Wireless Netw
DOI 10.1007/s11276-009-0228-4

A novel learning based solution for efficient data transport in heterogeneous wireless networks

Venkataramana Badarla · C. Siva Ram Murthy

Abstract There has been a spectacular growth in the use of wireless networks in recent times and consequently, adapting TCP to the wireless networks is a hot topic of current research. However, most of the existing works proposed for this problem have been designed for specific wireless networks, or they necessitate changes at either the receiver or the intermediate nodes, or at both, because of which their deployment becomes difficult. In this work, we propose a TCP variant which works over both multi-hop ad hoc wireless networks as well as single-hop (last-hop) wireless networks, like Wireless LANs, cellular, and satellite networks. We use a learning based method to dynamically change the congestion window size according to the network conditions. Our protocol does not rely on any explicit feedback from the network and requires only sender-side modifications. Through extensive simulations we show that our protocol achieves the desired goals of performance improvement in goodput, reduction in packet loss, and fairness to the competing flows. To the best of our knowledge, this is the first unified solution for both single-hop and multi-hop wireless networks.

Keywords TCP · Congestion control · Satellite network · Cellular network · Wireless LAN · Ad hoc network · Heterogeneous wireless networks · Learning automata · Performance evaluation

V. Badarla (✉)
Hamilton Institute, National University of Ireland Maynooth, Maynooth, Republic of Ireland
e-mail: badarla.venkataramana@nuim.ie; b.v.ramana@gmail.com

C. Siva Ram Murthy
Department of Computer Science and Engineering, Indian Institute of Technology Madras, Chennai 600 036, India
e-mail: murthy@iitm.ac.in

1 Introduction

The combination of Transport Control Protocol (TCP) and Internet Protocol dominates today's communication in various networks. The purpose of TCP is to provide a byte-streaming in-order connection-oriented reliable service to the application layer. Another important function of TCP is congestion control. It tries to control the flow of packets, to prevent an overflow of buffers at the routers, by maintaining a congestion window (cwnd). The cwnd is an upper bound on the maximum number of unacknowledged packets for a TCP flow in the network. TCP adjusts its cwnd in a deterministic fashion according to the network events. For more details about TCP, please refer [1].

1.1 A brief introduction to various wireless networks

Satellite links are characterized by a high latency and a high bandwidth, thus having a high bandwidth-delay product (BDP). These links typically have link asymmetry, with the downlink bandwidth usually being much higher than the uplink bandwidth. There is a huge variation in the bandwidth provided by the satellite links which varies from as much as 0.01–50 Mbps. The link losses in these networks are typically in the order of 10^{-2}, which is quite high.

The cellular networks have a moderately high bandwidth and high latencies, hence moderate BDP. They have a moderate wireless losses (10^{-3}) and link asymmetry. These networks often experience bandwidth oscillations and delay variations due to mobility of nodes and consequently handover the connection to a new base station.

Wireless LANs (WLANs) have a low latency and a high bandwidth. However, due to the link level retransmission scheme to handle the wireless losses, WLANs also have delay fluctuations. In WLANs, uplink and downlink channels are not independent as in cellular and satellite networks, but compete with each other for shared bandwidth.

Ad hoc wireless networks are multi-hop networks in which the nodes use multi-hop relaying technique to communicate with the nodes that are not directly reachable. Like WLANs, the medium is shared among the nodes. They have limited bandwidth and the nodes in these networks have limited battery power. Besides, they have a very high packet loss because of high channel contentions and frequent path breaks due to mobility of the nodes. These networks have a low BDP.

1.2 Limitations of TCP in wireless networks

There has been a huge growth in the use of wireless networks in recent times and consequently, adapting TCP to the wireless networks is a hot research topic. The following are some of the problems faced by TCP in wireless networks [2, 3].

1.2.1 Wireless losses

In wired networks, congestion is the main cause of packet loss. But, in wireless networks packet loss could also happen due to erroneous wireless links, unstable channel conditions, or user mobility. TCP assumes all these losses to be congestion losses and reduces its cwnd size, which adversely affects the throughput.

1.2.2 Bandwidth-delay product

As satellite links have a high bandwidth-delay product (BDP), TCP should increase its cwnd aggressively in these networks. In contrast, as ad hoc networks have low BDP, TCP should follow a conservative approach to increase its cwnd. But, in congestion avoidance (CA) phase, TCP increases its cwnd by $\frac{1}{\text{cwnd}}$ for every TCP acknowledge packet (ack) it receives, which is too small in satellite networks and too large in ad hoc networks. Hence, TCP should dynamically adjust its increment factor depending on the network in which it operates.

1.2.3 Reactive nature of TCP

Transport Control Protocol keeps increasing its cwnd until it experiences packet loss. As TCP is reactive rather than proactive for avoiding congestion, it experiences a high amount of packet loss. TCP needs to resend the lost packets which not only reduces the throughput, but also severely affects the ad hoc networks, as they are severely constrained by the bandwidth and battery power.

1.2.4 Asymmetric paths

The satellite and cellular links have asymmetric paths. As the bandwidth in the forward and reverse paths could be different, congestion in the reverse path would increase the round-trip time (RTT), which results in a slow increase in cwnd; thereby leading to a drop in the throughput.

1.3 Goals

We aim to design a TCP variant which works in both the single-hop wireless networks (such as WLAN, cellular, or satellite networks) and the multi-hop ad hoc wireless networks, while attaining the following goals.

1.3.1 Throughput improvement

We want to improve the throughput attained in heterogeneous wireless networks while minimizing the packet losses.

1.3.2 Fairness

Attaining a good throughput while the competing flows starve is not a good solution. Hence, we want to achieve fairness among the competing flows.

1.3.3 No support from the network

Our proposal should not take any explicit help from any of the *network components*.[1] Because in reality, one cannot expect making changes at all the network components.

2 Related work

A lot of research has been done to adapt TCP to address the challenges in the wireless domain. However, most of the work was done for specific wireless network paradigms or requires explicit support from the network components. A comparison of the existing proposals addressing this issue is given in Table 1.

The proposals [4–7] are designed specifically for satellite or high BDP networks. These proposals suggest an aggressive increase of cwnd to exploit the available bandwidth in the networks. But, TCP-Peach+ [4], and XCP [5] rely on an explicit support from the network

[1] Throughout this paper, we use the term *network components* to denote the intermediate nodes on the path or the receiver.

Table 1 Comparison of TCP variants for wireless networks

Protocol	TCP semantics	Modification required	Targeted networks	Reactive/ proactive	Packet transmission	Type of network support required
Peach+ [4]	End-to-end	Router, end stations	Satellite	Proactive	Window based	Priority mechanism at routers
XCP [5]	End-to-end	Router, end stations	Satellite	Proactive	Window based	Congestion notification
HS-TCP [6]	End-to-end	Sender	Satellite	Proactive	Window based	–
Compound TCP [7]	End-to-end	Sender	Satellite	Reactive	Window based	–
WTCP [8]	End-to-end	End stations	Cellular	Proactive	Rate based	Receiver classifies losses and calculates sending rate
Freeze-TCP [9]	End-to-end	End stations	Cellular	Reactive	Window based	Receiver freezes cwnd in the presence of impending disconnections
Snoop [11]	End-to-end	Base station (BS)	WLAN	Reactive	Window based	BS retransmits lost packets
I-TCP [12]	Split	Base station	WLAN	Reactive	Window based	BS maintains two separate connections with the end nodes
TCP-DCR [13]	End-to-end	Base station	WLAN	Reactive	Window based	BS retransmits lost packets
TCP-AP [14]	End-to-end	Sender, routers	Ad hoc	Proactive	Pacing	Feedback about the bandwidth along the path
TCP-FeW [15]	End-to-end	Sender	Ad hoc	Reactive	Window based	–
Learning-TCP [16]	End-to-end	Sender	Ad hoc	Proactive	Window based	–
TCP NewJersey [17]	End-to-end	Sender, routers	WLAN and cellular	Proactive	Window based	Explicit congestion notification
TCP Westwood [18]	End-to-end	Sender	WLAN and satellite	Proactive	Window based	–
ATL [19]	End-to-end	Sender and all wireless nodes	WLAN, cellular, and satellite	Reactive	Window based	Receiver supplies p_w and d_w values
UL-TCP [this paper]	End-to-end	Sender	Ad hoc, WLAN, cellular, and satellite	Proactive	Window based	–

components for their operation. Although HS-TCP (High Speed TCP) [6] and Compound-TCP [7] do not need any explicit network support, they are designed and evaluated exclusively for high BDP networks. The proposals [8–10] are designed for cellular networks. However, the assumptions made in WTCP [8] such as, presence of very low and highly variable bandwidth and high delays, make it suitable only for cellular networks. Further, these proposals need an explicit support from the network components. The proposals [11–13] are designed exclusively for WLANs. Further, Snoop [11] and I-TCP [12] seek changes at the network components. TCP-DCR [13] also faces a similar problem. The proposals [14–16] are designed for ad hoc networks and they do not rely on explicit network support. TCP Adaptive Pacing [14] assumes in its rate estimation the availability of uniform bandwidth in all the wireless links, which is usually not the case. The small additive increase factor used by TCP-FeW [15] makes the protocol more conservative, thereby making it suitable only for the ad hoc networks. The proposal in [16] works for ad hoc networks. However, its congestion estimation algorithm uses inter-arrival times, which are prone to high fluctuations in high BDP networks, thereby restricting its applicability only for ad hoc networks. Further, it uses Finite Action-set Learning Automata, because of which it is not capable of making finer updates in the cwnd size.

The proposals [17, 19, 20] are designed for more than one kind of network. TCP-NewJersey [17] was shown to perform well in WLANs and cellular networks. However, it relies on explicit congestion warnings from the network. TCP-Westwood [18] achieves improved performance over TCP in WLANs and satellite networks, while achieving fairness. However, Westwood depends completely on acks (i.e., inter ack arrival times) because of which its performance degrades when it is used in high delay links. This is due to the poorer accuracy in the bandwidth estimation done based on the delayed acks [19]. Adaptive Transport Layer (ATL) [19] has been designed for the single-hop wireless access networks, such as WLAN, cellular, and satellite networks. The key idea of ATL is the following. For a TCP flow, ATL tries to attain the throughput obtained in the wired part of the network ($\hat{T}$) while operating in a wireless network. It uses the TCP Friendly Rate Control (TFRC) equation [20] to obtain $\hat{T}$, then using $\hat{T}$, as shown in Eq. 1, it adjusts the additive increase factor (α) that reflects in the aggressiveness needed in increasing cwnd size.

$$\alpha = \frac{bp(1-\beta)}{2(1-\beta)}\left[\hat{T}\left(2R + 3T_0 p(1+32p^2)(1+\beta)\right)\right]^2 \qquad (1)$$

Here, p, β, R, T_0, and b represent packet loss probability, multiplicative decrease factor, end-to-end RTT, initial retransmission timeout, and the number of data packets acknowledged with a single ack, respectively. One important observation from Eq. 1 is that α is proportional to p and R, hence as the values of p and R increase with the increasing congestion in the network, α also increases, thereby increasing cwnd with higher amounts during the congestion. This makes the protocol more aggressive and leads to a high amount of packet loss during the congestion. We compare our work with ATL as it was proposed for WLANs, satellite, and cellular networks and had a performance gain over Snoop, WTCP, Peach+, and Westwood. However, ATL has several limitations which make its deployment extremely difficult. (a) ATL tries to attain the throughput of the wired part of the network by increasing its α aggressively. This causes an aggressive increase in cwnd thus causing unfairness to the competing flows in the wireless part of the network. (b) ATL is a reactive protocol. It increases its cwnd aggressively and results in a huge packet loss, which leads to a high battery power consumption. (c) ATL needs the receiver to explicitly supply the wireless link loss and delay values at the receiver's side. Hence, our objective is to reduce the packet loss and improve fairness to the competing flows while trying to attain the goodput of ATL, without any explicit feedback from the network.

3 Overview of our protocol

Here, we first give a brief introduction to learning automata followed by an overview of our protocol henceforth called as, Unified Learning-TCP (UL-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 [21]. 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 to be taken, 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. Figure 1 shows the interactions between the automaton and the environment.

