Learning-TCP: A Novel Learning Automata Based Reliable Transport Protocol for Ad hoc Wireless Networks

B. Venkata Ramana, B. S. Manoj, and C. Siva Ram Murthy
Department of Computer Science and Engineering
Indian Institute of Technology, Madras, India
{vramana,bsmanoj}@cs.iitm.ernet.in and murthy@iitm.ac.in

Abstract— **The use of traditional TCP, in its present form, for reliable transport over Ad hoc Wireless Networks (AWNs) leads to significant degradation in the network performance, in terms of reduction in average network throughput and increase in packet losses. This is primarily due to the congestion window updation and congestion control mechanisms employed by TCP. TCP follows a deterministic approach for updating the size of the congestion window, which is less suitable for loss-prone AWNs as it leads to very frequent occurrences of congestion in the network. Another reason is, TCP invokes the congestion control mechanism for both congestion and wireless losses, as it cannot distinguish between them. Hence, in order to use TCP in AWNs, an efficient mechanism must be provided with TCP for updating the size of the congestion window based on the network conditions and distinguishing the congestion losses from wireless losses.**

In order to address these problems, we propose Learning-TCP, a novel learning automata based reliable transport protocol for AWNs, which efficiently adjusts the size of the congestion window and thus reduces the packet losses. The key idea behind Learning-TCP is that, it dynamically adapts to the changing network conditions by observing the occurrence of events, such as arrival of acknowledgment (ACK) and duplicate ACK (DUPACK) packets and appropriately updates the congestion window size. In addition to this, we use a deterministic approach for packet loss discrimination, in order to take the appropriate action for each type of loss. Learning-TCP, unlike other existing proposals for reliable transport over AWNs, does not require any explicit feedback, such as congestion, link failure, and available bandwidth notifications, from the network. We provide extensive simulation studies of Learning-TCP under varying network conditions, that show increased throughput and reduced packet loss compared to that of the traditional TCP.

I. INTRODUCTION

Ad hoc Wireless Networks (AWNs) are formed dynamically by a collection of mobile wireless nodes in the absence of any fixed infrastructure. 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. The AWNs 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 on 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. AWNs 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 [1]. In our paper, we focus on the issues related to the reliable and adaptive data transport over AWNs. Providing reliable data transfer over AWNs is not a trivial task, since they are constrained by the mobility of nodes and the resources. There are several proposals that suggest either a new transport protocol or enhancements to the traditional Transmission Control Protocol (TCP) to work efficiently in AWNs.

TCP [2] is a reliable, end-to-end, connection-oriented transport layer protocol that provides a byte-stream based service. The major responsibilities of TCP include congestion control, flow control, and in-order delivery and reliable transportation of packets. TCP uses a window, called congestion window to handle the congestion situations in the network. TCP regulates its packet transmission by expanding and shrinking the congestion window. TCP operates in two phases *slow start* and *congestion avoidance*. In *slow start* phase, TCP increases the congestion window by one MSS (Maximum Segment Size is the maximum possible size for a TCP segment) on receiving an acknowledgment (ACK) packet which indicates a successful reception of a data packet by the receiver. Being in slow start phase, TCP doubles the congestion window for every Round Trip Time (RTT) until the congestion window size reaches a threshold, called slow-start threshold. Once the congestion window exceeds the slow-start threshold, TCP enters into *congestion avoidance* phase, in which for each arrival of an ACK, TCP increases the congestion window by a fraction of MSS.

The receiver sends a duplicate acknowledgment (DUPACK) packet for every out-of-order packet it receives. On receptions of three consecutive DUPACKs, interpreting it as a packet loss due to congestion, the TCP invokes the congestion control mechanism, which halves the current congestion window and retransmits the lost packet. When TCP does not receive an ACK before the retransmission timeout (RTO) period, TCP updates the slow-start threshold to half of the current congestion window, resets the congestion window to one MSS, and retransmits the packet for which timeout has occurred.

TCP was designed to run efficiently on wire-line networks. Using TCP in its present form for the AWNs, degrades the performance of AWNs, in terms of increase in the packet loss and reduction in the network throughput. The reasons are as follows. TCP uses a deterministic congestion window updating mechanism that increases congestion window on receipt of an ACK packet and decreases the congestion window on receiving three successive DUPACK packets or upon the occurrence of an RTO. This deterministic approach may often lead to the occurrence of congestion situations in the network, which results in a high packet loss. The high packet loss affects AWNs severely as they are highly constrained by the bandwidth and the battery power.

Another reason is the TCP's inability to distinguish congestion losses from wireless losses, as a result of which, TCP invokes the congestion control mechanism for both types of losses. The wireless losses occur mostly due to path breaks and co-channel interferences. Moreover, an intermediate node generates a $false$ route error message, even during congestion, as it is unable to distinguish congestion from link breaks. On receiving a route error message, the network layer in a node deletes the corresponding routing entries from its route cache and on receiving packets from the higher layers, it re-establishes the path to the destination. This may result in an increase in both the network traffic and the delay in routing the TCP segments by the sender's network layer, which results in the occurrence of successive RTOs in TCP that ultimately leads to the connection's termination. Hence, in order to use TCP in AWNs, an efficient mechanism must be provided with TCP for updating the congestion window, based on the network conditions and distinguishing the congestion losses from wireless losses in order to take appropriate actions for each of these types of losses.

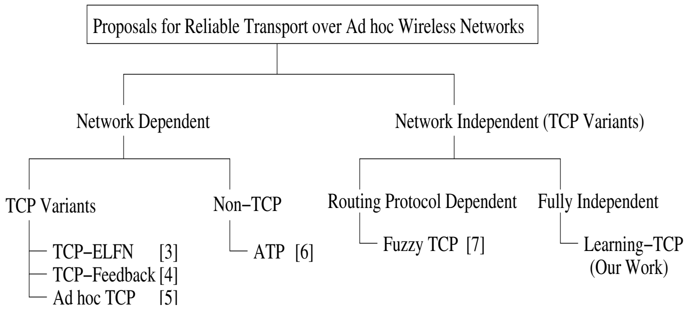


Fig. 1. Classification of the existing proposals.

