loss over TCP-Newreno, whereas Learning-TCP shows the maximum reduction in packet loss of about 90%.

We repeat the study for a group of six simultaneous flows over a 5×5 grid topology. In this case, we can see both congestion and wireless-related losses. Congestion occurs at the nodes where the horizontal and vertical flows intersect with each other. Fig. 10 shows the goodput, packet loss percentage, average congestion window, and fairness index results for the protocols considered.

In Fig. 10(a) we can observe the average goodput of six flows for varying PER. At 0.001 PER, the losses would be mostly due to the congestion. In this case, Learning-TCP shows about 35%, 30%, and 16% higher goodput over TCP-Newreno, TCP-FALA, and TCP-FeW, respectively. The goodput of these protocols decreases drastically with increasing PER. Fig. 10(b) shows the packet loss percentages of these protocols for varying PER. Learning-TCP shows a significant

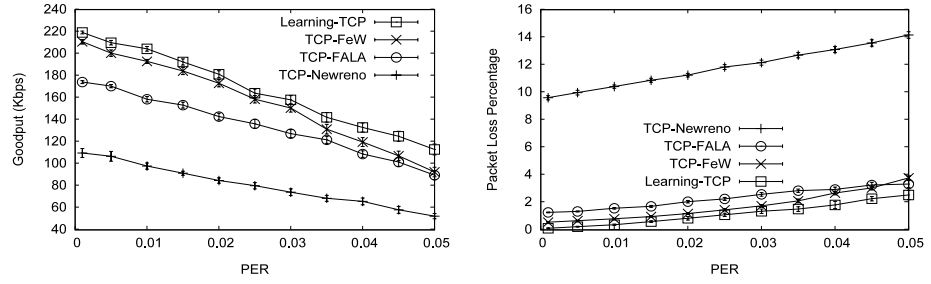
[4] Note that packet loss percentage is different from PER. While PER indicates the rate at which packets are dropped by each node, packet loss percentage denotes an end-to-end metric and it is computed over total number of data packets transmitted and retransmitted by the TCP sender.



(a) Goodput vs PER. (b) Packet loss percentage vs PER.

(c) Number of RTOs vs PER. (d) Fairness index vs PER.

Fig. 10. Uniform Error Model: There are six flows simultaneously running on a 5 × 5 grid topology. Leaving the four corner points, we took three horizontal and three vertical flows of hop length of four each. The inter-node distance is taken as 200 m.



(a) Goodput vs PER. (b) Packet loss percentage vs PER.

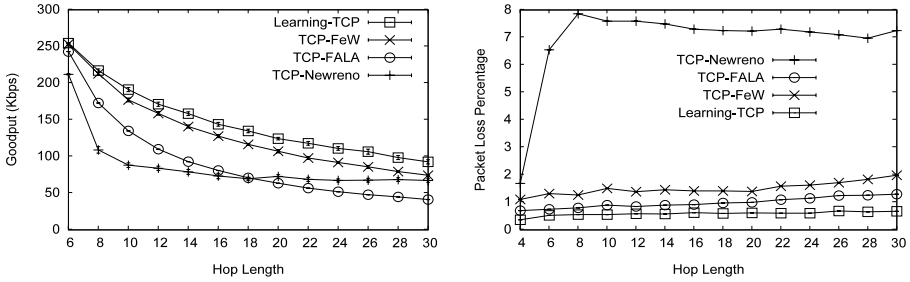
Fig. 11. Two-state Markov Error Model: There is a single flow running over a 8-hop chain.

reduction in packet loss of about 85% over TCP-Newreno and about 34% and 66% reduction over TCP-FeW and TCP-FALA, respectively. Fig. 10(c) shows the average number of timeouts for these protocols. All the protocols show an increase in the number of timeouts with increasing PER. Among the four protocols, Learning-TCP shows a lower number of timeouts. Fig. 10(d) shows the fairness index for these protocols. Learning-TCP shows higher fairness to the competing flows than the other protocols. The improvements in fairness are about 50%, 30%, and 20% over TCP-Newreno, TCP-FeW, and TCP-FALA, respectively. We notice that although TCP-FALA shows lower goodput and higher packet loss over TCP-FeW, it shows better fairness to the competing flows than TCP-FeW. From this we can infer that as both Learning-TCP and TCP-FALA follow a probabilistic action selection, they introduce some randomization in the packet transmission because of which no flow of these protocols would starve due to the aggressiveness of the other competing flows. Hence, they show better fairness over TCP-FeW and TCP-Newreno.

In order to study the effect of bursty transmission errors on the performance of these protocols, we introduce a two-state Markov error model at the end of the link, where the probability of going from the good to the bad state is p and the probability of going from the bad to the good state is q. There is a single flow running over an 8-hop chain and the error model is introduced at each node. Fig. 11(a) and (b) show the comparison of the goodput and packet loss percentage for these protocols for increasing values of p. Here, we take the value of q as 0.85. Like our study with uniform error model, in this case also Learning-TCP shows higher goodput and lower packet loss percentage over the other three protocols. Due to the bursty transmission errors, we observe decrease in goodput and increase in packet loss for these protocols. For the same reason, these protocols would experience higher number of timeouts. As a consequence of this, TCP-FeW (also TCP-Newreno) would often enter into slow-start phase, thereby loses the advantages of its conservative increase mechanism in congestion avoidance phase.