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

State of the automaton at any instant n, denoted by $\phi(n)$, is an element of the finite set $\Phi = \{\phi_1, \phi_2, \ldots, \phi_s\}$, where s represents the number of states.
Output (or) *action* of an automaton at any time instant n, denoted by $\alpha(n)$, is an element of the finite set $\alpha = \{\alpha_1, \alpha_2, \ldots, \alpha_r\}$, where r represents the number of actions.
Input (or) *Feedback* of an automaton at any time instant n, denoted by $\beta(n)$, is an element of the finite or infinite set $\beta = \{\beta_1, \beta_2, \ldots, \beta_m\}$ or $\beta = \{(a,b)\}$, where m is number of discrete values in the response from the environment and a and b are real numbers.
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. $\phi(n+1) = F[\phi(n), \beta(n)]$.

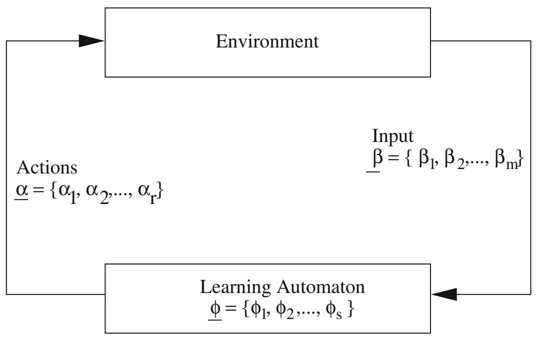

Environment
Input
$\underline{\beta} = \{\beta_1, \beta_2, \ldots, \beta_m\}$
Actions
$\underline{\alpha} = \{\alpha_1, \alpha_2, \ldots, \alpha_r\}$
Learning Automaton
$\underline{\phi} = \{\phi_1, \phi_2, \ldots, \phi_s\}$

Fig. 1 Learning automaton operation in stochastic environment

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 $\{\alpha, \beta, A\}$, 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 models of learning automata can be classified based on the number of actions in the action set (α). The finite action-set learning automata (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; these large number of actions may increase the time to update the action probabilities and also complicate the process of decision making.

A natural choice in such a case, would be to use continuous action-set learning automata (CALA) [22] 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 received 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 Sect. 4.

3.2 Details of UL-TCP

TCP increments its cwnd by a deterministic value. In the CA phase, it increases its cwnd by one maximum segment size (MSS) for every round-trip time (RTT). This limits the extensibility of TCP to various networks. For instance, as the satellite networks have a high BDP, a larger cwnd increment is needed to quickly utilize the large bandwidth available. In contrast, in ad hoc networks a more conservative cwnd increase gives better performance [15]. Also because of its reactive nature, TCP experiences a high packet loss, which leads to a poor utilization of network resources.

To overcome these problems, in this paper, we propose a learning-based mechanism, which uses CALA, to dynamically update the cwnd based on the network conditions by learning the effective amount of update (increase or decrease) in the cwnd size. The notion of an action in CALA here refers to the amount of update in the cwnd size. The motivation behind selecting CALA to our problem is the following. In CALA the size of the action set is infinite due to which more accurate mapping of network response to the actions is possible. Further, as the actions are learned (or the action probability distribution is updated dynamically based on the network condition), it is possible to increase the cwnd aggressively or conservatively, based on the network in which it is operating. For example, when the available bandwidth is high (satellite networks), the μ of the action probability distribution will be high. Thus, it can update congestion window size aggressively. On the other hand, in ad hoc networks as the available bandwidth is low, the μ will be low. Thus, it can operate conservatively in the ad hoc networks.

The following steps, also shown in Fig. 2, give a brief description of the protocol. They are explained in detail in Sect. 4.

Step 1: Getting the network conditions: Using the fluctuations in RTTs and the throughput in the forward-path, we capture the congestion and throughput fluctuations in the path into the parameters λ_1 and λ_2, respectively. Further, in order to minimize the number of timeouts, using the cwnd sizes at which previous timeouts have happened, we estimate the cwnd size at which the next timeout is likely to happen. Taking this as

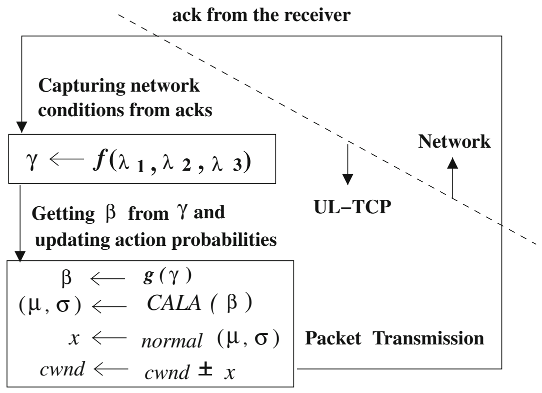

ack from the receiver
Capturing network conditions from acks
$\gamma \leftarrow f(\lambda_1, \lambda_2, \lambda_3)$
Getting β from γ and updating action probabilities
$\beta \leftarrow g(\gamma)$
$(\mu, \sigma) \leftarrow CALA(\beta)$
$x \leftarrow normal(\mu, \sigma)$
$cwnd \leftarrow cwnd \pm x$
UL–TCP
Network
Packet Transmission

Fig. 2 UL-TCP and its interaction with the network

a reference point, we obtain the degree of aggressiveness needed in cwnd increase as a parameter λ_3.

Step 2: Combining the parameters: The parameters λ_1, λ_2, and λ_3 are mapped into a single parameter γ, which corresponds to the network response.

Step 3: Mapping γ to β: β represents the input to the CALA learning algorithm. In order to update the action probability distribution of CALA, as it should reflect in the current network conditions, get the β from current γ. Using β, update μ and σ of the action probability distribution.

Step 4: Action Selection: Generate a normal random number x from the action probability distribution, which is updated in the previous step. Using this x, update the cwnd size, and transmit packets if the cwnd permits.

4 Protocol details

4.1 Inferring network conditions

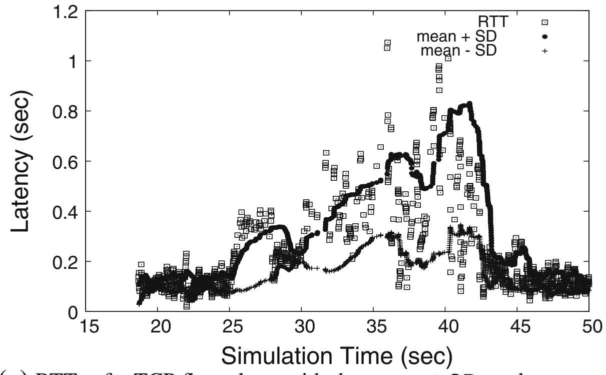
As we are not relying on any explicit support from the network, we infer the network conditions from the acks. This is the most crucial part of our work. Using the information contained in acks, we compute the RTT and the throughput in the forward-path. Then we use these in estimating the network conditions.

4.1.1 Mapping of congestion (λ_1)

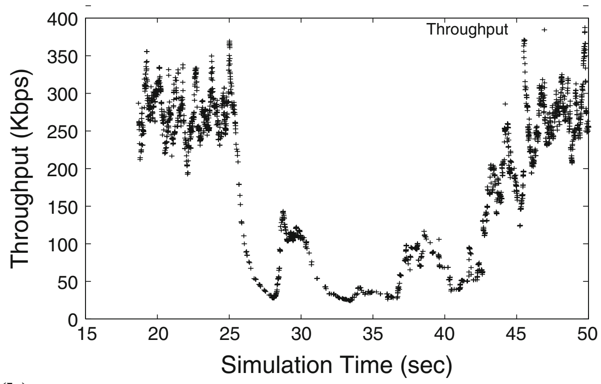
The RTT is potentially a good metric to estimate the congestion in the network [14, 23]. RTT can be obtained by enabling the timestamp option in the TCP header. In order to estimate the congestion, we keep the mean and standard deviation (SD) of most recent n RTT values. As we can observe from Fig. 3a, in the presence of congestion there is a sharp increase in the RTTs. Further, the SD increases with the increasing levels of congestion. Hence, we can detect the incipient congestion, from the SD of RTTs. The RTTs which are above mean + SD indicate a sign of congestion in the network and the RTTs below mean − SD indicate the availability of bandwidth in the network.

On a linear scale between 0 and 1, with the normalized values of mean + SD and mean − SD as the upper and lower bounds, respectively, the mean of k recent RTT values, mean$_k$ ($k < n$), is mapped to a parameter λ_1 (see Fig. 4a). We consider mean$_k$ instead of only the current RTT due to the fluctuations in the RTTs that typically occur in the wireless networks because of the bandwidth fluctuations and link-level error control. Any value below mean − SD is mapped to 1, to indicate absence of congestion in the network and any value above mean + SD is mapped to 0 to indicate that the network is congested. The following paragraph derives the relationship between the n and k by assuming that RTTs will increase during congestion [24].

Let rtt$_s$ denote the initial RTT. That is RTT when cwnd size is 1 MSS. As cwnd size grows, congestion in the network increases; as a result the RTT values will also

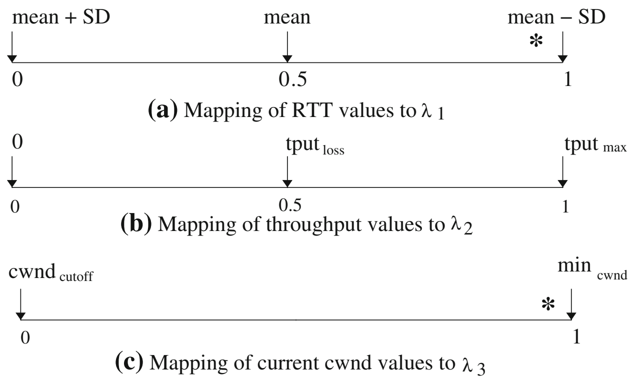


(a) RTTs of a TCP flow along with the $mean + \mathrm{SD}$, and $mean - \mathrm{SD}$ of RTTs.



(b) The corresponding fluctuations in the forward-path throughput.

Fig. 3 There is one TCP flow and three background UDP flows running between 25 and 35, 30 and 40, and 35 and 45 s, on a satellite link



(a) Mapping of RTT values to λ_1

(b) Mapping of throughput values to λ_2

(c) Mapping of current cwnd values to λ_3

Fig. 4 Mapping of network conditions to λ_1, λ_2, λ_3. Note that "*" denotes the favorable bound of λ_1 and λ_3 for cwnd increase action; for λ_2, the bound varies with network conditions

grow. Now, assume that n samples are taken in the estimation of mean and SD, for which the RTTs are rtt_s, $\text{rtt}_s + \Delta$, ..., $\text{rtt}_s + (n - 1)\Delta$. Let us assume that Δ is a growth factor by which RTT values will increase in every round. Then, $\text{mean} = \text{rtt}_s + \frac{\Delta(n-1)}{2}$, $\text{SD} = \Delta\sqrt{\frac{n^2-1}{12}}$ and, $\text{mean}_k = \text{rtt}_s + \frac{\Delta(2n-k-1)}{2}$.

Taking a general case where mean $+ c \times$ SD is the upper bound for a constant c, to capture congestion, $\text{mean}_k >$ mean $+ c \times$ SD. That is, $\text{mean}_k -$ mean $> c \times$ SD.

$$\text{mean}_k - \text{mean} = (\text{rtt}_s + \frac{\Delta(2n-k-1)}{2}) - (\text{rtt}_s + \frac{\Delta(n-1)}{2}) = \frac{\Delta}{2}(n-k)$$

As $\text{mean}_k - \text{mean} > c \times \text{SD}$

$$\Rightarrow \frac{\Delta}{2}(n-k) > c \times \frac{\Delta}{2}\sqrt{\frac{n^2-1}{3}}$$

$$\Rightarrow (n-k) > c \times \sqrt{\frac{n^2-1}{3}} \qquad \Rightarrow k < n - c \times \sqrt{\frac{n^2-1}{3}}.$$

Hence, for n samples in computing mean and SD, the above equation shows that any $k \leq 0.42 \times n$ is sufficient to identify congestion when mean + SD is used as an upper bound. Hence, we take n as 100, to consider a reasonable amount of history and get k as 42.

4.1.2 *Mapping of throughput fluctuations (λ_2)*

Throughput in the forward-path is another important parameter in wireless networks because of asymmetric paths. This can be measured simply through explicit feedbacks from the receiver. However, as this requires modifications at the receiver, we rely on TCP timestamp option to measure the forward-path throughput. As the receiver writes its timestamp in every ack it is sending, by observing the difference of every two successive timestamps and the amount of data has been acknowledged in this duration, the sender can get an approximate estimate of throughput in the forward-path. Note that to compute the forward-path throughput, the sender does not require the absolute clock time at the receiver. It just requires to know the length of clock tick at the receiver, so that the throughput can be estimated from the amount of data received by the receiver in two successive clock ticks. Hence, we compute the throughput over k acks to ensure that the sender receives acks that span at least in two different clock ticks at the receiver. Figure 3b shows the trends in forward-path throughput in the presence of background traffic. We can observe that the throughput drops drastically with the introduction of background traffic and this drop persists until the congestion ends.