Several proposals ([3]-[7]) have been made in order to address these problems. These proposals use different approaches for providing reliable transport over AWNs, and can be classified broadly into two categories – network dependent and network independent approaches. Figure 1 gives a detailed classification of these proposals. The proposals [3], [4], [5], and [6] are network dependent as they rely on explicit feedback information from the network, such as congestion, link failure, and available bandwidth notifications, in order to work efficiently. However, the network dependent proposals may perform poorly when the feedback information from network is unavailable or unreliable, which is very common in the AWNs. Hence the network independent approach is gaining research focus. The proposal [7], which belongs to this category, does not require any feedback from the network. Instead, it performs the loss classification and takes appropriate actions, by maintaining the history and observing various parameters, such as mean and variance of the round-trip times and hop-length of the TCP connections.

In [7], the authors propose a fuzzy engine for distinguishing congestion losses from channel error losses, over AWNs. The fuzzy engine monitors the rate of change in RTT and the number of hops (obtained from the routing protocol) in the TCP session and detects congestion when RTT increases n times successively, in which every increment is larger than a given α (where n and α are predefined). The channel error losses are detected when the mean RTT is small and variance is high. The path break errors are determined in a more deterministic way - on the occurrence of an RTO.

In this paper, we propose a novel idea that uses learning automata [8], in order to learn the network behavior and accordingly update the congestion window, by observing the inter arrival times of the ACK and DUPACK packets. Our work, Learning-TCP, is motivated by the advantages of learning automata over that of conventional mechanisms, such as machine learning and fuzzy logic techniques. Learning automata based solutions learn the network state better and faster and do not require the modeling of the network. Unlike other techniques, learning automata requires simple algorithms for updating probabilities, and the amount of information that needs to be maintained and the number of computations to be done, are significantly low. Thus, our learning automata based approach to the problem scales well with the increasing number of flows generated by a node.

The rest of the paper is organized as follows. We briefly give an introduction to Learning Automata in Section II. In Section III, we provide the design and functional details of Learning-TCP. In Section IV, we provide a detailed discussion about Learning-TCP. We present the simulation results and analysis of the results in Section V. Section VI discusses about the future work. In Section VII, we summarize our work. The convergence of our learning scheme can be found in [9].

II. Learning Automata - An Introduction

In a learning automata system, a finite number of actions can be performed in a random environment. When a specific action is performed, the environment provides either a favorable or an unfavorable response. 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. At any stage, the choice of action could be either deterministic or stochastic. In the latter case, probabilities are maintained for each possible action to be taken which are updated with the reception of each response from the environment. Figure 2 shows the interactions between the automaton and the environment.

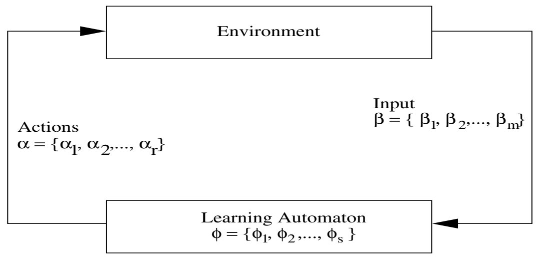


Fig. 2. Learning automaton operation in stochastic environment.

A learning automata can be formally and precisely 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 the 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* of an automaton at the 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 a and b are real numbers.
4) *Transition function* $F(.,.)$ determines the state at the instant $(n+1)$ in terms of the state and input at the 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 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 algorithm* or the *reinforcement scheme*. In this paper, we use the terms updating algorithm and updating function 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 updating function can be either linear or non-linear. Determining the updating algorithm for a variable-structure stochastic automaton makes for a very important design choice.

Based on the nature of feedback or reinforcement obtained from the environment, several models for learning automata have been developed. When the feedback is a binary-valued response, the model is called the P-model. Some well known linear updating functions for P-model include the Linear Reward-Penalty (L_{R-P}) scheme, the Linear Reward-Inaction (L_{R-I}) scheme, and the Linear Reward-ϵ-Penalty ($L_{R-\epsilon P}$) scheme [8]. However, in practice, it may be important to receive finer distinctions in the feedback, 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. Q- and S-models of learning automata have been devised to satisfy this need, where the response can take multiple values arising from a predefined performance index. In the Q-model, the feedback is a finite number of values in the unit interval [0,1]. If the performance index is continuous, the values could be obtained from quantization of this index and then normalizing them to the range [0,1]. This number can also vary from action to action and is denoted by m_i for action α_i ($i = 1, 2, \ldots, r$). In the S-model, the environment responses can take continuous real values in the interval [0,1], which can be obtained by normalization of the responses arising from the performance index.

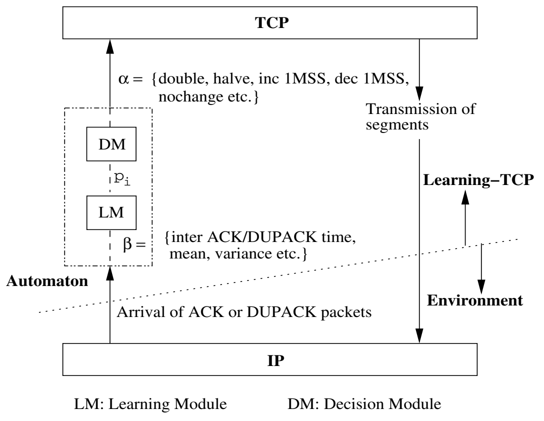
Let the response $\overline{\beta}^i$ to action α_i from the environment be in the range $[a, b]$, for all values of i. The normalized response β^i always lies in the range [0,1] and can be calculated as follows:
$$\beta^i = \frac{\overline{\beta}^i - a}{b - a}$$

Similar to the P-model, updating algorithms have been proposed for the Q- and S-models. These are denoted using the prefix S (e.g., SL_{R-I}, SL_{R-P}). For a detailed description of learning automata, readers are referred to [8].

III. Design and Implementation of Learning-TCP

Learning-TCP follows the learning automata based approach to adapt to the network conditions and update the size of the congestion window, accordingly. Though there exist several proposals for reliable and efficient transport over the AWNs, there is no proposal available that uses the probabilistic approach for updating the congestion window size and thus reducing the congestion losses. Moreover, Learning-TCP does not require any explicit feedbacks from the network. In this section, we describe the design and implementation of Learning-TCP.

Figure 3 shows the relationship between the various components of Learning-TCP. Learning-TCP consists of TCP (Reno) and a learning automaton which implements learning and decision algorithms; it is compatible with TCP as it does not



TCP
α = {double, halve, inc 1MSS, dec 1MSS, nochange etc.}
Transmission of segments
DM
p_i
LM
Learning–TCP
β = {inter ACK/DUPACK time, mean, variance etc.}
Automaton
Environment
Arrival of ACK or DUPACK packets
IP
LM: Learning Module
DM: Decision Module

Fig. 3. Relationship between the learning automaton and TCP.

modify the semantics and header format of TCP, but changes the congestion window updating mechanism. For this purpose, a learning automaton is placed at each node that works along with TCP.