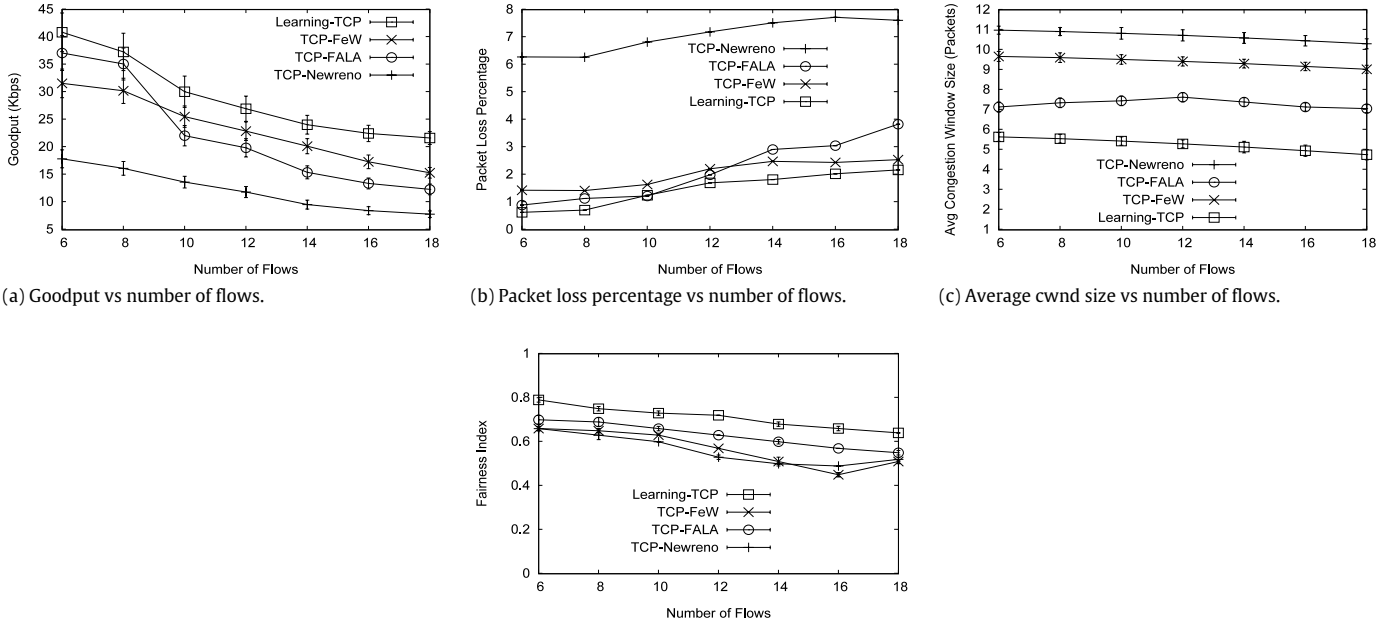
5.3. Performance of Learning-TCP for varying hop lengths

In this section, we study the performance of Learning-TCP on a chain topology with number of hops varying from 4 to 30. Fig. 12(a) and (b) show the goodput and packet loss percentage, respectively for a single flow. We can observe that for all the hop lengths Learning-TCP performs best among the four protocols by showing up to 80% higher goodput and 90% reduction in packet loss over TCP-Newreno. TCP-FeW is doing next by showing up to 70% higher goodput and 85% reduction in packet loss over TCP-Newreno. TCP-FALA shows higher goodput over TCP-Newreno, however at higher hop lengths it shows lower goodput than TCP-Newreno. In Fig. 12(b), we observe a noticeable increase in packet loss percentage for TCP-FeW and TCP-FALA with increasing hop lengths. Learning-TCP and TCP-Newreno do not show any drastic



(a) Goodput vs hop length. (b) Packet loss percentage vs hop length.

Fig. 12. A single flow running over a chain for a fixed uniform PER of 0.001.



(a) Goodput vs number of flows. (b) Packet loss percentage vs number of flows. (c) Average cwnd size vs number of flows.

(d) Fairness index vs number of flows.

Fig. 13. For a group of simultaneous flows over a 11 × 11 grid topology at a fixed uniform PER of 0.001. The inter-node distance is taken as 200 m.

change in the packet loss percentage for increasing hop lengths; however, there is a drastic fall in the goodput of all these protocols for increasing hop lengths. The reason is that the RTTs would increase with increasing hop lengths. As transmission in these protocols is triggered by the arrival of acks, the late arrival of acks would slow down the transmission rates of these protocols, thereby reducing the goodput of these protocols.

5.4. Performance of Learning-TCP for varying load

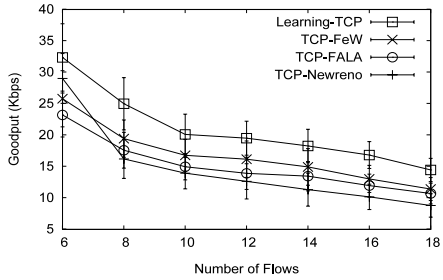
In this section, we evaluate the performance of Learning-TCP for different offered loads via the following studies. First we take a 11 × 11 grid topology and vary the number of simultaneous flows from 6 to 18, and next we repeats the same over a random topology. In both cases, we introduced the uniform error model with fixed PER of 0.001 at each node.

Fig. 13 shows performance of the protocols considered. Fig. 13(a) and (b) show the results of goodput and packet loss percentage for 6 to 18 simultaneous flows. The corresponding average *cwnd* and fairness index results are shown in Fig. 13(c) and (d). We observe that for all the cases, Learning-TCP outperforms other three protocols by showing higher goodput, lower packet loss and number of timeouts, and better fairness to the competing flows. Learning-TCP shows up to 67% higher goodput, 85% reduction in packet loss, and 30% higher fairness over TCP-Newreno, by operating fairly at smaller congestion window. One important observation is that while maintaining smaller *cwnd*, Learning-TCP is able to achieve higher goodput and reduction in packet loss. This smaller *cwnd* further confirms the observations made in [11].[5] Both TCP-FALA and TCP-FeW perform better than TCP-Newreno. At lighter loads (i.e., up to 8 flows) TCP-FALA shows higher goodput over TCP-FeW and for higher loads TCP-FeW fares better. However, TCP-FALA experiences a lower number of timeouts and shows better fairness to the competing flows than TCP-FeW.