In order to map the throughput to a parameter λ_2, we keep the maximum throughput (tput_{max}) observed so far. However, taking only the tput_{max} into account will not completely indicate the current network conditions as one best case tput_{max} will always try to increase the cwnd to achieve this value. Hence, we introduce the throughput at the previous loss ($\text{tput}_{\text{loss}}$) as a second reference point, at the mid scale. In the new mapping, the values between 0 and $\text{tput}_{\text{loss}}$ are mapped to [0, 0.5] and the values between $\text{tput}_{\text{loss}}$ and tput_{max} are mapped to [0.5, 1] (see Fig. 4b). Hence, when the throughput is very low, $(1 - \lambda_2)$ will be close to 1 and we should favor for increasing cwnd size. When the throughput is close to tput_{max}, we should favor decreasing cwnd size to proactively avoid the congestion.

4.1.3 *Degree of aggressiveness (λ_3)*

Retransmission timeout (RTO) has a significant impact on the performance of TCP and UL-TCP, in particular, in high BDP networks, it takes several RTTs to reach the optimal cwnd size from one MSS after an RTO. To avoid (or reduce the number of) RTOs, we bring cwnd size into feedback to control the aggressiveness needed in increasing cwnd size. The idea is to maintain an approximate size of the cwnd at which timeout is likely to occur and taking it as a reference point, update the cwnd size based on how close the cwnd size is to this reference point. For this purpose, we maintain the values for the parameters $\text{cwnd}_{\text{prev}}$ (the cwnd size at which the previous timeout has happened) and $\text{cwnd}_{\text{mean}}$ (the mean of the cwnd sizes at which all the previous timeouts have happened). From these parameters, on the arrival of each ack, using an auto-regressive technique, as shown in Eq. 2, we obtain the cwnd size at which the next timeout is likely to occur ($\text{cwnd}_{\text{cutoff}}$).

$$\text{cwnd}_{\text{cutoff}} = f(t) \times \text{cwnd}_{\text{prev}} + (1 - f(t)) \times \text{cwnd}_{\text{mean}} \qquad (2)$$

Here, $0 < f(t) < 1$, where $f(t)$ is a monotonically decreasing function with time t. The idea behind this equation is that initially $\text{cwnd}_{\text{cutoff}}$ will be close to the $\text{cwnd}_{\text{prev}}$. However, if there is no timeout for a long time, $\text{cwnd}_{\text{prev}}$ may become a bottleneck. Hence, as time progresses, $\text{cwnd}_{\text{cutoff}}$ is gradually shifted towards the average cwnd size at which most of the timeouts happened. Using $\text{cwnd}_{\text{cutoff}}$ as a reference point, we capture the degree of aggressiveness needed in cwnd increase into λ_3. On a linear scale between 0 and 1, with the normalized values of $\text{cwnd}_{\text{cutoff}}$ and minimum cwnd (i.e., one MSS) as the upper and lower bounds, respectively, the current cwnd size is mapped to λ_3 (see Fig. 4c). Any cwnd size above $\text{cwnd}_{\text{cutoff}}$ is mapped to 0. This metric is essential as timeouts result in a drastic throughput drop, especially in high BDP networks.

4.2 Mapping network conditions into a single parameter γ

Here, we discuss how to combine the parameters λ_1, λ_2, and λ_3 into a parameter γ, which provides the network response. A high γ indicates a favorable sign for cwnd increase and a low γ indicates a favorable sign for cwnd decrease. The first three cases shown below combine λ_1 and λ_2 into γ, then the fourth case combines λ_3 and γ.

case 1: During the congestion, RTT values will start growing. In this case, mean_k will be higher than mean. Therefore, we detect the congestion if $\text{mean}_k > \text{mean} + \text{SD}$ and favor cwnd decreasing actions. We determine the degree of intensity needed in decreasing cwnd size, by taking $\gamma = \min\{\lambda_1, \lambda_2\}$. As λ_2 will also be low during congestion, we take the minimum of λ_1 and λ_2 to quickly react to the congestion.
case 2: When the mean_k is lower than mean $-$ SD, the network is probably not congested and hence, the feedback should favor the cwnd increase actions. In this case, we take $\gamma = \max\{\lambda_1, 1 - \lambda_2\}$. This case is very useful in quickly increasing the cwnd after congestion. Just after congestion, the RTTs fall and hence, λ_1 will be high. Besides, just after congestion the throughput will be low and so $1 - \lambda_2$ will be high indicating that there is a lot of unused bandwidth in the network. We take the maximum of these two values as a feedback to the CALA to quickly increase the cwnd.
case 3: For the remaining values of mean_k (i.e., $\text{mean} - \text{SD} < \text{mean}_k < \text{mean} + \text{SD}$), we take $\gamma = \frac{\lambda_1+\lambda_2}{2}$, which gives equal importance to both the RTT and throughput parameters. For instance, when both λ_1 and λ_2 are high, a high value of λ_1 indicates that the network is not congested. However, λ_2 is high implies that we are close to the maximum throughput and so we should be cautious, and $\gamma = \frac{\lambda_1+\lambda_2}{2}$ takes both these factors into account.
case 4: Finally, we take the λ_3 into account as shown below. As mentioned earlier in this section, RTOs are expensive, so we try to avoid them by giving them more importance. The γ that we got from previous steps, will be adjusted as $\gamma = \frac{\gamma+\lambda_3}{2}$. When λ_3 is close to 1, γ will be adjusted between 1 and 0.5 and thus increase actions will be further favored. When λ_3 is close to 0, γ will be mapped between 0.5 and 0, and decrease actions will be further favored.

To summarize, the above mechanism for mapping λ_1, λ_2, and λ_3 to γ ensures a value close to 1 to favor increasing the cwnd size and a value close to 0 to favor decreasing the cwnd size. The mapping of γ to β, which represents the input to the CALA learning algorithm, is done based on the state of the automaton. We define two states *increase* and *decrease* for the automaton. The mapping of γ to β in these two states is described in detail in the following section.

4.3 Learning algorithm and action selection

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. As the number of actions is infinite, the CALA maintains the action probability distribution instead of probability for each action, where the probability distribution is assumed to follow a normal distribution. In CALA, the functions for updating the action probability distribution are simple. Further, CALA does not require the discretization of the action set. Further, there exists proof for the convergence of CALA learning algorithm that follows a normal distribution. The learning algorithm given in [22] 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 [22], 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 an effective amount of update (either increase or decrease) in the cwnd size. Note that when β is close to 1, the magnitude of change in μ will be high and when β is close to 0, the magnitude of change in μ is low.

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

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

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.

4.4 Discussion about the learning mechanism

The intuition behind the updating functions of the action probability distribution is as follows. Equation 3 essentially shifts $\mu(n)$ towards $x(n)$ which is an amount of update in congestion window at time step n. For a favorable response (i.e., $\beta(n)$ is close to 1), the shift in $\mu(n)$ towards $x(n)$ will be high. For unfavorable responses, (i.e., $\beta(n)$ is close to 0), there will be either no shift in $\mu(n)$ or a small shift in $\mu(n)$ towards $x(n)$. Equation 4 updates (expands or shrinks) the region of action probability distribution. An increase in $\sigma(n)$ essentially expands the action probability distribution and vice versa. When the selected action, $x(n)$ falls outside the region, then the region will be expanded for further exploration. The magnitude of the expansion depends on the response from the network. The expansion in the region is large when the response is favorable, otherwise the expansion is small. Similarly, the region will shrink to a large extent when $x(n)$ falls inside the region and gets favorable response, otherwise it shrinks by a small value. The reduction term in Eq. 4 is to make sure that $\sigma(n)$ will stay close to 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 network response, γ is treated differently in these two states. In *increase* state, when γ value is close to 1, in order to favor aggressive increase in cwnd size, the magnitude of increase in μ should be high. On the other hand, when γ value is close to 0 then there should not be any increase in μ or the magnitude of increase in μ should be low; this is to ensure no significant change in μ and in effective amount of update in cwnd size. Hence, we directly take γ as β ($\beta = \gamma$) when the automaton is in *increase* state. In *decrease* state, we map γ to β as shown in Eq. 5. The intuition is that, when the network is extremely congested ($\text{mean}_k >$ mean + SD), a significant reduction in μ is required to reduce the cwnd size aggressively. 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.

$$\beta = \begin{cases} 1 \text{ and } \mu = \frac{\mu}{2} & \text{if } \text{mean}_k > \text{mean} + \text{SD}, \\ 1 - \gamma & \text{otherwise.} \end{cases} \tag{5}$$

4.5 Convergence proof and selection of parameters

The convergence proof for CALA learning algorithm is well studied in the literature [22]. 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 [25, 26]. 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 0.01 (a small value), 0.01 ($\frac{1}{100}$ of a packet), and 2,000 (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 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 0.01. 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 sudden rise or fall in the congestion window which adversely affects the achievable goodput. The value for K affects σ. For a specific λ value, smaller values for K shifts 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 the packet loss, for the λ value of 0.01, we take K value as 2,000. As σ_L represents the lower bound on σ, as UL-TCP needs to operate in ad hoc networks along with the single-hop wireless networks, σ_L should take a smaller value. Hence, we take the value for σ_L as $\frac{1}{100}$, thus keeping the updates in cwnd as low as 0.01 of a packet.

4.6 Final protocol

The working mechanism of UL-TCP is given in Algorithm 1. At the connection startup, as UL-TCP needs to operate in wide range of wireless networks, the learning algorithm must be provided with a sufficiently large action set. Hence, we initialize the mean (μ) and standard deviation (σ) of the probability distribution with $\frac{\text{MSS}}{2}$ and one MSS, respectively. Because of these initial values, during connection startup, UL-TCP prefers actions for cwnd increase with a high probability, and increases cwnd with larger increments. We initialize $\text{cwnd}_{\text{prev}}$ with a relatively large value 2,000, which helps the protocol to perform a definite increase in cwnd during the connection startup. UL-TCP will be in the loop until the end of the TCP session or abrupt connection close due to 12 successive retransmission timeouts. One important note is that, we

Algorithm 1 Algorithm 1: The pseudo code of unified learning-TCP

```
Algorithm 1 The Pseudo Code of Unified Learning-TCP
repeat
    1. Compute $mean$ and SD over previous $n$ RTTs
    and compute $mean_k$ which is a mean of previous
    $k$ RTTs. Now get $\lambda_1$ as explained below.
    if $mean_k$ is in range [$mean$+ SD, $mean$ − SD]
    then
        map $mean_k$ linearly on a scale of [0, 1] and
        assign this mapping to parameter $\lambda_1$.
    else if $mean_k < mean$ − SD then
        $\lambda_1 = 1$
    else
        $\lambda_1 = 0$
    end if
    2. Compute the forward path throughput $tput_{curr}$
    over past $k$ packets. Now get $\lambda_2$ as explained below.
    if $tput_{curr}$ is in range [0, $tput_{loss}$] then
        map $tput_{curr}$ linearly on a scale [0, 0.5] and
        assign this mapping to parameter $\lambda_2$.
    else
        map $tput_{curr}$ linearly on a scale [0.5, 1] and
        assign this mapping to parameter $\lambda_2$.
    end if
    3. Compute the network response $\gamma$ from $\lambda_1$ and $\lambda_2$
    as explained below.
    if $mean_k > mean$ + SD then
        $\gamma = \min\{\lambda_1, \lambda_2\}$
    else if $mean_k < mean$ − SD then
        $\gamma = \max\{\lambda_1, 1 - \lambda_2\}$
    else
        $\gamma = \frac{\lambda_1+\lambda_2}{2}$
    end if
    4. Parameter $\lambda_3$ is used to avoid timeouts.
    if $cwnd > cwnd_{cutoff}$ then
        $\lambda_3 = 0$
    else
        Linearly map cwnd values from [$cwnd_{cutoff}$, 1]
        to [0, 1]
    end if
    5. $\gamma$ is adjusted as $\gamma = \frac{\gamma+\lambda_3}{2}$.
    6. Depending on the state, take $\beta$ as $\gamma$ or $1 - \gamma$,
    refer Eq. 5
    7. Using $\beta$, update the $\mu$ and $\sigma$ of action probability
    distribution of CALA (Eqs. 3 and 4).
    8. Generate a normal random number $x$ from this
    action probability distribution.
    9. Using this $x$, increase or decrease the cwnd size,
    and transmit packets if the cwnd permits.
until (the end of TCP session OR abrupt connection
close due to 12 successive RTOs)
```

allow a continuous decrease in cwnd size up to half of cwnd size that was at the start of decreasing the cwnd. For example, assume that cwnd size at the start of decreasing is 30 packets. In our algorithm, we allow continuous decrease in cwnd size, at successive time instances, up to 15 packets. This avoids any drastic fall in cwnd size and helps to quickly reach the maximum capacity offered by the network.