As shown in the figure, the Learning Module (LM) implements the learning algorithm. Instead of explicit feedback information from the network, the LM obtains the network conditions in terms of the inter arrival times of the TCP packets (ACK/DUPACK) that are received from IP. Henceforth, in this paper, we use the term *network response* to represent the inter arrival times of the TCP packets. This network response is transformed into the performance index, $\beta(n)$ that ranges between [0,1]. The performance index, $\beta(n)$ indicates how far is the reward (in terms of delays) obtained from the network, from the maximum (best) possible reward. $\beta(n) = 0$ indicates the highest reward that reflects the best network conditions. The $\beta(n)$ that is obtained from the network, is then used in the continuous-response (S-model) reinforcement scheme with linear reward-penalty, called SL_{R-P}, to update the probability values $p_1, p_2, \ldots, p_5$ that correspond to the actions $\alpha_1, \alpha_2, \ldots, \alpha_5$. These action probabilities are used by the Decision Module (DM), which implements the decision algorithm, in deciding the action to be taken. The decisions taken by the DM are probabilistic rather than deterministic.

The following subsections discuss, in detail, the states and the corresponding actions of the learning automaton and the functionality of the learning and decision modules.

A. States $\phi(n)$ and Actions $\alpha(n)$

At any time instance, the automaton will be in any one of the five states, Multiplicative increase (Mult_Inc), Multiplicative decrease (Mult_Dec), Inaction, Additive increase (Addt_Inc), or Additive decrease (Addt_Dec). The actions $(\alpha_1, \alpha_2, \ldots, \alpha_5)$ corresponding to each of the above-mentioned automaton states $(\phi_1, \phi_2, \ldots, \phi_5)$ are doubling, halving, no-change, linear-increment, and linear-decrement, respectively. Table I gives the details of how the congestion window is updated for every action, and the expected size of the congestion window by the end of the RTT.

The actions, doubling and linear-increment, increase the congestion window by one MSS and fraction of MSS (*frac*), respectively. This *frac* can be defined as $\frac{MSS \times MSS}{cwnd}$. These two actions correspond to the TCP congestion window updating mechanism in slow start and congestion avoidance phases, respectively. However, in Learning-TCP, decreasing the congestion window is different from that of TCP. At any time instance, the linear-decrement action decreases the congestion window by a *frac* and the halving action results in the decrease of the congestion window by one MSS. If the same action of halving is selected and performed for the successive events, till the end of the current RTT, the congestion window is reduced to a maximum extent, which is half the current congestion window size. Though there is a higher probability to select the same action for the successive events in an RTT, the automaton might select different actions depending on the action probabilities.

B. Learning Module (LM)

The LM is responsible for learning the network conditions by observing the network response. For this purpose, the LM maintains the inter arrival times between any two TCP acknowledgment packets, called *inter_arrival_time*. It also maintains the average (*mean*) and standard deviation (*sdev*) of the inter arrival times, which are computed at the end of every RTT. It is often possible that a packet with a very high inter arrival time causes a large (or sudden) increase in the *mean*. These fluctuations may be due to the poor channel conditions in the reverse path (from receiver node to sender node), in which case Learning-TCP may wrongly interpret these conditions as congestion on the forward path. In order to prevent the fluctuations, the LM verifies and eliminates the *inter_arrival_time* values that are not in the range [*mean-sdev, mean+sdev*], when the *sdev* is high (i.e., $sdev > \frac{mean}{2}$). DUPACK packets are handled in the similar way as the ACK packets except in computing the values for *mean* and *sdev*. In order to ensure that the *mean* and *sdev* represent the state of the congestion-free network, these parameters are computed only for the RTTs which did not receive any DUPACK packets.

On receipt of a packet, the LM, as shown in Equation 1, captures the congestion state of the network into a parameter *time_ratio*, using the current inter arrival time and the *mean* which computed over the inter arrival times of the ACK packets arrived in the previous RTT. The *time_ratio* has an upper bound of 1 and a lower bound of δ which takes the value -2. In our simulations, we assume that the maximum time interval between any two consecutive events to be three times the *mean*. Hence, the only possible value for δ is -2. Any packet that arrives later than three times that of the current mean, is considered as a very-late packet and the *time_ratio* is taken as -2. However, the current *inter_arrival_time* is stored and is used for calculating the *mean* value at the end of the RTT. The *time_ratio* obtains a value close to one when the current *inter_arrival_time* is negligible compared to the *mean*, in which case, the packet is called a very-early packet. As shown in Equation 2, the state of the network obtained into

TABLE I

COMPUTATION OF CONGESTION WINDOW (CWND) FOR EACH ACTION

State (ϕ)	**Action** (α)	**Immediate change in cwnd**	**Expected change in cwnd per RTT**
Mult_Inc	Doubling	cwnd = cwnd + MSS	cwnd = cwnd * 2
Mult_Dec	Halving	cwnd = cwnd - MSS	cwnd = cwnd / 2
Inaction	No-change	cwnd = cwnd + 0	cwnd
Addt_Inc	Linear increment	cwnd = cwnd + frac	cwnd = cwnd + MSS
Addt_Dec	Linear decrement	cwnd = cwnd - frac	cwnd = cwnd - MSS

TABLE II

EQUATIONS FOR COMPUTING β VALUE

State (ϕ) at time step n	**On the arrival of**	
	ACK packet	DUPACK packet
Mult_Inc	β = 1 - γ	β = $\frac{2-\gamma}{2}$
Mult_Dec	β = γ	β = $\frac{\gamma}{2}$
Inaction	β = $\lvert\frac{0.5-\gamma}{0.5}\rvert$	β = $\lvert\frac{0.5-\gamma}{0.5}\rvert$
Addt_Inc	β = $\lvert\frac{0.75-\gamma}{0.75}\rvert$	β = $\lvert\frac{0.75-\gamma}{0.75}\rvert$
Addt_Dec	β = $\lvert\frac{0.25-\gamma}{0.75}\rvert$	β = $\lvert\frac{0.25-\gamma}{0.75}\rvert$

time_ratio is normalized and stored in the parameter γ.

$$time\_ratio = \frac{mean - current\ inter\_arrival\_time}{mean} \quad (1)$$

$$\gamma = \frac{time\_ratio - \delta}{1 - \delta} \quad (2)$$

After obtaining the normalized input value γ, depending on the state of the automaton and type of the packet arrived, the LM computes the performance index $\beta(n)$ from the corresponding equation in Table II. Note that, a favorable network response results in a value close to zero for $\beta(n)$. We present the detailed discussion on the intuition behind the equations mentioned in Table II and their influence on learning, in Section IV.