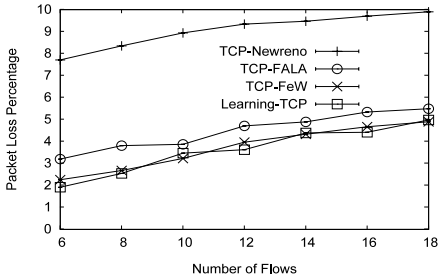
In our second study, we take 75 nodes which are randomly distributed in a terrain size of 500 m × 1500 m to measure the performance of Learning-TCP in an uncontrolled environment. Since the nodes are randomly distributed, there will be a lot of difference in the level of interference, subsequently in the number of collisions, experienced by the flows of these nodes. In this study, the nodes are considered to be static and each flow runs through at least five hops.

Fig. 14 shows the corresponding results for the four protocols. We observe that the trends shown are similar to those of the previous topology. Learning-TCP shows about 65% higher goodput

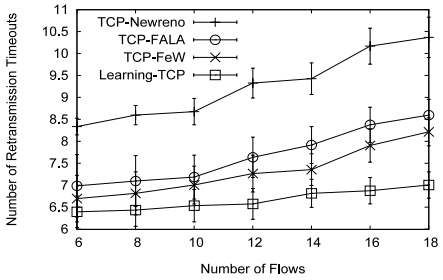
[5] This work identifies the relationship between *cwnd* and hop length *h*. It shows that, TCP achieves maximum throughput when the effective *cwnd* is $\frac{h}{4}$ for longer hop lengths (i.e., $h > 20$). For shorter hop lengths, the effective *cwnd* value is one or two packets larger than $\frac{h}{4}$.



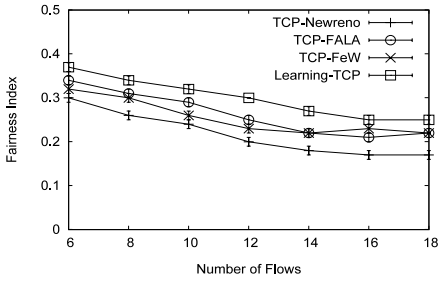
(a) Goodput vs number of flows.



(b) Packet loss percentage vs number of flows.



(c) Number of retransmission timeouts vs number of flows.



(d) Fairness index vs number of flows.

Fig. 14. For a group of simultaneous flows between randomly distributed nodes in a terrain of size 500 m × 1500 m, at a fixed PER of 0.001 in a static network.

and 75% reduction in packet loss over TCP-Newreno. Further, it shows up to 38% reduction in timeouts and 40% better fairness to the competing flows than TCP-Newreno. Both Learning-TCP and TCP-FeW experience the same amount of packet loss, but there is a considerable difference between them in the number of timeouts and fairness. TCP-FeW experiences up to 20% higher timeouts and about 29% lower fairness over Learning-TCP.

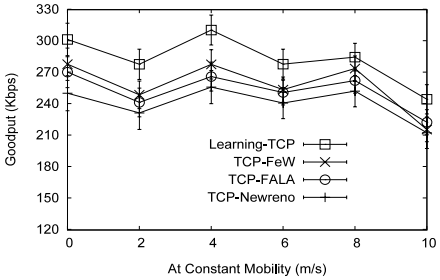
5.5. Performance of Learning-TCP for varying mobility

In this section, we evaluate Learning-TCP over a random topology in the presence of node mobility. Terrain area used is 500 m × 1500 m. Mobility model used is the random-way point model with a pause time of 0 s. The number of nodes in the network is 75. We considered a range of mobilities, static, 2 m/s, 4 m/s, 6 m/s, 8 m/s, and 10 m/s. There are 20 flows starting at a random time chosen from [0,5] s time interval. Each flow runs for 650 s before it terminates.

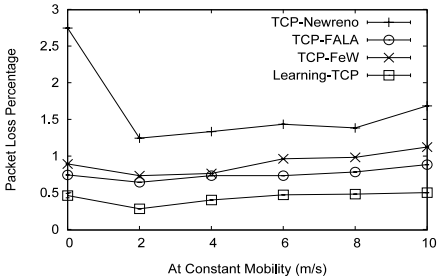
Fig. 15(a) presents the goodput of the protocols considered. It can be noticed that Learning-TCP shows significant improvement in the goodput over other protocols. Fig. 15(b) and (c) show the corresponding packet loss percentage and number of timeouts. From these figures, we make the following observations. Learning-TCP shows a significant reduction in the packet loss over TCP-Newreno, TCP-FeW, and TCP-FALA and all the protocols show lower packet loss at a node mobility of 2 m/s and after that the packet loss increases with increasing mobility. The rate of increase in the packet loss for TCP-FeW and TCP-Newreno is higher than that of Learning-TCP and TCP-FALA. Similar to the packet loss results, the number of timeouts shown by these protocols is a minimum at mobility 2 m/s and after that the number of timeouts increases rapidly. Finally, in Fig. 15(d), we can observe that Learning-TCP maintains the smallest *cwnd*, whereas TCP-FeW and TCP-Newreno maintain a very large congestion window.