5 Simulation results

We conducted simulation experiments using ns-2.28. For the single-hop wireless networks (WLAN, cellular, and satellite networks) we take a dumbbell topology, which is shown in Fig. 5. Here, over a wireless link, the senders are connected to a router R1, where the R1 is connected over a wired link to the second router R2. The receivers are connected over wired links to R2. All the wired links have 100 Mbps bandwidth. All the nodes use drop-tail queues. Unless otherwise specified, the maximum queue size is taken as 64 packets and the packet size is taken as 1,460 bytes. In the ad hoc networks, we consider chain, grid, and random topologies. All simulations are conducted for 600 s. Each simulation scenario is ran for 30 different seed values and results are obtained with 95% confidence level. We use FTP traffic over TCP in all the cases. We compare UL-TCP against TCP-Newreno in both single-hop and multi-hop wireless networks. Also, in all single-hop wireless network studies, we use Sack enabled ATL [19] for comparison as it was proposed for WLANs, satellite, and cellular networks and had a performance gain over Snoop, WTCP, Peach+, and Westwood. However, as we show in this section, ATL shows a significant increase in packet loss and an unfairness to the competing TCP-Newreno flows. Hence, our aim is to reduce the packet loss and improve fairness to the competing flows while trying to attain the goodput of ATL.

V. SIMULATION RESULTS

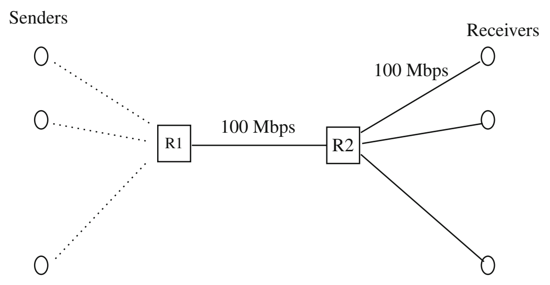


Fig. 5 Dumbbell topology

(1) *Goodput* is the number of bytes successfully transmitted per unit time.
(2) *Packet loss percentage* is the ratio of the total bytes retransmitted to the total bytes transmitted. Apart from Goodput, packet loss percentage is also an important metric to be considered in wireless networks. As a reliable transport protocol recovers the lost packets by several retransmissions, when the loss is high, the packet retransmissions consume a considerable amount of battery power. Due to this the node may drain off its battery faster, thereby reducing the network lifetime.
(3) *Bandwidth stolen* is a measure of inter-protocol fairness. If a TCP-Newreno flow has a goodput of T_1 when competing with another TCP-Newreno flow and T_2 when competing with an aggressive protocol, then bandwidth stolen by the aggressive one is $(T_1 - T_2)/T_1$ if $T_1 > T_2$, and 0 otherwise.

Note that in order to introduce the wireless losses, we use the uniform error model available in ns-2.28.

The following are main advantages of the proposed UL-TCP over TCP-Newreno and ATL.

- Unlike in TCP-Newreno, increments and decrements in cwnd size are not restricted in UL-TCP. Depending on the network, it can learn effective amount of update in cwnd size through μ and σ of the action probability distribution. Hence, it can increase the cwnd size by higher amounts when it is operating in high BDP networks. However, in low BDP networks it can even increase cwnd size by a few bytes.
- Unlike in TCP-Newreno and ATL, UL-TCP can proactively decrease the cwnd size when incipient congestion is detected, thereby it can avoid packet loss during the congestion events. Also this mechanism helps UL-TCP to overcome the problem of *congestion window synchronization* [27], which is a well known problem of TCP and ATL as their operations are deterministic in nature.
- Unlike in TCP-Newreno and ATL, the action selection in UL-TCP is probabilistic in nature. Thus, the nodes that detect the incipient congestion respond differently. Some of the nodes that detected the congestion decrease the cwnd size, while the other nodes increase cwnd size. This leads to better fairness (intra- and inter-protocol) among the competing flows as no single flow can completely dominate all the other flows. Also, it is well known that the shorter hop TCP flows often starve the longer hop TCP flows. This case is unlikely to arise in UL-TCP as the shorter hop flows cannot completely take away the 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.
- Like *slow-start threshold* in TCP-Newreno, UL-TCP uses $\text{cwnd}_{\text{cutoff}}$ to adjust the aggressiveness needed in increasing cwnd size. However, by considering the history of several timeout events and the elapsed time, UL-TCP can set a more appropriate value for $\text{cwnd}_{\text{cutoff}}$ and can avoid timeouts. This is extremely useful in satellite (high BDP) networks as timeouts result in a drastic throughput drop in these networks.

5.1 Behavior of UL-TCP

In this section, through simulation studies, we show that the proposed learning mechanism updates the congestion window size efficiently. First, we monitor the fluctuations in β (which represents the network response), the progression of mean_k (which is mean of recent k RTTs), and progression of congestion window size, for a flow running over a satellite link. Here, uplink and downlink bandwidths are kept fixed at 500 Kbps. Figure 6 plots the fluctuations in β and progression of cwnd size which show just opposite trends. As expected, we can observe that as cwnd size increases, due to increasing level of congestion in the network, β decreases. Also, the trend shown by the progression of mean_k exactly matches with the progression of cwnd (refer Fig. 7).

Figures 8 and 9 show the progression of cwnd size and the corresponding fluctuations in β and μ, for a single UL-TCP connection running over a 1 Mbps satellite link. As mentioned earlier, the trends shown by the progression of cwnd and fluctuations in β are just opposite. In Fig. 9, we can see the fluctuations in μ of the action probability distribution from which an action is drawn. Note that the action refers to an effective amount of change in the cwnd size. In the figure, we can observe that as μ increases, there is a corresponding increase in cwnd and vice versa. It is clearly seen that μ is operating at lower values when timeout happens (around 120 s). However, there after, we

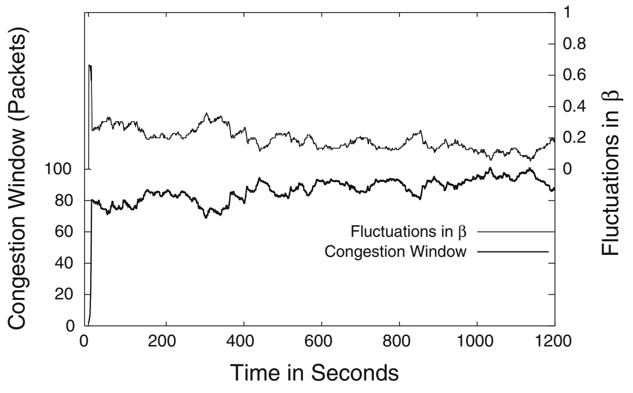


Fig. 6 Satellite link (500 Kbps): progression of cwnd versus fluctuations in β

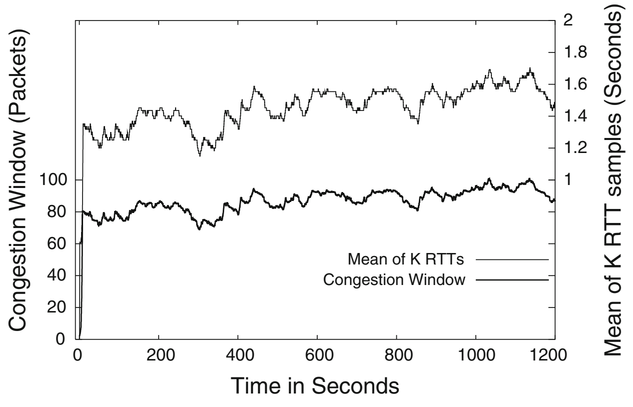


Fig. 7 Satellite link (500 Kbps): progression of cwnd versus fluctuations in mean_k

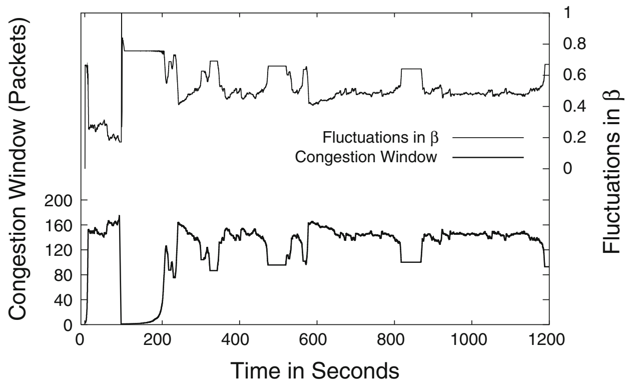


Fig. 8 Satellite link (1 Mbps): progression of cwnd versus fluctuations in β

can see a steady increase in μ and subsequently in cwnd size. As μ falls much below zero, cwnd decreases drastically. As mentioned earlier in Sect. 6, we will not allow a continuous drop below half of cwnd size that was at the start of decreasing. The flat lines in the cwnd size progression explain this fact. Figures 10 and 11 show the similar results but for a flow running for four hops in an ad hoc wireless network. We notice many fluctuations in β compared to the studies on satellite and cellular links. This is mainly due to the variations in link availability and the presence of multiple wireless links in the ad hoc networks that lead to self contentions among the packets of a flow. However, the trends in the progression of cwnd size and fluctuations in β and μ are same as in the previous study over a satellite link.

5.2 Simulations in satellite networks

As mentioned earlier, we use a dumbbell topology for the studies on the satellite networks. We use the satellite MAC available in ns-2 to get more accurate results. In satellite

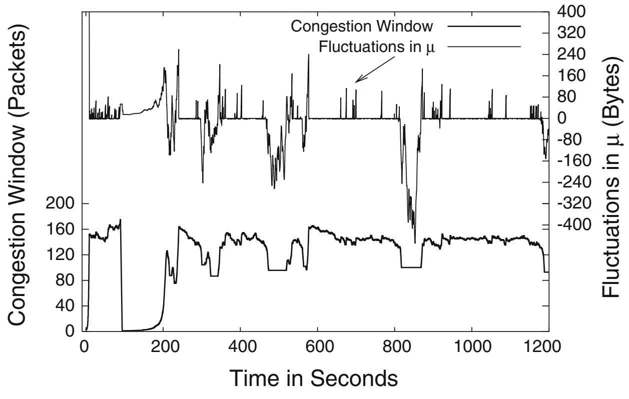


Fig. 9 Satellite link (1 Mbps): progression of cwnd versus fluctuations in μ

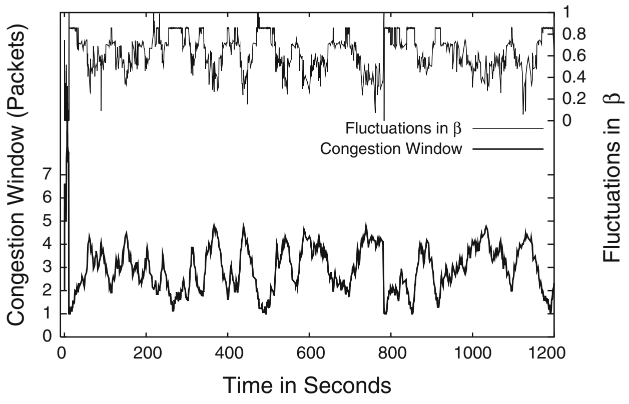


Fig. 10 Ad hoc network: progression of cwnd versus fluctuations in β

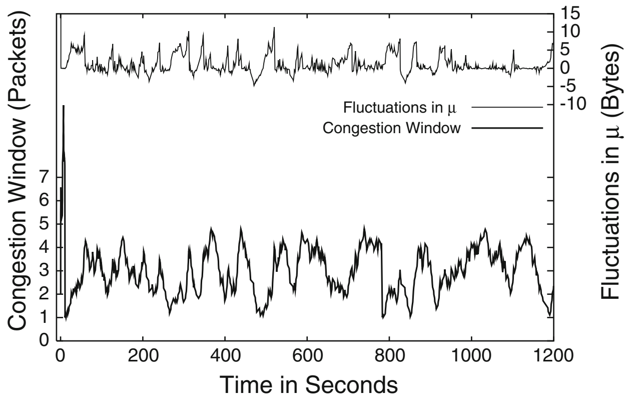


Fig. 11 Ad hoc network: progression of cwnd versus fluctuations in μ

MAC, the link delays will be automatically set based on the location of the satellite and the terrestrial node and the delay turned out to be around 250 ms as we used Geosynchronous satellite. In the initial study, we take the uplink bandwidth as 11 Mbps and later it is varied up to 40 Mbps to observe the effect of higher bandwidths on performance. We used a constant downlink bandwidth of 40 Mbps.