Due to the continuous nature of the performance index $\beta(n)$, we use the S-model of learning automata, where the response from the environment can take continuous values in the interval [0,1]. Further, we use the reward-penalty scheme, L_{R-P} [10] and [11], as discussed in Section III-C, largely because of its simplicity. Once $\beta(n)$ is obtained from the environment, using the S-model reinforcement scheme with linear reward-penalty SL_{R-P} of learning automata we update the probabilities of all the actions.

C. The SL_{R-P} Scheme

The basic idea behind a reward-penalty scheme is as follows. When a positive response is obtained for an action, its probability is increased and the probabilities of all other actions are decreased. If a negative feedback is received for an action, the probability of that action is decreased and that of others is increased. In this section, we first describe the L_{R-P} scheme [12] to elucidate the reward-penalty concept. We then describe the more complex S-model version of linear reward-penalty scheme, SL_{R-P}.

An L_{R-P} scheme can be a two action or a multi-action scheme. Mathematically, a two action L_{R-P} scheme can be expressed as follows:

- When a positive feedback is obtained for action 1,
  $p_1(n+1) = p_1(n) + a(1 - p_1(n))$
  $p_2(n+1) = (1 - a)p_2(n)$
- When a negative feedback is obtained for action 1,
  $p_1(n+1) = (1 - b)p_1(n)$
  $p_2(n+1) = p_2(n) + b(1 - p_2(n))$

and similarly for action 2, where $0 < a < 1, 0 \leq b < 1$.

For a multi-action system with r actions, the updating algorithm can be written as follows:

- When a positive feedback is obtained for action i,
  $p_i(n+1) = p_i(n) + a(1 - p_i(n))$
  $p_j(n+1) = (1 - a)p_j(n), j \neq i$
- When a negative feedback is obtained for action i,
  $p_i(n+1) = (1 - b)p_i(n)$
  $p_j(n+1) = \frac{b}{r-1} + (1 - b)p_j(n)), j \neq i$

As mentioned above, due to the continuous nature of the performance index $\beta(n)$ and the possibility of multiple actions, we use the S-model reinforcement scheme with multi-action linear reward-penalty scheme L_{R-P}, known as SL_{R-P}. In the SL_{R-P} scheme, responses may be partly favorable and partly unfavorable to the action. The extent of favorability is specified as a value between 0 and 1. The equations for SL_{R-P} reinforcement scheme are given mathematically in Equation 3.

$$\begin{aligned} p_i(n+1) &= p_i(n) - (1 - \beta(n))g_i(p(n)) \\ &\qquad + \beta(n)h_i(p(n))\ if\ \alpha(n) \neq \alpha_i \\ p_i(n+1) &= p_i(n) + (1 - \beta(n))\sum_{j\neq i}^{r} g_j(p(n)) \\ &\qquad - \beta(n)\sum_{j\neq i}^{r} h_j(p(n)), \quad if \ \ \alpha(n) = \alpha_i \end{aligned} \quad (3)$$

The action probabilities at time step $n+1$ are computed from action probabilities at time step n and the current input values of $\beta(n)$. The α_i, in the equation, refers to the action selected at time step n. The functions $g_j(.)$ and $h_j(.)$ correspond to reward and penalty functions, respectively, of the multi-action L_{R-P} scheme and are computed based on Equation 4.

$$\begin{aligned} g_j(p) &= ap_j(n) \\ h_j(n) &= \frac{a}{r-1} - ap_j(n) \end{aligned} \quad (4)$$

The terms *a* and *r* in these equations correspond to the *learning factor* and *number of actions*, respectively.

D. Decision Module (DM)

TCP has a deterministic decision mechanism for increasing or decreasing the congestion window. On receiving an ACK, TCP increases the congestion window without considering the delays in receiving the ACKs. However, increasing congestion window when the ACKs are arriving with high delays leads to congestion on the network and hence, leads to a loss of several TCP segments on the network. However, our decision module in the learning automaton stochastically selects the action to be taken based on the probabilities computed by the LM. This is a salient feature which helps the automaton to perform various actions efficiently and accordingly adapt to changing network conditions. Since the preferred actions are assigned higher probability values by the LM, there is a high probability for these actions to be selected by the decision module, at the same time the actions with lower probabilities are not completely ignored.

E. Loss Discrimination

The learning automaton provides an efficient way of updating the congestion window. Though we do not use the learning automaton for explicit loss classification, the efficient congestion window updation results in fewer congestion losses. The information about the wireless losses due to link breaks is obtained by using a deterministic approach. On occurrence of an RTO, the most recent *mean* and *sdev* are compared by the automaton. If *sdev* is significantly low compared to the *mean* (i.e., $sdev < \frac{mean}{2}$), it interprets this situation as a link break and freezes the TCP connection, and pushes TCP into FREEZE state. Otherwise, it allows TCP to invoke the usual congestion control algorithm. In FREEZE state, TCP periodically sends a probe packet. It resumes the normal state and reset the congestion window to one MSS, when an ACK for a probe packet is received.

IV. Discussion about Learning-TCP

In this section, we discuss in detail, about the intuition behind the selection of various actions and states for learning automaton, and the effect of the equations given in Table II which we use for mapping the network response onto the $\beta(n)$.

A. Selection of Actions (States)

The states that the automaton could be present are described in Table I. Each state of the automaton represents a distinct action that is performed by the automaton when it is in that state. As the network conditions vary continuously, two actions, increase and decrease of the congestion window by one MSS, may not be sufficient. In order to efficiently handle the variations in the network conditions, we have considered five actions for the Learning-TCP. For the purpose of this discussion, we classify the packets (ACK/DUPACK) that arrive at the TCP sender into four classes. Table III, gives the classification of the packets, the network conditions that result in each type of packet and the preferred actions on receiving such packets.

When the network is very lightly loaded, the packets sent by the TCP receiver arrive at the TCP sender at higher rate (i.e., with negligible *inter_arrival_time*) than when the load in the network is high. These packets are called very-early packets. The very-early packets, which cause a value close to one for γ computed from Equation 2, give an indication that the network is currently very lightly loaded and the congestion window should be increased at a high rate. The doubling action that increases the congestion window by one MSS, is the favorable action in this case.