5.6. Large number of short-lived flows – varying mean pause time of the flow arrivals

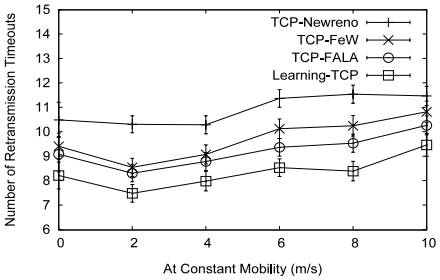
It is well known that TCP traffic comprises a large number of short-lived flows and a small number of long-lived flows, which can be expressed as a Pareto distribution. In order to evaluate the performance of Learning-TCP here, we conduct two different studies. In our first study, we take 100 flows whose flow sizes follow a Pareto distribution with fixed values of mean file size and shape factor, 30 packets and 1.8, respectively and vary the mean pause time between the flow arrivals which follow an exponential distribution. Fig. 16 shows the results for varying mean pause time between the flow arrivals from one second to eight seconds for 100 flows over a 5 × 5 grid topology. Fig. 16(a) shows the average goodput of all the 100 flows for Learning-TCP, TCP-FeW, TCP-FALA, and TCP-Newreno. We can observe that for increasing values of mean pause time between the flow arrivals, the goodput shown by these protocols increases. Further, Learning-TCP shows higher goodput over the other protocols and this improvement increases for increasing values of mean pause time between the flow arrivals. We also observe that TCP-FALA shows a higher goodput over TCP-FeW. Since the flows are short-lived, many of the flows will complete in the slow-start phase of TCP itself. Due to this, TCP-FeW will not gain from its conservative increase mechanism in the congestion avoidance phase. Fig. 16(b) shows the average packet loss percentage by these protocols. Learning-TCP outperforms the other three by showing a significant reduction in the packet loss, and TCP-FALA performs better than the other two. As the mean pause time between the flow arrivals increases, because of the reduction in contentions among the competing flows, we can see a significant reduction in packet loss of up to 90% for Learning-TCP and TCP-FALA. However, for the same case TCP-FeW and TCP-Newreno show just about 11% reduction in packet loss. Fig. 16(c) shows the average number of retransmission timeouts for these protocols. We observe a drastic reduction in timeouts for increasing mean pause times, and Learning-TCP shows the lowest number of timeouts compared to the other three protocols. Finally, Fig. 16(d) shows the corresponding results of average *cwnd* of these protocols, wherein we can observe that Learning-TCP and TCP-FALA maintain their *cwnd* between 2 and 2.5 packets even at higher mean pause times. However, TCP-FeW and TCP-Newreno rapidly increase their *cwnd* with increasing mean pause times, due to which, even though contentions between the flows reduce with increasing pause times, they could not reduce the packet loss percentage as significantly as Learning-TCP and TCP-FALA.



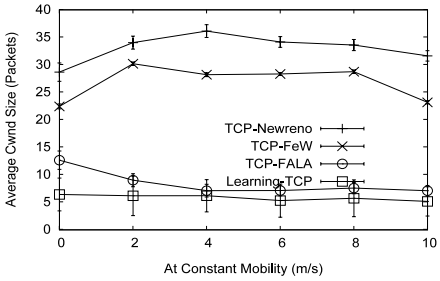
(a) Goodput vs mobility.



(b) Packet loss percentage vs mobility.

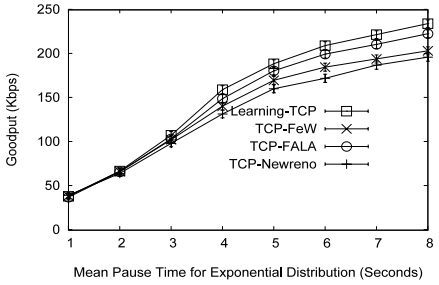


(c) Number of retransmission timeouts vs mobility.

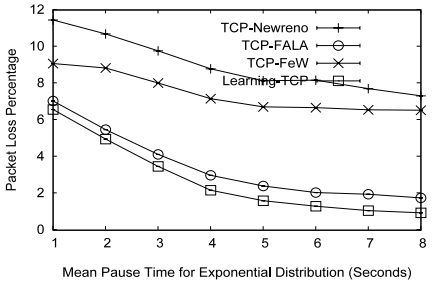


(d) Average cwnd size vs mobility.

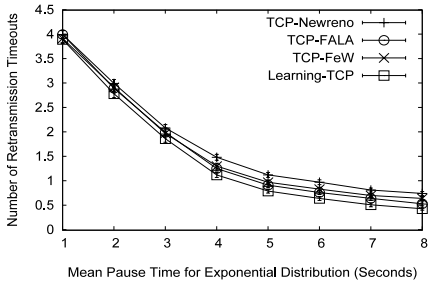
Fig. 15. 20 simultaneous flows in a terrain of size 500 m × 1500 m in which nodes move with random-way point mobility model.



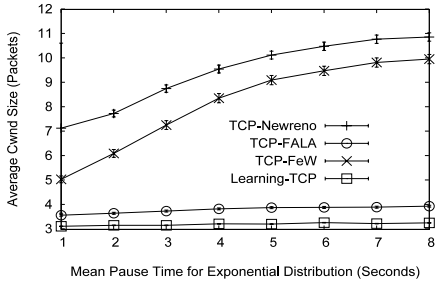
(a) Goodput vs mean pause time.



(b) Packet loss percentage vs mean pause time.



(c) Number of retransmission timeouts vs mean pause time.



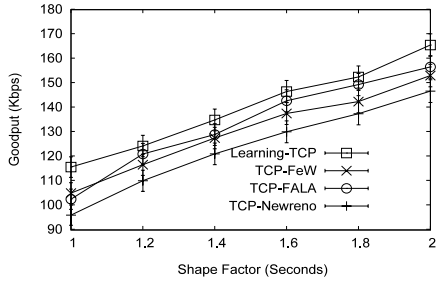
(d) Average cwnd size vs mean pause time.

Fig. 16. For a group of 100 flows running over a 5 × 5 grid topology with mean pause time between the flow arrivals exponentially distributed.

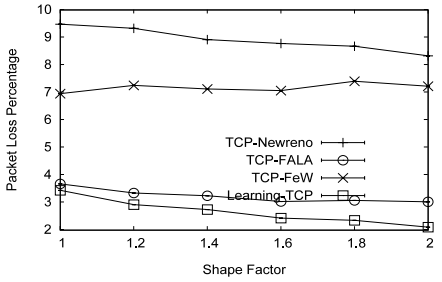
In this study, we observe that Learning-TCP achieves up to 20% and 18% higher goodput over TCP-Newreno and TCP-FeW, respectively, while achieving up to 90% and 85% reduction in packet loss over the TCP-Newreno, TCP-FeW, respectively.