First, we study the performance of UL-TCP for varying wireless loss probabilities, in the presence of background traffic. We have taken nine UDP flows to generate the background traffic, where each flow operates at 20 packets per second. Figure 12 shows the goodput of a single UL-TCP flow (also TCP-Newreno and ATL) in the presence of these nine UDP flows. At all loss probabilities, UL-TCP shows significantly higher goodput than TCP-Newreno. We observed that at the typical satellite link loss probability of 0.01 [19], UL-TCP shows *four times* higher goodput than TCP-Newreno. Also, we observed that ATL shows almost *five times* higher goodput over TCP-Newreno and up to 25% higher goodput over UL-TCP. As wireless losses become dominant at the higher values of loss probability, the goodput of these three protocols decreases with increasing values of loss probability. Figure 13 shows the comparison of packet loss for these three protocols. We notice that UL-TCP shows marginally higher packet loss (up to 6%) over TCP-Newreno. ATL, on the other hand, has a significantly higher packet loss of almost *four times* and *three times* over TCP-Newreno and UL-TCP, respectively.

Next, we study the inter-flow fairness of UL-TCP and ATL and show that the goodput improvement shown by UL-TCP is not at the cost of unfairness to the competing TCP flows. We measure the mean goodput of *five* TCP-Newreno flows in the presence of five other TCP-Newreno flows. Then we measure the mean goodput of the same *five* TCP-Newreno flows in the presence of five UL-TCP flows, and subsequently in the presence of five ATL flows. We then measure the bandwidth stolen by UL-TCP and ATL in Fig. 14. We can observe that UL-TCP achieves better fairness than ATL. UL-TCP steals a maximum of 18% of bandwidth from the competing TCP-Newreno flows at link loss probability of 0.001 and in all other cases it steals about 10%. However, ATL steals a maximum of 92% of bandwidth from the competing TCP-Newreno flows, which is certainly unacceptable. The reason for the ATL's unfairness is the following. ATL increases its cwnd aggressively, thereby causes congestion losses and forces the competing flows to half their cwnd even before they get their fair share of the bottleneck bandwidth. This is clearly visible at low loss probabilties where congestion losses are dominant. Since wireless losses are dominant at high loss probabilities, the effect of ATL's aggressiveness is not that visible, hence it steals low bandwidth at high loss probabilities. Figure 15 shows the corresponding cumulative goodput attained by both UL-TCP and TCP-Newreno when we have *five* flows of each type competing together. At all loss probabilities, UL-TCP has a significant improvement

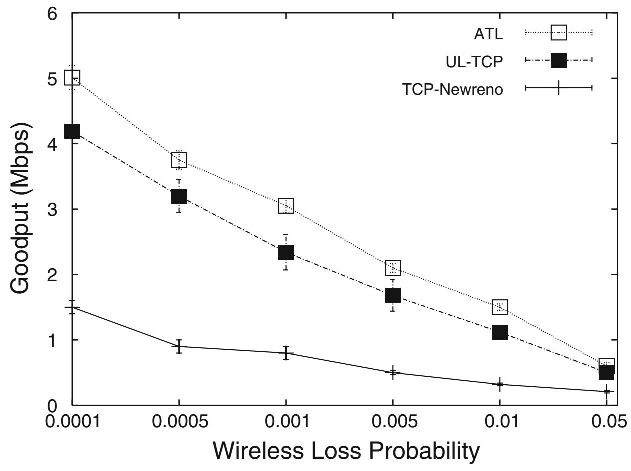


Fig. 12 Single flow: goodput in satellite networks for varying loss probability

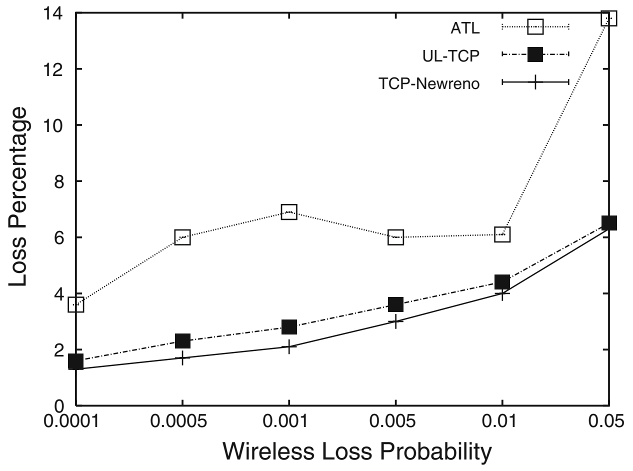


Fig. 13 Single flow: packet loss percentage in satellite networks for varying loss probability

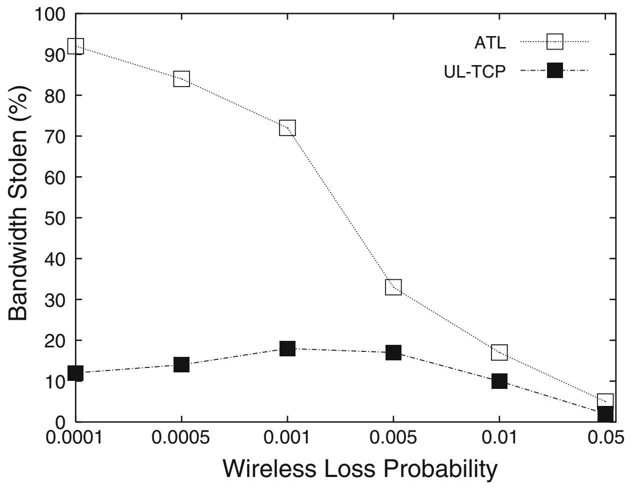


Fig. 14 Five UL-TCP (ATL) flows competing with Five Newreno flows. Bandwidth stolen in satellite networks

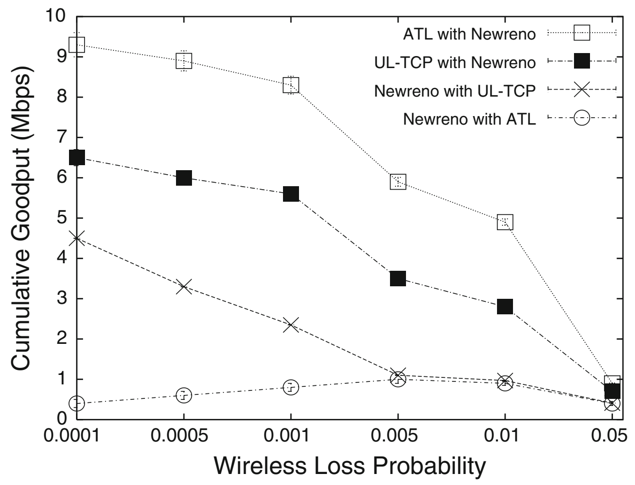


Fig. 15 Five UL-TCP (ATL) flows competing with Five Newreno flows. Cumulative goodput in satellite networks

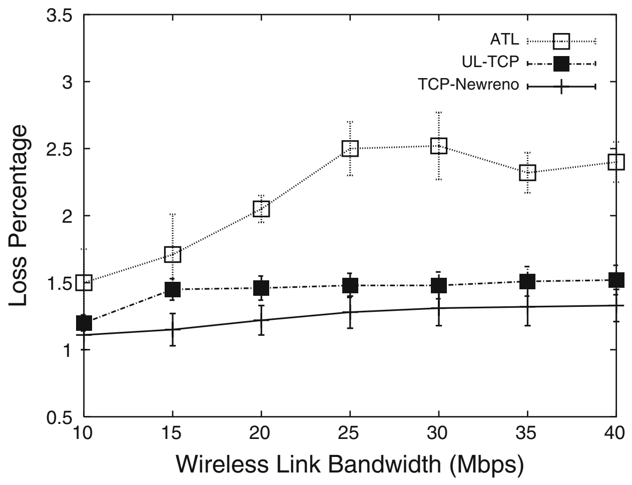


Fig. 17 Single flow: packet loss percentage in satellite networks at varying bandwidth

in cumulative goodput over TCP-Newreno. Also, we can notice that ATL achieves higher cumulative goodput by causing severe unfairness to the competing TCP-Newreno flows.

Our final study related to satellite networks is the following. As there is a large variation in the satellite link bandwidth from as much as 0.01–50 Mbps, we study the goodput achieved by a UL-TCP flow with varying link bandwidth at a typical satellite link loss probability of 0.01. In Fig. 16, we can notice that TCP-Newreno is not responding to the increase in bandwidth. On the other hand, both UL-TCP and ATL show an improvement in goodput with the increase in bandwidth. As UL-TCP tries to balance between goodput and fairness (also packet loss), the goodput improvement is not proportional to the increase in bandwidth. Figure 17 shows the comparison of the corresponding packet losses. At all link bandwidths, ATL shows

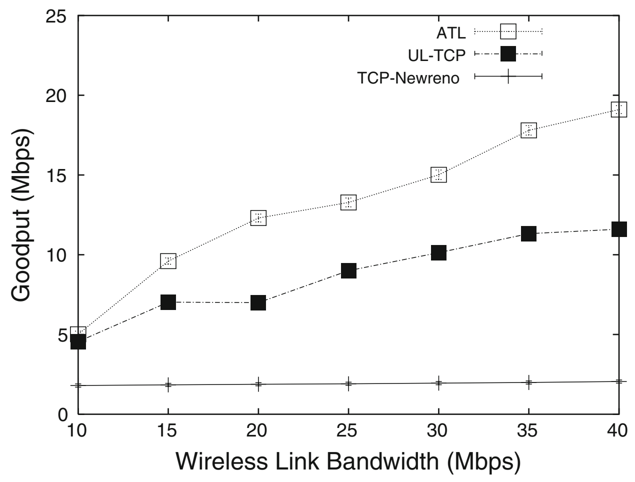


Fig. 16 Single flow: goodput in satellite networks at varying bandwidth

significantly higher losses over UL-TCP and TCP-Newreno, where as UL-TCP shows slightly higher packet loss over TCP-Newreno.

To summarize the results, we observed that UL-TCP shows a significant improvement in goodput and in fairness to the competing flows while showing a marginal increase in packet loss over TCP-Newreno. Further, UL-TCP shows a significant reduction (up to 70%) in packet loss and improvement in fairness (up to 88%) over ATL while losing in the goodput up to 21%.

5.3 Simulations in cellular networks

In this section, we discuss the performance of UL-TCP in cellular networks. We take the same dumbbell topology (Fig. 5). To the best of our knowledge, there is no cellular MAC available in ns-2.28. Hence, as mentioned in [2], we model the cellular link by introducing the typical characteristics of cellular link such as delay and bandwidth, over a wired link. As suggested in [2], delay and bandwidth values over uplink are taken as 100 ms and 500 Kbps, respectively and downlink delay and bandwidth are taken as 100 ms and 1 Mbps, respectively. Simulation setup for the studies *varying the wireless loss probabilities* and *multi-flow cumulative goodput and fairness* is same as that of the corresponding satellite network studies.