When the automaton is in Mult_Inc state and it receives an ACK packet, it uses the equation $\beta(n) = 1 - \gamma$ to map the network response to the performance index, $\beta(n)$. Since in S-model reinforcement scheme, a value close to zero for $\beta(n)$ is considered as favorable response from the network, the above equation evaluates $\beta(n)$ close to zero for a very-early ACK packet, which results in a further increase in the probability value of the doubling action and a proportional decrease in the probabilities of the other actions. The rate of increase in the probability of doubling action decreases for increasing $\beta(n)$ values. Maximum reduction in the probability of doubling takes place when the received ACK is a very-late packet.

We define separate equations for $\beta(n)$ for the cases of ACK and DUPACK arrivals. The automaton uses the equation $\beta(n) = |\frac{2-\gamma}{2}|$ when it is in Mult_Inc state and receives a DUPACK packet. This equation shifts the network response to the interval [0.5,1] to indicate a relatively unfavorable response compared to that of an ACK arrival.

Similarly, the automaton should favor the actions that decrease congestion window when a severe congestion situation is anticipated. During the congestion situation, the packets sent by the receiver arrive with high *inter_arrival_time*. We call these packets as very-late packets. When the automaton is in Mult_Dec state, it takes the value for $\beta(n)$ as γ and $\frac{\gamma}{2}$, on the arrival of an ACK and DUPACK packet, respectively. A very-late ACK packet results in a value close to zero for $\beta(n)$, which further increases the probability of the halving action and thus favors the same action. Compared to that of a late packet, a very-early or early packet results in the maximum reduction in the probability of halving action that makes the action as unfavorable for the next time step. However, a very-early or early DUPACK packet does not penalize the halving action as much as that of very-early or early ACK packet. And a very-late or late DUPACK packet is rewarded higher compared to that of very-late or late ACK packet.

The doubling and halving actions, update the congestion window at a higher rate compared to the linear-increment and linear-decrement actions. These actions are preferred to increase/decrease the congestion window aggressively. However, these actions may result in an unfavorable response (leading to congestion or under utilization of the network) when the network is lightly or moderately loaded. For these cases, we propose linear-increment and linear-decrement actions which update the congestion window by a fraction ($\frac{MSS \times MSS}{cwnd}$) for each occurrence of a favorable event.

TABLE III

CLASSIFICATION OF PACKETS

Gamma(γ)	Packet Type	Indication of the Network State	Action to be taken on cwnd
0 – 0.25	very-late packet	Heavily loaded, severe congestion may occur immediately	Deflate the *cwnd* at a higher rate
0.25 – 0.5	late packet	Moderately loaded, congestion may arise in near future	Deflate the *cwnd* at a lower rate
0.5 – 0.75	early packet	Lightly loaded, congestion may not occur	Inflate the *cwnd* at a lower rate
0.75 – 1	very-early packet	Very lightly loaded, handle more load	Inflate the *cwnd* at a higher rate

When the automaton is in Addt_Inc state, on arrival of a packet, it uses the equation $\beta(n) = |\frac{0.75-\gamma}{0.75}|$ to obtain the performance index. This equation evaluates $\beta(n)$ close to zero when the current packet is an early packet, which further increases the favorability for the linear-increment action. The very-late and late packets result in a higher value for $\beta(n)$, which penalizes the linear-increment action and rewards the other actions. The very-early packets result in a small reward for doubling action. Similarly, in Addt_Dec state, the automaton uses the equation $\beta(n) = |\frac{0.25-\gamma}{0.25}|$ to reward the linear-decrement action when late packets arrive, which otherwise gets penalized for other types of packets.

Finally, when the automaton is in Inaction state, it rewards the no-change action and penalizes the other actions if the current *inter_arrival_time* is close to *mean*, in which case the equation $\beta(n) = |\frac{0.50-\gamma}{0.50}|$ evaluates to a value close to zero 0.

B. Handling of DUPACKs

In Learning-TCP, we propose a few changes to the duplicate acknowledgment (DUPACK) handling mechanism of TCP-Reno. As in TCP-Reno, a segment is retransmitted once the sender receives three consecutive DUPACKs for that segment. However, the congestion window is not halved here, instead it is updated by following the congestion window updating mechanism described in the previous section.

In general, the very high inter arrival times in the preceding RTT may result in the DUPACK packets in the next RTT. Hence, it is incorrect to use the current *mean*, which was computed using the inter arrival times in the previous RTT, for obtaining $\beta(n)$ on receiving DUPACK packets. For this purpose, we maintain the *mean* values of the two most recent RTTs, minimum of which is called *expected-mean*. This value will be used to obtain $\beta(n)$ values from the network response, for all the packets (DUPACK/ACK) received until the next value for *mean* is calculated.

V. SIMULATION RESULTS

We used GloMoSim to simulate our protocol. FTP is used as the application layer protocol to generate TCP traffic. Terrain area used for the simulations is 1000m$\times$1000m. The number of nodes in network is 100. Transmission range of a node is taken to be 195m. Mobility model used is the random waypoint model, with a pause time of zero seconds. The two-ray pathloss is used as a pathloss model. The MAC and routing protocols used in our simulations are IEEE 802.11 and AODV, respectively. The channel capacity is assumed to be 2 Mbps.

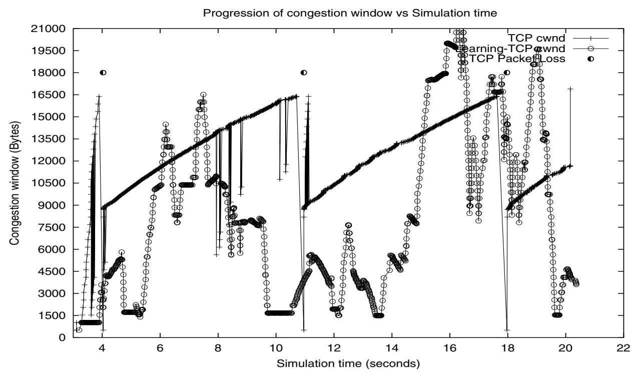


Fig. 4. Progression of cwnd of TCP and Learning-TCP without mobility.