5.7. Large number of short-lived flows — varying shape factor of the flow sizes

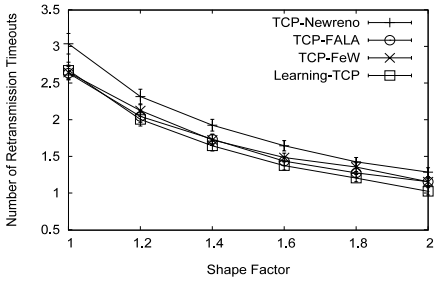
In this section, we conduct a similar study for the fixed values of mean pause time of the exponential distribution (5 s) and mean file size of the Pareto distribution (30 packets) and vary the shape factor of the Pareto distribution between 1 and 2. Since the random numbers (i.e., file sizes in this case) chosen from Pareto distribution have extreme values, varying the shape factor helps us to account for a wide range of scenarios in our simulations. Fig. 17 shows the results of the four protocols for varying shape factor between 1 and 2. Fig. 17(a) shows the goodput for these protocols. We can observe that Learning-TCP performs best among the four protocols for all shape factors and TCP-FALA performs better than the other two protocols with the only exception at shape factor one, where TCP-FeW does slightly better than TCP-FALA. Another observation is that all the protocols show an increase in goodput for increasing values of shape factors. This is due to the reason that as we increase the shape factor, the density of flows with file sizes around the mean file size increases, thus reducing the selection of extreme file sizes and the number of long-lived flows. Fig. 17(b)



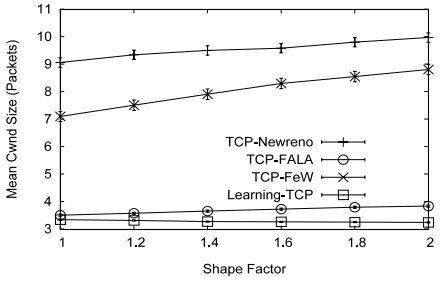
(a) Goodput vs shape factor.



(b) Packet loss percentage vs shape factor.



(c) Number of retransmission timeouts vs shape factor.



(d) Average cwnd size vs shape factor.

Fig. 17. For a group of 100 flows running over a 5 × 5 grid topology where the flow sizes follow a Pareto distribution with fixed mean file size as 30 packets.

shows the packet loss percentage for the four protocols. One clear observation is Learning-TCP, TCP-FALA, and TCP-Newreno have decreasing loss percentages for increasing values of shape factor. However, TCP-FeW shows the opposite trend. It shows a slight increase in loss percentage for increasing shape factors. TCP-FeW gains from its conservative increase mechanism only when there are long-lived flows, so it performs better (just below 7%) at shape factor one. However, as the number of long-lived flows comes down for increasing shape factors, the impact of its conservative increase also comes down. Hence, it increases its loss percentage. In Fig. 17(c), we observe that Learning-TCP shows a lower number of timeouts than the other three protocols and the number of timeouts experienced by all these protocols decreases for increasing values of shape factors. Fig. 17(d) shows the corresponding average *cwnd* of these protocols. TCP-Newreno and TCP-FeW show extremely higher *cwnd* over Learning-TCP and TCP-FALA. Another observation is that TCP-Newreno and TCP-FeW increase their *cwnd* for increasing shape factors. However, due to a lower number of long-lived flows, the rate of increase in the *cwnd* for TCP-FeW is much higher than that for TCP-Newreno. TCP-FALA slightly increases its *cwnd* for increasing values of shape factors, whereas Learning-TCP maintains almost same *cwnd* at all shape factors.

In this study, Learning-TCP outperforms TCP-Newreno, by showing up to 13% higher goodput and up to 67% lower packet loss percentage, while maintaining a lower number of timeouts.

5.8. Discussion on the results

In this section, we discuss the reasons for the poor performance of TCP-Newreno (i.e., traditional TCP), TCP-FeW, and TCP-FALA and the factors that attributed to the superior performance of Learning-TCP.

The main reason for the poor performance of TCP-Newreno is due to its deterministic approach for updating the congestion window. TCP-Newreno strictly increases the *cwnd* even when the ack packets are received with high delays (*late acks*). Moreover, in the slow-start phase, TCP-Newreno increases the *cwnd* by one MSS for every ack packet it receives. Though this aggressive increase helps TCP-Newreno to probe and fill to the maximum network capacity at a fast rate, it leads to heavy loss when the network reaches a highly congested state. Although TCP-FeW follows a conservative increase in *cwnd* in the congestion avoidance phase, it has no mechanism to control the TCP's aggressiveness in slow-start phase. Due to this, TCP-FeW can gain from its conservative increase only for a long-lived flow, that is, when a flow spends considerably longer time in congestion avoidance phase. However in AWNs, due to node mobility and unstable channel conditions, the packet loss and consequently retransmission timeouts will be high. Hence, the flows often enter into a slow-start phase, thus TCP-FeW cannot gain much from its conservative increase mechanism. This fact has been observed in the results provided in the previous sections, where TCP-FeW suffers greatly at higher PER and higher mobility. Similarly, as the flows do not get a chance to spend more time in the congestion avoidance phase, TCP-FeW does not show much performance improvement for short-lived flows.

The *congestion window synchronization* problem affects greatly the fairness of the flows of TCP-Newreno and TCP-FeW. When the flows of these protocols compete for the buffers at the bottleneck node on the path, the shorter hop flow aggressively increases its *cwnd* and occupies a large portion of the shared buffer. This aggressive increase further leads to the buffer overflow at the bottleneck node and causes congestion losses, and forces all the competing flows to halve their *cwnd* even before they reach their fair share of the bottleneck bandwidth. Hence, there are greater chances for the longer hop flows of these protocols getting starved. In Fig. 10(d), 13(d), and 14(d), we can clearly observe this problem. Due to their deterministic *cwnd* updating mechanism, TCP-Newreno and TCP-FeW show greater unfairness to the competing flows, whereas Learning-TCP and TCP-FALA provide better fairness over TCP-Newreno and TCP-FeW. Note that, even in the cases where TCP-FALA achieves lower goodput over TCP-FeW, it shows better fairness over TCP-FeW. The probabilistic action selection approach contributes to the better fairness of Learning-TCP and TCP-FALA. Here, the nodes that detect the incipient congestion (experience higher delays for acks) respond differently. Some nodes affected by the congestion perform decrease actions, while the remaining nodes increase *cwnd* at a low rate. Obviously, the shorter hop flows here cannot completely take away the