Figures 18 and 19 show the goodput and packet loss for a single UL-TCP/TCP-Newreno/ATL flow for varying wireless loss probability. The trends are similar to those of the corresponding study in satellite networks. UL-TCP shows significantly higher goodput than TCP-Newreno, with an improvement of 78% at the typical cellular wireless loss probability of 0.001 [19]. ATL shows higher goodput over TCP-Newreno (up to 133%) and UL-TCP (up to 25%). However, ATL shows significantly higher packet

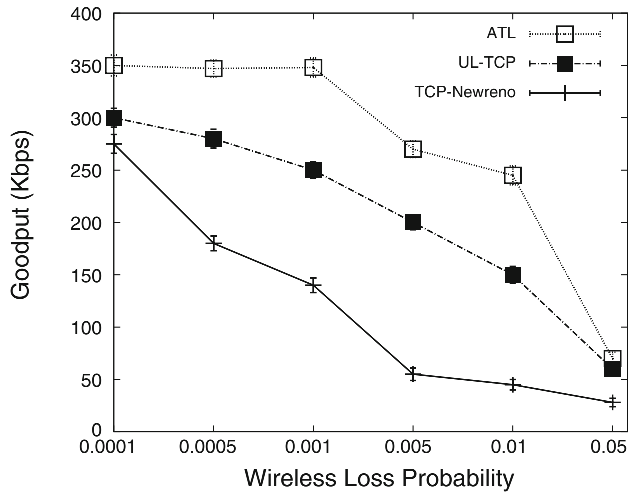


Fig. 18 Single flow: goodput in cellular networks for varying loss probability

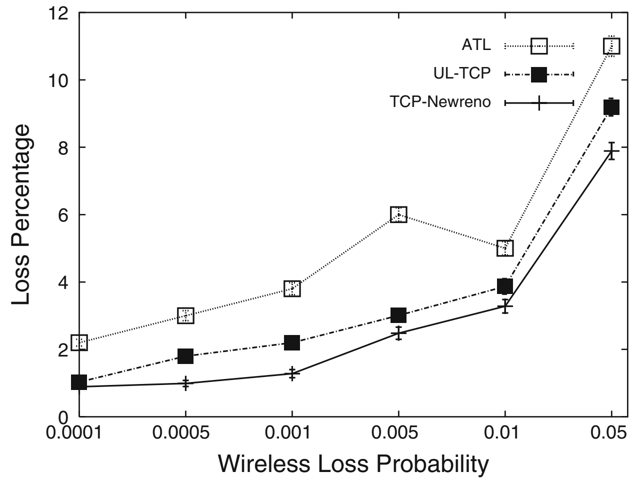


Fig. 19 Single flow: packet loss percentage in cellular networks for varying loss probability

loss than both TCP-Newreno (up to 200%) and UL-TCP (up to 113%). As this high packet loss is recovered through several retransmissions, the ATL nodes spend significant portion of their battery power on recovering from losses.

Figure 20 shows the bandwidth stolen by ATL and UL-TCP, respectively, when they are competing with TCP-Newreno flows. The trends are similar to those of the corresponding study for satellite networks. UL-TCP shows a significantly better fairness to the competing TCP-Newreno flows than ATL. The bandwidth stolen by UL-TCP is high when congestion losses are dominant (i.e., for the loss probabilities $10^{-4} - 10^{-2}$). The five UL-TCP flows together steal a maximum of 18% of bandwidth from the competing five TCP-Newreno flows. The five ATL flows on the other hand steal up to 68.8% of TCP-Newreno's

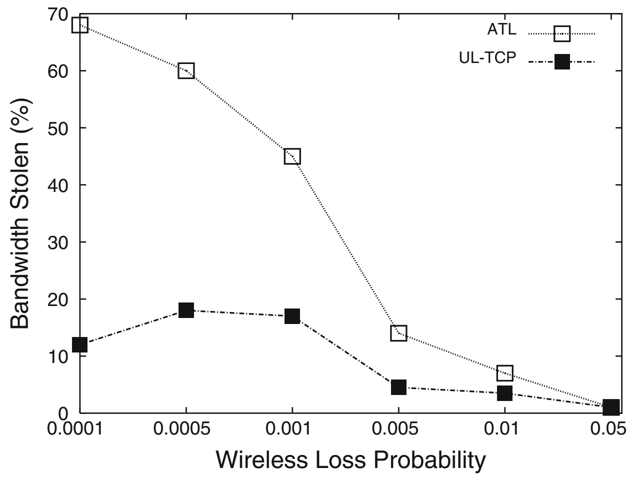


Fig. 20 Five UL-TCP (ATL) flows competing with Five Newreno flows. Bandwidth stolen in cellular networks

bandwidth, which is quite high. The corresponding cumulative goodput results are shown in Fig. 21. At all loss probabilities, UL-TCP shows higher cumulative goodput than TCP-Newreno. These results confirm the fact that ATL achieves higher goodput while causing severe unfairness to the competing TCP-Newreno flows.

Due to the variations in the number of active users and also due to mobility of nodes, the cellular links are prone to bandwidth fluctuations [2]. Hence, we conduct a study by oscillating the bandwidth around the mean value of 500 Kbps. For example, [4, 6] in Fig. 22 means that the bandwidth will oscillate between two values, 600 and 400 Kbps, staying for 5 s at each value before changing to the other. The goodput comparison at the typical cellular loss probability of 0.001 is shown in Fig. 22. UL-TCP performs better than TCP-Newreno for all the ranges of bandwidth oscillations and shows almost *two times* higher

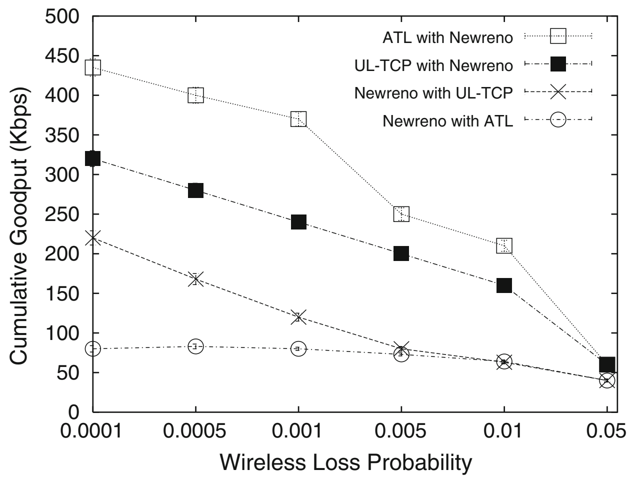


Fig. 21 Five UL-TCP (ATL) flows competing with Five Newreno flows. Cumulative goodput in cellular networks

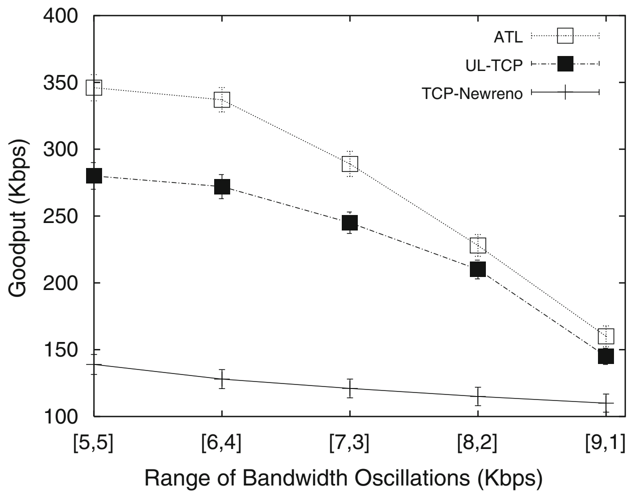


Fig. 22 Single flow: goodput in cellular networks with bandwidth oscillations

goodput than TCP-Newreno. As the bandwidth fluctuations increase, due to the aggressive nature, goodput of both UL-TCP and ATL started decreasing, but the aggressiveness of UL-TCP is not leading to an excess packet loss (refer Fig. 23), whereas ATL's packet loss is increasing exponentially with increasing variations. ATL experiences about *six times* higher packet loss over TCP-Newreno and about *five times* higher packet loss over UL-TCP.

In order to study the effect of link outage (due to mobility) on the performance of UL-TCP, we conduct the following study. We take a fixed value of 10 ms for link outage time and vary the link active time from 50 to 600 s. During the link active time, the link operates at 1 Mbps and during the link outage time the link bandwidth is set to 0. The goodput and the corresponding packet loss percentage for a single flow are shown in Figs. 24 and 25, respectively.

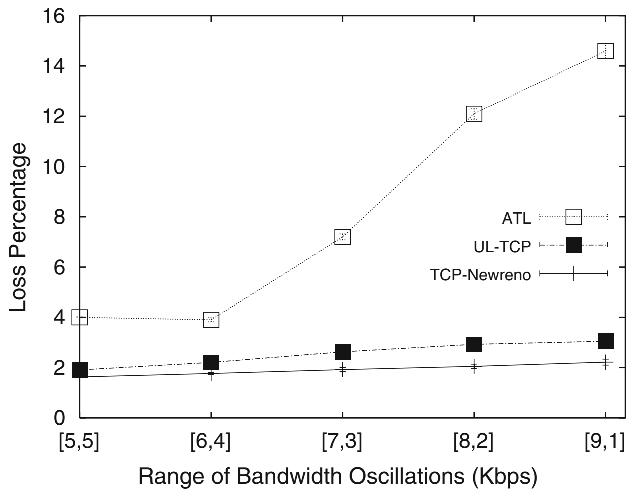


Fig. 23 Single flow: packet loss percentage in cellular networks with bandwidth oscillations

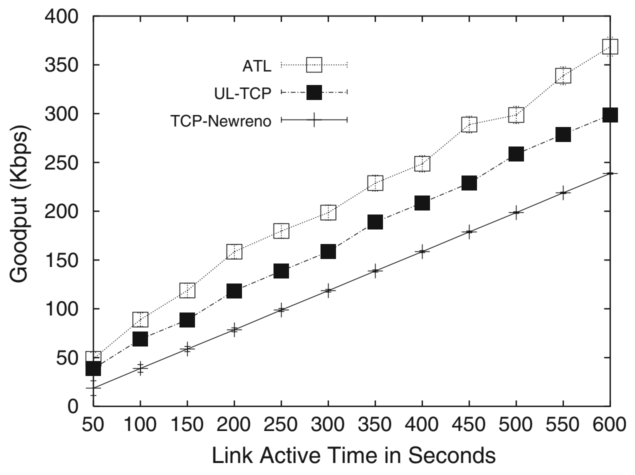


Fig. 24 Single flow: goodput of a TCP connection for varying duration of connection uptime over a cellular link

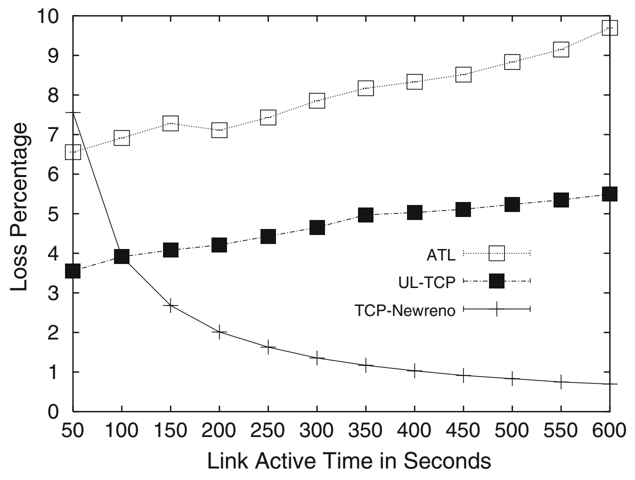


Fig. 25 Single flow: packet loss percentage of a TCP connection for varying duration of connection uptime over a cellular link

All the three protocols show an increase in goodput for increasing values of link active time. For smaller values of link active time, the goodput shown by UL-TCP is same as that of ATL. However, due to its extreme aggressiveness, ATL shows higher goodput and experiences significantly higher packet loss over UL-TCP and TCP-Newreno for increasing link active times. UL-TCP, on the other hand, shows consistently higher goodput with a moderate increase in packet loss over TCP-Newreno. UL-TCP maintains cwnd size at which the timeout may happen and cautiously increases cwnd size as it approaches to that value. Because of this, for smaller values of link active times, it gets benefited as it experiences lesser amount of losses and thus attaining higher goodput compared to TCP-Newreno. As it can adjust the cwnd size in a better way when the network is stable, it continues to show higher goodput over TCP-Newreno, with increasing link active times.

5.4 Simulations in wireless LANs

In this section, we study the performance of UL-TCP over Wireless LANs. We use IEEE 802.11 MAC operating at 2 Mbps in these simulations. The rest of the simulation setup details are same as those of satellite networks. The results are consistent with our studies in the satellite and cellular networks.

Figures 26 and 27 show the goodput and packet loss, respectively, for varying wireless loss probabilities. UL-TCP shows up to 20% higher goodput over TCP-Newreno and almost same goodput as that of ATL. However, ATL shows almost *four times* higher packet loss over TCP-Newreno and about *two times* higher packet loss over UL-TCP, whereas UL-TCP shows about 40% higher losses over TCP-Newreno. Next, we will study the behavior of UL-TCP at higher loads in the network. We carry out this study by increasing the number of TCP flows with a typical loss probability of 10^{-4} [19]. Figures 28 and 29 show the cumulative goodput and packet loss, respectively. UL-TCP shows consistently higher goodput over TCP-Newreno for increasing number of flows. It shows the maximum improvement in goodput of about 10% when there are 30 flows. ATL also shows about 12% higher goodput over TCP-Newreno and its goodput is approaching to that of UL-TCP with increasing number of flows. ATL experiences about *three times* higher packet loss over TCP-Newreno. On the other hand, UL-TCP shows a marginal increase in packet loss for most of the cases, and experiences a maximum of 38% higher packet loss over TCP-Newreno.