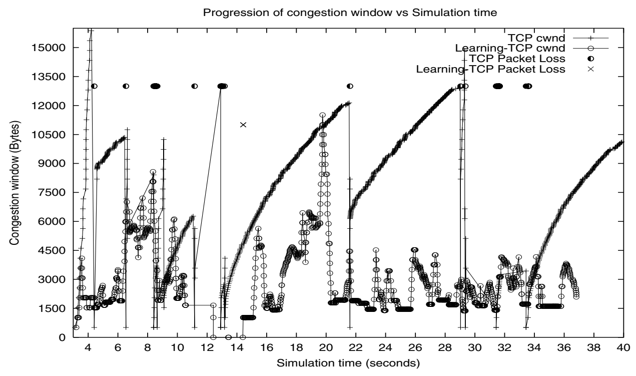


Fig. 5. Progression of cwnd of TCP and Learning-TCP with mobility.

Each point in the graphs is obtained out of results from several runs, each with a different seed. All the results in this section have been obtained at 95% confidence level.

We tested Learning-TCP for a number of FTP flows varying from one to fifteen flows, for the mobility from 0m/s to 16m/s, each with a size of 1.1 MB. We have considered different mobilities to study the performance of Learning-TCP in different scenarios, such as on-the-fly conferences, electronic class rooms, and trade shows, where the users will be moving with a low mobility and military and emergency rescue operations, where the users will be moving with a high mobility. To the best of our knowledge there exists no proposal in literature that is independent of the explicit network feedback and uses the probabilistic approach for updating the size of the congestion window and controlling the congestion on the network. We compare the performance of Learning-TCP with that of TCP (Reno). The metrics used for comparison are

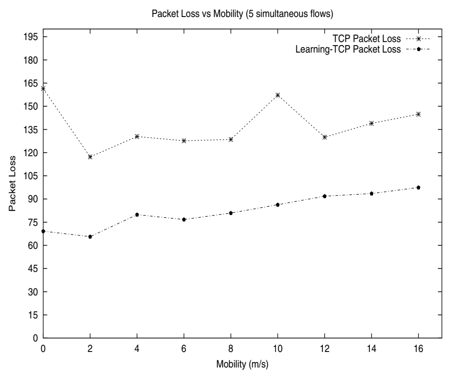


Fig. 7. Packet Loss vs Mobility for 5 fbws.

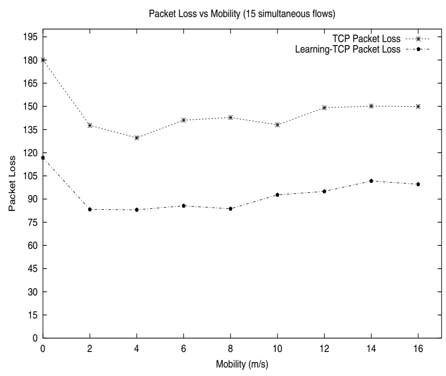


Fig. 8. Packet Loss vs Mobility for 15 fbws.

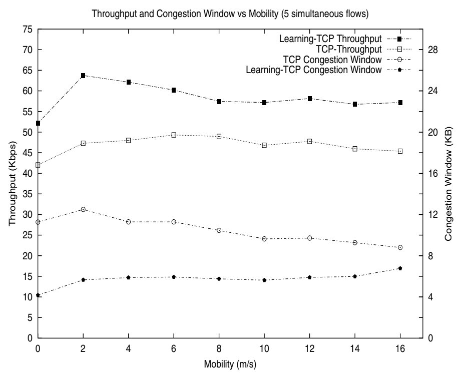


Fig. 9. Throughput and Congestion Window vs Mobility for 5 fbws.

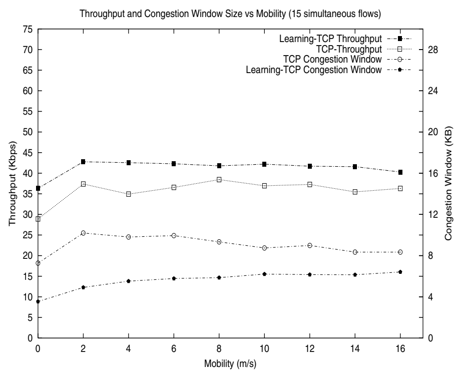


Fig. 10. Throughput and Congestion Window vs Mobility for 15 fbws.

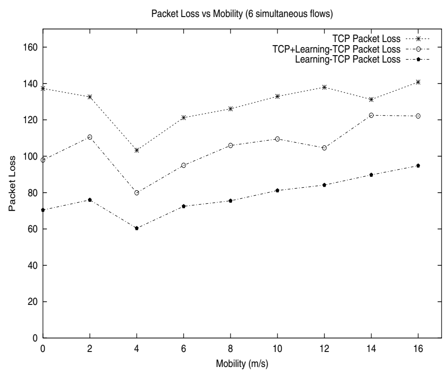


Fig. 11. Packet Loss vs Mobility for 6 fbws.

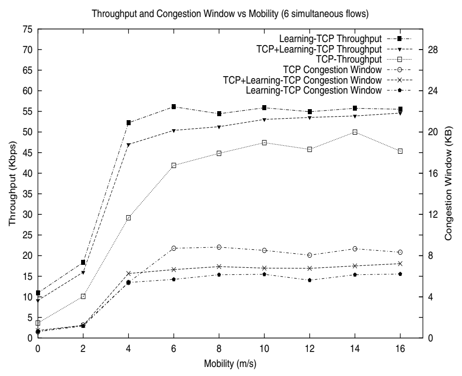


Fig. 12. Throughput and Congestion Window vs Mobility for 6 fbws.

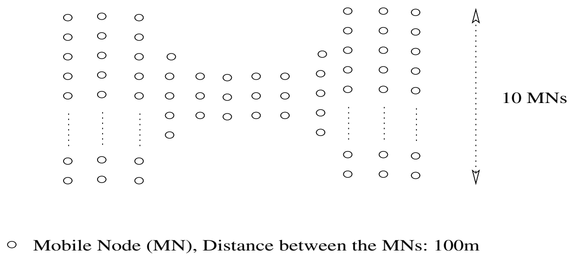


Fig. 6. Placement of Nodes.

average packet loss, *overall network throughput*, and *average congestion window*.