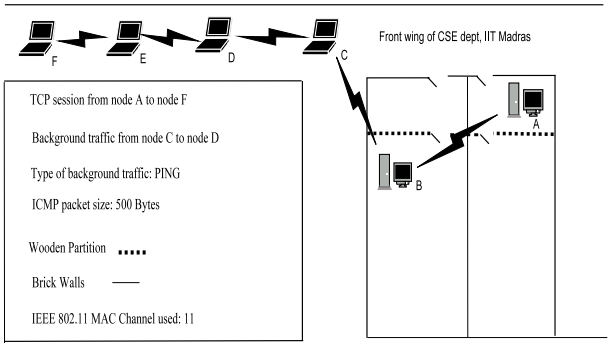


Fig. 18. Experimental Setup: A 6-node ad hoc wireless network is formed with the help of 4 laptops and 2 desktops computers.

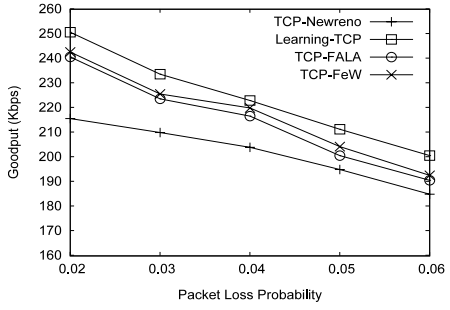
bandwidth at the bottleneck node. Hence, there is a high chance for the longer hop flows to get a fair share of bandwidth at the bottleneck node.

Apart from better fairness, the probabilistic approach helps Learning-TCP and TCP-FALA to act on the incipient congestion in the network proactively. Upon the arrival of *late acks*, the learning automaton in these protocols favors the actions that decrease the congestion window. As a result, Learning-TCP experiences lower packet loss when the network is actually congested. This directly helps to achieve higher goodput and reduces the possibility of congestion, and in turn reduces the number of retransmissions. Due to lower congestion in the network, the control overhead due to the *false* route error messages is also reduced.

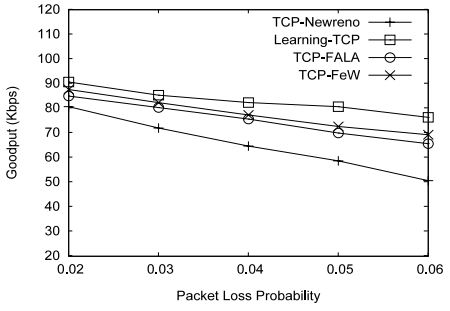
The unique feature of Learning-TCP that makes it outperform all the three protocols is its capability of finer updates in the congestion window. Unlike the other three protocols, the increments and decrements in *cwnd* are not restricted in Learning-TCP. Learning-TCP increases the *cwnd* by higher amounts (more than one MSS) when it perceives the network to be lightly loaded. That is, when the load in the network is relatively low, μ reaches a high value and the *cwnd* is increased by large amounts. However, when the iat of *itacks* are high, the updating algorithm decreases μ. Depending on the congestion level in the network, it can even increase *cwnd* by a few bytes.

5.9. Experimental results

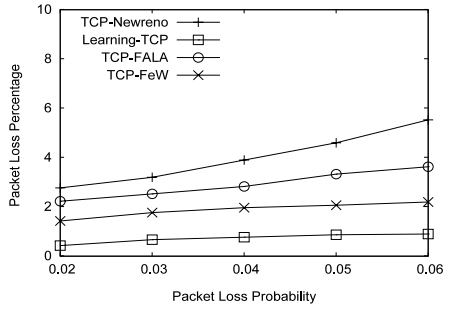
We implemented Learning-TCP in the Linux kernel 2.6 and tested it over a 5-hop testbed as shown in Fig. 18. In this section, we present the details of the experimental setup and experimental evaluation of Learning-TCP. The experimental setup consists of two Linux PCs and four laptops forming a chain topology. The



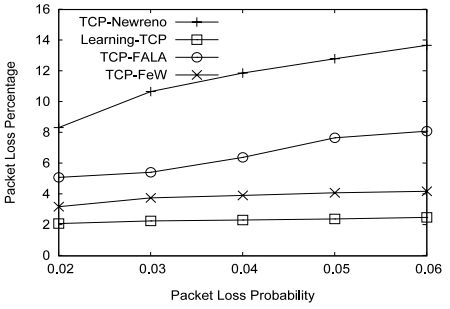
(a) Short-lived flow: goodput.



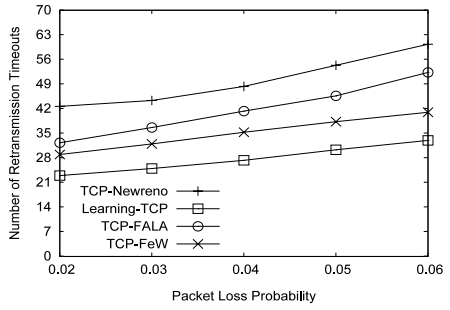
(b) Long-lived flow: goodput.



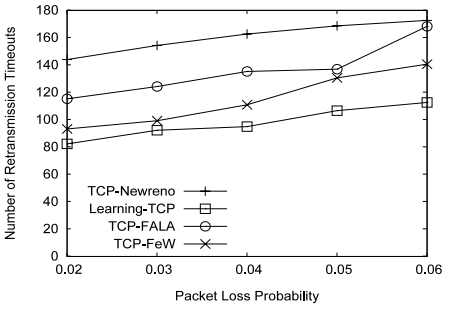
(c) Short-lived flow: % of packet loss.



(d) Long-lived flow: % of packet loss.



(e) Short-lived flow: timeouts.