We also conducted a study to measure the fairness of UL-TCP to the competing flows in WLANs. Like our fairness related studies in satellite and cellular networks, here also we observed that UL-TCP is perfectly fair to the competing flows. As the trends are similar, we avoid giving the corresponding results (graphs) in this paper.

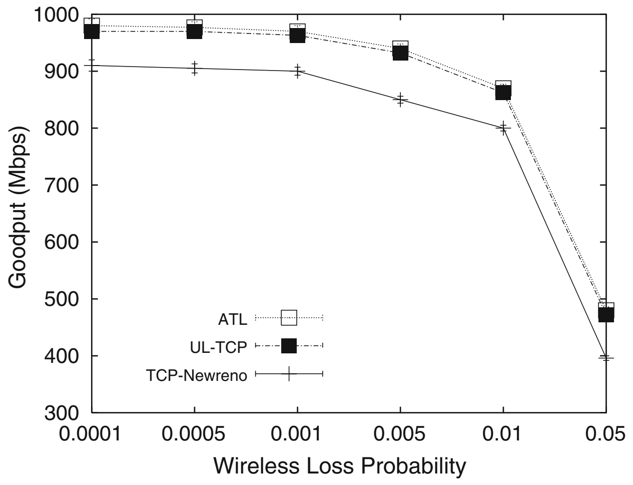


Fig. 26 Single flow: goodput in WLAN for varying loss probability

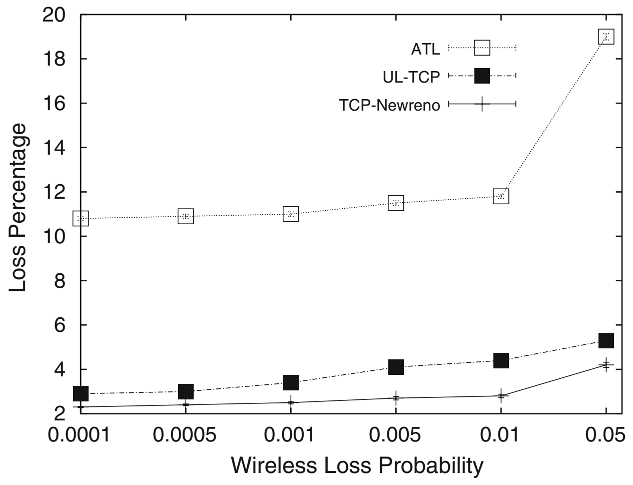


Fig. 27 Single flow: packet loss percentage in WLAN for varying loss probability

5.5 Simulations for short-lived flows in satellite, cellular, and wireless LANs

Short-lived flows can be expressed as a Pareto distribution which is a power-law distribution used in a large number of real-world situations. Pareto distribution is defined in terms of mode (a modal value which is also the minimum value) and shape factor (defines the concentration of data towards mode). In order to evaluate the performance of UL-TCP for

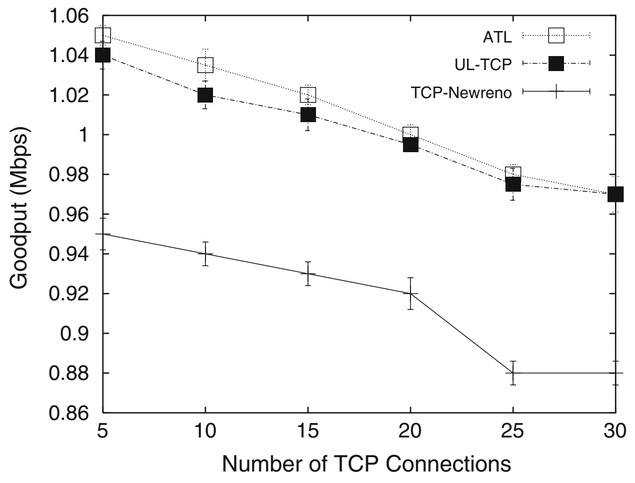


Fig. 28 Cumulative goodput for varying number of flows in WLAN

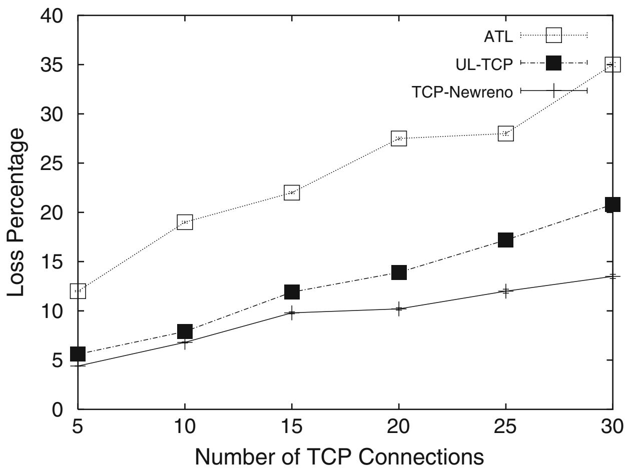


Fig. 29 Average packet loss percentage for varying number of flows in WLAN

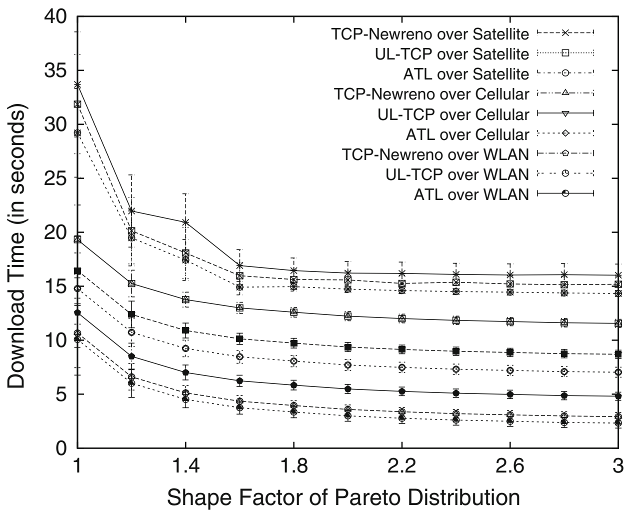


Fig. 30 Download time for varying shape factor with fixed number of flows 20 and mode 50 packets

short-lived flows, we conduct two different studies. Table 2 shows the simulation parameters used in these experiments.

In the first study, we take a fixed mode value of 50 (i.e., minimum 50 packets for each flow) and vary the shape factor between 1 and 3. We do this experiment for 50 flows which start at a random time selected from an exponential distribution with mean value 2 s. Figure 30 shows the average flow completion time in Satellite, Cellular, and WLANs. As the deviations in file sizes from mode are high for smaller values of shape factor, we can observe higher completion times initially. For the larger values of shape factor, the file sizes will be closely populated around mode, therefore the completion times are relatively small. In the next study, we have taken a fixed value for shape factor as 1.2 as this is the point at which there are moderate deviations from mode 50 (in the previous study) and varied the number of flows between 20 and 100. These results are presented in Fig. 31. We can observe that, as the number of flows increases in the network, the download time is also increasing. One common observation in these two is that UL-TCP and TCP are doing almost close, and ATL is doing well among all. Note that as mentioned in Sect. 5, UL-TCP learns the network conditions after first few RTOs, but the short term flows typically end even before the initial timeout. However, the initial values for μ and σ, and $\text{cwnd}_{\text{prev}}$ will help UL-TCP to perform similar to TCP at the connection startup. As the number of flows is increasing (in Fig. 31), RTOs are likely to happen sooner, therefore UL-TCP performs well over TCP at higher number of flows.

Table 2 Simulation parameters

Network	Wireless loss probability	Uplink (downlink) bandwidth	Link delay
Satellite	10^{-2}	500 Kbps (1 Mbps)	250 ms
Cellular	10^{-3}	500 Kbps (1 Mbps)	100 ms
WLAN	10^{-4}	11 Mbps	Not applicable

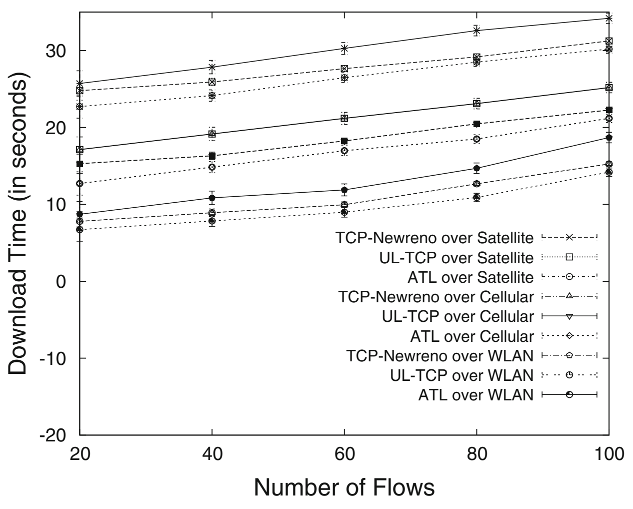


Fig. 31 Download time for varying number of flows with fixed values for shape factor and mode as 1.2 and 50 packets, respectively

5.6 Simulations over multi-hop wired network

In general, the flows originated in a wireless network run for several hops in the wired network. While it is common to find the resources of a wireless network as bottleneck, sometimes it is true with the resources in wired part. Therefore, in this section we study the performance of UL-TCP in a multi-hop wired network. We take a simple parking-lot topology with four wired routers (each router is serving five nodes) and a last-hop wireless network (which consists of twenty nodes). In order to create the bottleneck in the wired network, the bandwidth and delay for the wired links are taken as 10 Mbps and 10 ms, respectively, and the parameters of wireless links are retained as mentioned in the initial section (for example, we used uplink and downlink bandwidths for a satellite link as 11 and 40 Mbps, respectively). A flow starts randomly at a wireless node and runs for several hops in the wired network before reaching the destination. It is obvious from this topology that each flow experiences a different level of congestion. We conduct this study for 20 flows, where the starting times of the flows are derived from an exponential distribution with mean 2 s. We measured intra-flow fairness and average goodput over all the flows going to a specific hop-length. Note that intra-flow fairness is different from the bandwidth stolen which is used in previous sections. While bandwidth stolen represents the inter-flow fairness (flows of different type), intra-flow fairness shows the fairness between the flows of same type, which is obtained using Jain's fairness index formula (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.) Figure 32 shows the intra-flow fairness and goodput results, which are obtained over a satellite link. We can observe that, while achieving better fairness among the competing flows, UL-TCP shows moderate improvement in goodput over TCP-Newreno; also, the goodput of the flows running for shorter hops is higher than that of the flows running for longer hops. As we observed similar trends in the other networks also, we do not keep those results.

5.7 Simulations in ad hoc wireless networks

We now study the performance of UL-TCP in ad hoc networks and compare it with that of TCP-Newreno. Note that ATL was not designed for ad hoc networks. Even if ATL is extended to ad hoc networks, the aggressive nature of ATL would not be suitable for these networks. Hence, we use Learning-TCP [16] for comparison as it was shown to perform better over TCP-Reno in ad hoc networks. Unlike UL-TCP, Learning-TCP has a fixed number of actions, therefore it can not make finer updates in cwnd size. We perform simulations on chain and grid topologies and then under random placement with mobility. The transmission and carrier sense ranges of the nodes are set to 250 and 550 m, respectively. The routing and MAC protocols used are ad hoc on-demand distance vector routing (AODV) protocol and IEEE 802.11 with 2 Mbps, respectively. The nodes use TwoRayGround as a propagation model and drop-tail queues with maximum queue size of 50 packets. One main advantage of UL-TCP over TCP-Newreno in ad hoc networks is its finer updates in cwnd size and its proactive nature of handling incipient congestion in the network, which should intuitively result in a higher goodput for UL-TCP accompanied with a lower packet loss. Figures 33 and 34 show the goodput and packet loss for these three protocols with varying hop lengths over a chain topology. Here, the nodes are static and the loss probability at each node is fixed at 10^{-4}. For all hop lengths, UL-TCP outperforms the other two protocols. It shows up to 95% higher goodput and up to 75% lower packet loss than TCP-Newreno. Note that unlike in single-hop networks, UL-TCP achieves a significant reduction in packet loss over TCP-Newreno. Learning-TCP shows up to 30% higher goodput, however at higher hop lengths its goodput approaches that of TCP-Newreno.