Figures 4 and 5 show the progression of the congestion windows and packet losses of a single TCP and Learning-TCP flow, for two different scenarios, with and without mobility, respectively. In Figure 4, both TCP and Learning-TCP are transmitting 1MB of data and completing transmission almost at the same time (at 21 sec). We notice that TCP is rapidly increasing the congestion window and falling suddenly to the minimum congestion window size and there was loss of three packets. However, Learning-TCP is steadily increasing the congestion window and is able to retain it for longer duration compared to that of TCP. Moreover, it is able to achieve larger congestion window than that of TCP (at time 16 sec). From Figure 5, we can make several observations, firstly TCP is aggressively increasing the congestion window. However, it is experiencing high packet loss (88 packets) compared to that of Learning-TCP (3 packets). Secondly, though both are transmitting the same amount of data, Learning-TCP is able to complete its transmission faster (in 37 sec) compared to that of TCP (in 40 sec), even with the comparatively moderate size of the congestion window. We can infer from these two graphs that, Learning-TCP is able to obtain larger congestion window when the network is smooth, and in the presence of congestion in the network, Learning-TCP operates at lower congestion window size. In either case, Learning-TCP is performing superior to TCP, in terms of reduction in packet loss and increase in throughput, by efficiently updating the congestion window according to the state of the network.

Figures 7 and 8 present the packet losses of TCP and Learning-TCP for varying number of simultaneous flows, 5 and 15, respectively, at different mobility. We observe a significant and consistent reduction in the packet loss of Learning-TCP over that of TCP. Both TCP and Learning-TCP show an increase in the packet loss at higher mobilities and larger number of simultaneous flows. However, Learning-TCP shows considerably very low and consistent reduction in packet loss compared to that of TCP. The reduction in the packet loss

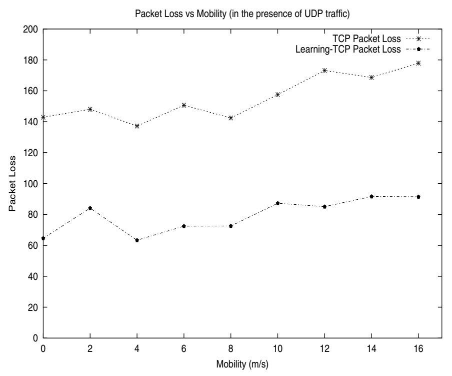


Fig. 13. Packet Loss vs Mobility.

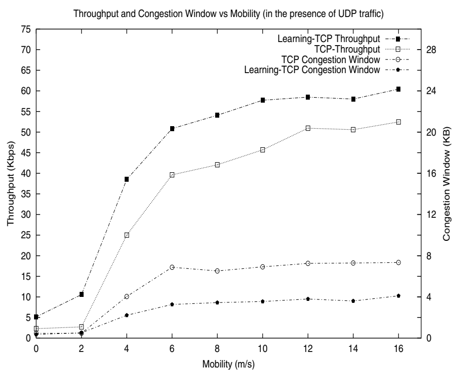


Fig. 14. Throughput and Congestion Window vs Mobility.

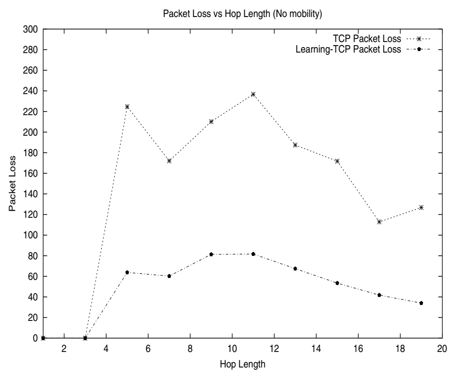


Fig. 15. Packet Loss vs Hop Length for one fbw.

ranges between 30% and 58% for 5 flows, and between 33% and 42% for 15 flows. The adaptive congestion window updation followed by the Learning-TCP, helps to achieve this better performance over TCP. When the packets arrive with high delays, there is a higher probability for choosing the actions that decrease the size of the congestion window by the automaton. This directly reduces the possibility of a congestion situation and minimizes the congestion losses, and in turn minimizes the retransmissions. Moreover, it helps the network to recover quickly from the congestion situation, since all the nodes which experience higher delays are most likely to perform similar (decrease) actions. Hence, the congestion recovery can be done faster with fewer congestion losses. As a result, the channel utilization and energy consumption at the nodes improve.

Figures 9 and 10 show corresponding throughput and congestion window for the same setup. In both 5 and 15 flows scenarios, we observe a significant and consistent improvement in the throughput of Learning-TCP over TCP and a higher congestion window for TCP across all the mobilities than for Learning-TCP. As explained earlier, the efficient congestion window updation mechanism inflates or deflates size of the congestion window based on the network conditions. When the automaton is receiving very-early packets, it is more likely that the automaton performs the doubling or linear-increment actions which increases the size of the congestion window. However, it is very unlikely to perform these actions when the automaton is receiving late or very-late packets, in which case the automaton prefers the linear-decrement or halving actions that decreases the size of congestion window. In this way, Learning-TCP learns and adapts to the network state from each event, updates the size of congestion window accordingly, and takes necessary actions faster compared to the TCP. However, TCP does not consider the delays in the arrival of the ACKs. It strictly increases the congestion window even while the ACK packets are received with high delays that may lead to severe congestion situations in the network, which further result in higher packet losses. The improvement in the throughput of Learning-TCP over that of TCP ranges between 17% and 35% for 5 flows, and between 9% and 26% for 15 flows. We observe similar trends in the peformance of Learning-TCP over TCP for 10 flows. The reduction in packet loss and improvement in throughput ranges between 31% and 43%, and 17% and 32%, respectively, compared to TCP.

Figure 6 shows the topology of the network we have used for the remaining simulation studies presented in this paper. Our intuition in selecting the dumbbell shape topology is that it is a simple and appropriate topology that introduces congestion losses when the flows are crossing the bottleneck link.

Figures 11 and 12 represent the performance of Learning-TCP in the presence of TCP traffic. We have taken TCP and Learning-TCP flows of three each and compared the performance of overall packet loss, size of congestion window, and throughput with that of six Learning-TCP flows and six TCP flows. All the flows are starting at the same time and four of them cross the bottleneck link. All the three cases (only TCP flows, only Learning-TCP flows, and TCP+Learning-TCP flows) show the similar trend of increase in the packet losses with increasing mobility, except for the static and 2m/s mobility. We observe a significant improvement in the performance of Learning-TCP over the other two cases. Most importantly, the amount of packet loss on the network when there are both TCP and Learning-TCP flows, is reduced almost by 50% compared to that of equal number of TCP flows. Similarly, the throughput increases drastically and is almost close to the case of when there are same number of Learning-TCP flows. This implies that the TCP and Learning-TCP coexist well and when the network has a mixture of TCP and Learning-TCP flows, the overall network throughput improves compared to an equal number of TCP flows. This is due to the adaptation of Learning-TCP nodes to the dynamically changing network conditions, which results in a fewer number of congestion situations and thus improves the availability of network resources for the TCP flows.