(f) Long-lived flow: timeouts.

Fig. 19. For a group of 100 flows running over a 5 × 5 grid topology where the flow sizes follow a Pareto distribution with fixed mean file size as 30 packets.

IEEE 802.11 cards used are US Robotics (USR012415) with a prism chipset. The channel number used for transmission is eleven which has a frequency of 2.462 GHz, since it is relatively less loaded than the default channel six. Though the maximum transmission range is up to 800 ft under a maximum power of 20 dBm, in order to form a strict multi-hop network we have set the transmission powers of the cards to −10 dBm. Accordingly, we conducted the experiments by enabling the auto data rate, wherein a node tries to transmit a frame at the maximum permissible data rate by the channel. An FTP session is established between first and last nodes of the chain. The file sizes taken for file transfer are 250 KB (short-lived) and 2 MB (long-lived).

As we do this study on a testbed, unlike in simulations, it is not required to introduce packet loss due to PER. However, there should be congestion in the network. In order to account for congestion, we drop packets with some probability at the network layer of each node. We call this packet loss probability. In this experimental study, by varying the packet loss probability, we measure the goodput, packet loss percentage, and number of retransmission timeouts for TCP-Newreno, Learning-TCP, TCP-FALA, and TCP-FeW.

Fig. 19(a) and (b) show the goodput with varying packet loss probability for the file sizes 250 KB and 2 MB, respectively. The corresponding packet loss percentages are shown in Fig. 19(c) and (d), respectively. Fig. 19(e) and (f) show the number of retransmission timeouts for the same case. In both cases (for short- and long- lived), we observe that Learning-TCP shows higher goodput over TCP-FeW, TCP-FALA, and TCP-Newreno while reducing the packet loss percentage and number of timeouts. The improvements shown through the simulation studies are consistent with the results of the experimental studies mentioned here. We also tested the performance of Learning-TCP for the files sizes 500 KB and 1 MB and observed similar improvements.

6. Conclusions

In this paper, we proposed a novel Learning-TCP for AWNs, which efficiently updates the *cwnd* size without relying on explicit feedback from the network. Learning-TCP adapts to the network conditions by observing the inter-arrival times of TCP *itacks*. We used learning automata due to the reasons that the learning algorithms are fairly simple, the amount of state information to be maintained and the computational requirements are significantly low, and finally they converge to an optimal solution asymptotically. When the incipient congestion is detected, Learning-TCP reduces *cwnd* proactively to avoid multiple losses and timeouts, thereby conserving battery power and bandwidth in the network. Through extensive simulation and experimental studies, we showed that Learning-TCP provides significantly higher goodput over TCP-Newreno while achieving significant reduction in packet loss and better fairness among the competing flows. We also compared Learning-TCP against TCP-FeW and TCP-FALA and observed higher goodput and lower packet loss for Learning-TCP. Learning-TCP neither expects explicit feedback from the network nor requires any changes at the intermediate nodes or at the receiver, thereby making its deployment easier.

Acknowledgments

The authors wish to thank the anonymous reviewers for their valuable comments and suggestions. This work was supported by the Department of Science and Technology, New Delhi, India.

References

[1] W. Abd-Almageed, A. El-Osery, C.E. Smith, Non-parametric expectation maximization: a learning automata approach, in: Proc. IEEE Conference on Systems, Man and Cybernetics, vol. 3, October 2003, pp. 2996–3001.

[2] V. Anantharaman, S.J. Park, K. Sundaresan, R. Sivakumar, TCP performance over mobile ad-hoc networks: a quantitative study, Wireless Communication and Mobile Computing Journal 4 (2) (2004) 203–222.

[3] S. Bansal, R. Shorey, A.A. Kherani, Performance of TCP and UDP protocols in multi-hop multi-rate wireless networks, in: Proc. WCNC, vol. 1, March 2004, pp. 231–236.

[4] A. Benveniste, M. Metivier, P. Priouret, Adaptive Algorithms and Stochastic Approximations, Springer Verlag, New York, 1987.

[5] K. Chandran, S. Raghunathan, S. Venkatesan, R. Prakash, A feedback based scheme for improving TCP performance in ad hoc wireless networks, IEEE Personal Communications Magazine 8 (1) (2001) 34–39.

[6] K. Chandrayana, S. Ramakrishnan, B. Sikdar, S. Kalyanaraman, On randomizing the sending times in TCP and other window based algorithms, Computer Networks 50 (3) (2006) 422–447.

[7] X. Chen, H. Zhai, J. Wang, Y. Fang, TCP performance over mobile ad hoc networks, Canadian Journal of Electrical and Computer Engineering 29 (1) (2004) 129–134.

[8] I. Chlamtac, M. Conti, J.J.N. Liu, Mobile ad hoc networking: imperatives and challenges, Journal of Ad Hoc Networks 1 (1) (2003) 13–64.

[9] T.D. Dyer, R.V. Boppana, A comparison of TCP performance over three routing protocols for mobile ad hoc networks, in: Proc. ACM MobiHoc, October 2001, pp. 56–66.

[10] S.M. ElRakabawy, A. Klemn, C. Lindemann, TCP with adaptive pacing for multihop wireless networks, in: Proc. ACM MobiHoc, May 2005, pp. 288–299.

[11] Z. Fu, H. Luo, P. Zerfos, S. Lu, L. Zhang, M. Gerla, The impact of multihop wireless channel on TCP performance, IEEE Transactions on Mobile Computing 4 (2) (2005) 209–221.

[12] Z. Fu, X. Meng, S. Lu, How bad TCP can perform in mobile ad hoc networks, in: Proc. IEEE International Symposium on Computers and Communications, July 2002, pp. 298–303.

[13] M. Gerla, R. Bagrodia, L. Zhang, K. Tang, L. Wang, TCP over wireless multi-hop protocols: simulations and experiments, in: Proc. IEEE ICC, June 1999, pp. 1089–1094.

[14] A. Gupta, I. Wormsbecker, C. Wilhainson, Experimental evaluation of TCP performance in multi-hop wireless ad hoc networks, in: Proc. IEEE MASCOTS, October 2004, pp. 3–11.

[15] A. Gurtov, S. Floyd, Modeling wireless links for transport protocols, ACM Sigcomm Computer Communication Review 34 (2) (2004) 85–96.

[16] G. Holland, N. Vaidya, Analysis of TCP over mobile ad hoc networks, in: Proc. ACM MobiCom, August 1999, pp. 219–230.

[17] Y. Hu, A. Perrig, D.B. Johnson, Rushing attacks and defense in wireless ad hoc network routing protocols, in: Proc. ACM Workshop on Wireless Security, September 2003, pp. 30–40.

[18] D. Kim, C.K. Toh, Y. Choi, TCP-BuS: Improving TCP performance in wireless ad hoc networks, Journal of Communications and Networks 3 (2) (2001) 59–71.

[19] H.J. Kushner, G.G. Yin, Stochastic Approximation Algorithms and Applications, Springer Verlag, New York, 1997.

[20] J. Liu, S. Singh, ATCP: TCP for mobile ad hoc networks, IEEE Journal on Selected Area in Communications 19 (7) (2001) 1300–1315.

[21] S. Mascolo, C. Casetti, M. Gerla, M.Y. Sanadidi, R. Wang, TCP Westwood: Bandwidth estimation for enhanced transport over wireless links, in: Proc. ACM MobiCom, July 2001, pp. 287–297.

[22] K. Nahm, A. Helmy, C.C. Jay Kuo, TCP over Multihop 802.11 Networks: Issues and performance enhancement, in: Proc. ACM MobiHoc, May 2005, pp. 277–287.

[23] K.S. Narendra, M.A.L. Thathachar, Learning Automata: An Introduction, Prentice Hall, New Jersey, 1989.

[24] The network simulator. http://www.isi.edu/nsnam/ns/.

[25] R. Oliveira, T. Braun, A delay-based approach using fuzzy logic to improve TCP error detection in ad hoc networks, in: Proc. IEEE WCNC, vol. 3, March 2004, pp. 21–25.

[26] P. Papadimitratos, Z.J. Haas, Secure routing for mobile ad hoc networks, in: Proc. SCS CNDS, January 2002, pp. 193–204.

[27] V. Ramarathinam, M. Labrador, Performance analysis of TCP over static ad hoc wireless networks, in: Proc. ISCA PDCS, September 2002, pp. 410–415.

[28] C. Siva Ram Murthy, B.S. Manoj, Ad Hoc Wireless Networks: Architectures and Protocols, Prentice Hall, New Jersey, 2004.

[29] K. Sundaresan, V. Anantharaman, H. Hsieh, R. Sivakumar, ATP: A reliable transport protocol for ad hoc networks, in: Proc. ACM MobiHoc, June 2003, pp. 64–75.

[30] M.A.L. Thathachar, P.S. Sastry, Networks of Learning Automata: Techniques for Online Stochastic Optimization, Kluwer Academic, New Jersey, 2004.

[31] B. Venkata Ramana, B.S. Manoj, C. Siva Ram Murthy, Leaning-TCP: A novel learning automata based reliable transport protocol for ad hoc wireless networks, in: Proc. IEEE BroadNets, October 2005, pp. 484–493.

[32] F. Wang, Y. Zhang, Improving TCP performance over mobile ad hoc networks with out-of-order detection and response, in: Proc. ACM MobiHoc, June 2002, pp. 217–225.

[33] X. Yu, Improving TCP performance over mobile ad hoc networks by exploiting cross-layer information awareness, in: Proc. ACM MobiCom, September 2004, pp. 231–244.

Venkataramana Badarla received the B. Tech. degree in Computer Science and Engineering from Nagarjuna University, India, in 1995 and the M.E. degree in Systems and Information from Birla Institute of Technology and Science (BITS), Pilani, India, in 1997. Between 1997–02, he worked as a software engineer and a faculty member. He worked on his PhD during 2002–07 in the Department of Computer Science and Engineering at the Indian Institute of Technology (IIT) Madras, India, where he focused on provisioning of efficient data transport over ad hoc wireless networks. During 2006–07 he was a project officer at IIT, Madras, India. From May 2007 to Dec 2010 he worked as a research fellow at the Hamilton Institute, National University of Ireland Maynooth, Ireland. Currently he is working as an assistant professor at IIT Rajasthan. He received a Best paper of the conference award from 14th IEEE Conference on Networks (ICON 2006). His research interests include experimental evaluation of the MAC and routing protocols over multi-hop wireless networks and sensor networks.

C. Siva Ram Murthy received the PhD degree from the Indian Institute of Science, Bangalore, India. He has been with the Department of Computer Science and Engineering, Indian Institute of Technology Madras, Chennai, India, since 1988, where he is currently a professor. His research interests include parallel and distributed computing, real-time systems, lightwave networks, and wireless networks. He has published more than 125 papers in refereed international journals and more than 125 papers in refereed international conferences in these areas. He has coauthored the textbooks Parallel Computers: Architecture and Programming, (Prentice-Hall of India, New Delhi, India), New Parallel Algorithms for Direct Solution of Linear Equations, (John Wiley & Sons, Inc., New York, USA), Resource Management in Real-time Systems and Networks (MIT Press, Cambridge, USA), WDM Optical Networks: Concepts, Design, and Algorithms (Prentice Hall, New Jersey, USA), and Ad Hoc Wireless Networks: Architectures and Protocols (Prentice Hall, New Jersey, USA). He is a recipient of Best Ph.D. Thesis Award from the Indian Institute of Science, Indian National Science Academy (INSA) Medal for Young Scientists, Dr. Vikram Sarabhai Research Award, and IBM Real-Time Innovation Faculty Award. He is a fellow of the Indian National Academy of Engineering, an associate editor of the IEEE Transactions on Computers, and a subject area editor of the Journal of Parallel and Distributed Computing. He is a senior member of the IEEE.