Figures 13 and 14 correspond to the results in the presence of UDP flows. Figure 13 shows the packet loss versus mobility and Figure 14 shows throughput and congestion window size versus mobility for both Learning-TCP and TCP. There are three UDP flows running simultaneously with the three TCP or Learning-TCP flows. Each UDP flow is generating packets of size 512 bytes with 100ms of packet inter departure time. Learning-TCP outperforms TCP by showing significant reduction in the packet loss and consistent improvement in the throughput with increasing mobility. The maximum improvement of Learning-TCP over TCP in throughput and reduction in packet loss are 54% and 55%, respectively, which confirms that the learning mechanism works efficiently even in the presence of non-TCP traffic.

In Figures 12 and 14, we observe a very low throughput at the static and 2m/s mobility and throughput sharply increases with increasing mobility. This is due to unavailability of the paths for the TCP/Learning-TCP flows that are crossing the bottleneck link because of the heavy traffic. Moreover, the $false$ route error messages cause an increase in the network traffic and a high delay in routing the TCP segments by the sender's network layer, which results in the occurrence of several retransmission timeouts in TCP that further causes TCP to terminate the connection.

Figure 15 provides performance of Learning-TCP over TCP in terms of packet loss with increasing hop lengths. We observe, both TCP and Learning-TCP show increasing packet loss up to the hop length of 11, from then the packet loss is decreasing. However, Learning-TCP shows a very high reduction in the packet loss compared to that of TCP. The reduction in packet loss ranges between 62% and 74%. Due to space constraints, we are unable to present the corresponding throughput versus hop length graph. However, for hop lengths from 5 to 15, Learning-TCP shows significant improvement in the throughput compared to TCP. It shows the maximum improvement of 58% for the hop length of 11 and marginal improvements for the hop lengths of 1, 2, 17, and 19.

In all the simulation studies, we observe a higher congestion window for TCP than for Learning-TCP. This is mainly due to the exponential increase in the congestion window size during the slow-start phase of TCP. Though the exponential increase results in a high congestion window, it leads to a high packet loss for TCP, which further results in a long loss recovery period. Hence, TCP shows a lower throughput even with the higher (about 40%) congestion window, compared to that of Learning-TCP.

VI. Future Work

In this paper, we presented a novel idea that uses the learning automata based approach to update the TCP congestion window based on the network conditions that leads to fewer number of congestion situations in loss-prone ad hoc wireless network. We have used the inter arrival times of the packets as a measure for obtaining network conditions and continuous-response reinforcement scheme with linear reward-penalty (SL_{R-P}) for updating the action probabilities. In our future work, we will check the possibility of other parameters such as current congestion window and variations in the round-trip times, in order to obtain the network conditions more accurately. Currently, the action probabilities are bounded by the values 0 and 1. When the probability of an action approaches the lower bound zero, the action may not be selected for a long period of time. Hence, in our future work, we shall try to enforce a lower bound on the probabilities of the actions, so that all the actions will have a fair chance to get selected.

VII. Conclusions

In this paper, we proposed a novel Learning-TCP for ad hoc wireless networks, which does not depend on explicit feedback from the network such as congestion, link failure, and available bandwidth notifications, but it adapts to the network conditions by observing the inter arrival times of ACK and DUPACK packets. We used learning automata because of its unique feature of learning the network state better and faster by maintaining considerably lesser amount of information and with negligible computational requirements. Learning-TCP was simulated using GloMoSim and the simulation results have clearly indicated an efficient congestion window updation that resulted in a few number of congestion situations. Learning-TCP has shown a reduction in the packet loss by about 30-74% and an increase in the throughput by about 9-58% compared to that of TCP. The lesser packet losses, in the case of Learning-TCP, results in a better utilization of the network and a reduction in the energy consumption of the nodes, which is vital for resource constrained ad hoc wireless networks. Moreover, Learning-TCP neither expects explicit feedback from the network nor performs extensive computations and, hence, it is scalable. In addition, Learning-TCP is compatible with the traditional TCP as it does not modify semantics and header format of TCP.

References

[1] I. Chlamtac, M. Conti, and J. J. N. Liu, "Mobile Ad Hoc Networking: Imperatives and Challenges," *Journal of Ad Hoc Networks*, vol. 1, no. 1, pp. 13-64, July 2003.

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

[3] G. Holland and N. Vaidya, "Analysis of TCP over Mobile Ad Hoc Networks," *in Proc. ACM MobiCom*, pp. 219-230, August 1999.

[4] K. Chandran, S. Raghunathan, S. Venkatesan, and R. Prakash, "A Feedback Based Scheme for Improving TCP Performance in Ad Hoc Wireless Networks," *IEEE Personal Communications Magazine*, vol. 8, no. 1, pp. 34-39, February 2001.

[5] J. Liu and S. Singh, "ATCP: TCP for Mobile Ad Hoc Networks," *IEEE Journal on Selected Area in Communications*, vol. 19, no. 7, pp. 1300-1315, July 2001.

[6] K. Sundaresan, V. Anantharaman, H. Hsieh, and R. Sivakumar, "ATP: A Reliable Transport Protocol for Ad-hoc Networks," *in Proc. ACM MobiHoc*, pp. 64-75, June 2003.

[7] R. Oliveira and T. Braun, "A Delay-based Approach Using Fuzzy Logic to Improve TCP Error Detection in Ad Hoc Networks," *in Proc. IEEE WCNC*, vol. 3, pp. 21-25, March 2004.

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

[9] B. Venkata Ramana, B. S. Manoj, and C. Siva Ram Murthy, "Learning-TCP: A Novel Learning Automata Based Reliable Transport Protocol for Ad hoc Wireless Networks," IIT Madras, Tech. Rep., January 2005.

[10] K. S. Fu and R. W. Mclaren, "An Application of Stochastic Automata to the Synthesis of Learning Systems," *Report TR-EE-65-17*, Purdue University, Lafayette, 1965.

[11] K. S. Fu and G. J. McMurtry, "A Study of Stochastic Automata as a Model for Learning and Adaptive Controllers," *IEEE Trans. Automatic Control*, AC-11, pp. 379-387, 1966.

[12] R. R. Bush and F. Mosteller, *Stochastic Models for Learning,* John Wiley & Sons, New York, 1958.