Since the nodes are mobile in ad hoc networks, here we study the performance of UL-TCP in the presence of node mobility. We take a random topology in a terrain size of 500 m × 1,500 m and take 75 nodes that are randomly distributed in this area. Mobility model used is the random-

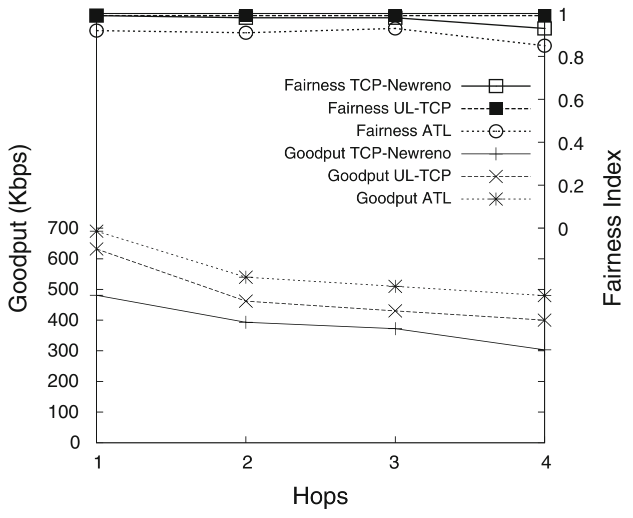


Fig. 32 Fairness and goodput over a multi-hop wired network with last-hop satellite link

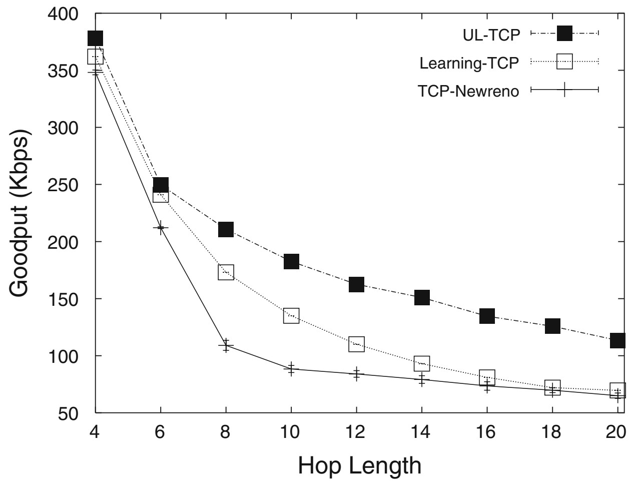


Fig. 33 Single flow: goodput for varying hop length in a static ad hoc network

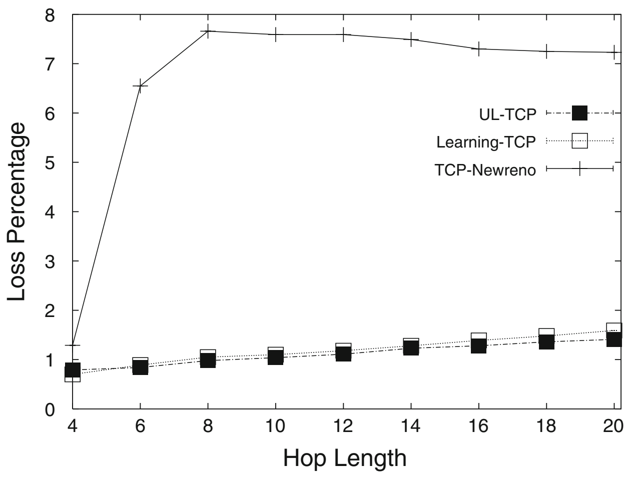


Fig. 34 Single flow: packet loss percentage for varying hop length in a static ad hoc network

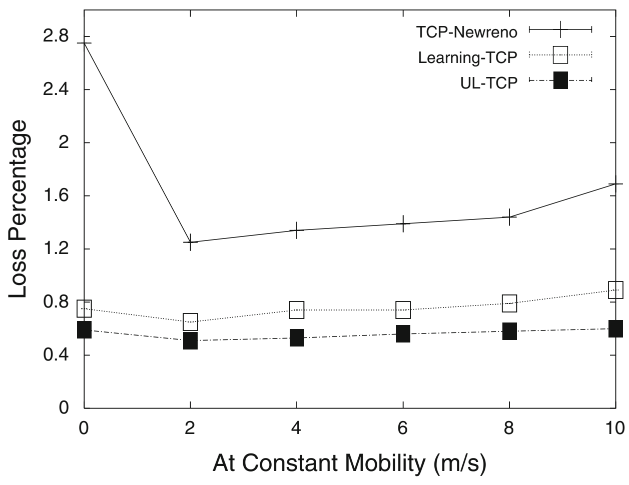


Fig. 36 Average packet loss percentage of 20 flows for varying mobility in an ad hoc network

way point model with a pause time of 0 s. We conducted this study for various mobilities, 0, 2, 4, 6, 8, and 10 m/s in the presence of 20 flows starting at a random time chosen from [0, 5] seconds time interval. Each flow runs for 600 s before it terminates. Figure 35 shows the cumulative goodput of these 20 flows. Here also UL-TCP is the best followed by Learning-TCP and TCP-Newreno. The improvement in the goodput for UL-TCP and Learning-TCP over TCP-Newreno are up to 23% and up to 4%, respectively. Figure 36 shows the corresponding packet loss percentage, where in we can observe that UL-TCP and Learning-TCP show a significant reduction in packet loss of up to 78 and 70%, respectively, over TCP-Newreno.

Our next study is related to the performance of UL-TCP for varying amount of load in the network. We take a static 11×11 grid topology and a fixed loss probability of 10^{-4} at each node. On this 11×11 grid, we vary the number of

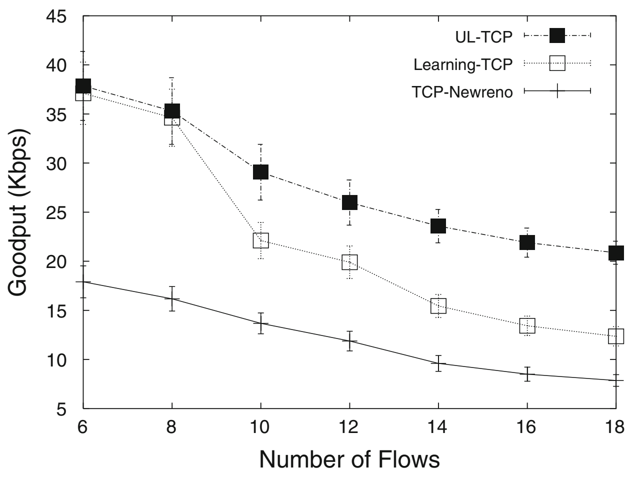


Fig. 37 Average goodput of the flows in an 11×11 grid ad hoc network for varying number of flows

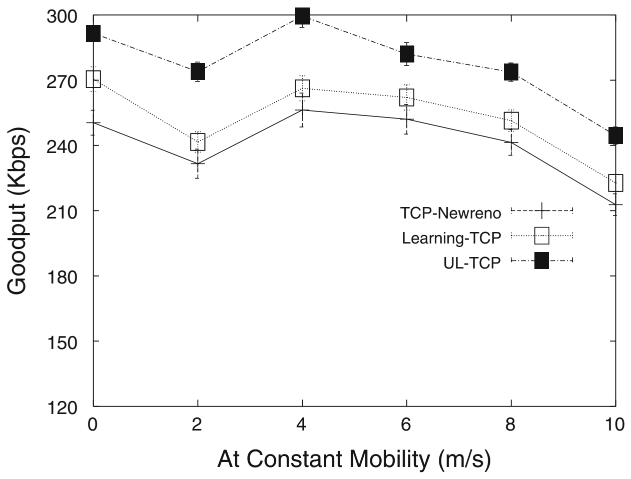


Fig. 35 Cumulative goodput of 20 flows for varying mobility in an ad hoc network

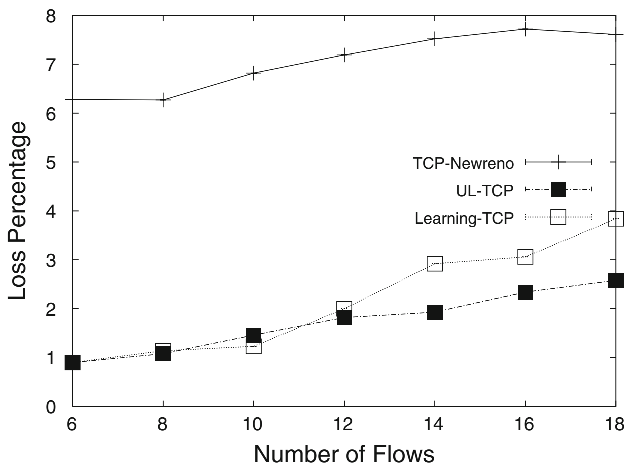


Fig. 38 Average packet loss percentage of the flows in an 11×11 grid ad hoc network for varying number of flows

flows from 6 to 18, where each flow runs from one end to the other end of the grid (i.e., 10 hops). This is an extremely congested network. Figures 37 and 38 show the average goodput and packet loss percentage for varying number of flows. UL-TCP stands best among the three by showing almost *three times* higher goodput and up to 80% reduction in packet loss over TCP-Newreno. Learning-TCP performs same as UL-TCP at lighter loads, however as we increase the load, its performance degrades drastically.

6 Conclusions and future work

In this paper, we proposed UL-TCP, a unified reliable data transport protocol for heterogeneous wireless networks. It does not rely on any explicit network support and seek any changes in *network components*, thus enabling its easier deployment. It can operate aggressively in high BDP networks, whereas in low BDP networks it can act conservatively. When the incipient congestion is detected, it reduces cwnd proactively to avoid multiple losses and timeouts, thereby conserving battery power and bandwidth in the network. Through extensive simulations, we showed that UL-TCP achieves significantly higher goodput over TCP-Newreno with a marginal increase in packet loss in single-hop wireless networks. In ad hoc networks UL-TCP shows a significant gain in goodput while achieving a significant reduction in packet loss over TCP-Newreno. We also compared UL-TCP against the Sack enabled ATL [19], which was proposed for single-hop wireless networks and observed significantly higher goodput for ATL over TCP-Newreno. However, this improvement is achieved at the cost of a very high packet loss and up to 90% unfairness to the competing TCP-Newreno flows which are certainly unacceptable. When UL-TCP is compared with ATL, UL-TCP shows a significant gain in terms of reduction in packet loss and improvement in fairness while not losing much of its goodput.

While in this work we evaluated UL-TCP through extensive simulations, in future, we plan to implement UL-TCP in Linux kernel and evaluate its performance over a real test-bed. We also plan to derive the values of parameters used in UL-TCP through theoretical analysis.

Acknowledgments The authors would like 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. Tian, Y., Xu, K., & Ansari, N. (2005). TCP in wireless environments: Problems and solutions. *IEEE Communications Magazine, 43*(3), S27–S32.
2. Gurtov, A., & Floyd, S. (2004). Modeling wireless links for transport protocols. *ACM Sigcomm Computer Communication Review, 34*(2), 85–96.
3. Barakat, C., Altman, E., & Dabbous, W. (2000). On TCP performance in a heterogeneous network: A survey. *IEEE Communication Magazine, 38*(1), 40–46.
4. Akyildiz, I. F., Zhang, X., & Fang, J. (2002). TCP-peach+: Enhancement of TCP-peach for satellite IP networks. *IEEE Communication Letters, 6*(7): 303–305.
5. Katabi, D., Handley, M., & Rohrs, C. (2002). Congestion control for high bandwidth-delay product networks. In *Proceedings of ACM Sigcomm* (pp. 89–102).
6. Jain, A. K., & Floyd, S. (2002). Quick-start for TCP and IP. In *Internet draft draft-amit-quick-start-01.txt, IETF*.
7. Tan, K., Song, J., & Zhang, Q. (2005). A compound TCP approach for high-speed and long distance networks. In *Proceedings of ACM Mobihoc* (pp. 288–299).
8. Sinha, P., Nandagopal, T., Venkitaraman, N., Sivakumar, R., & Bharghavan, V. (1999). WTCP: A reliable transport protocol for wireless wide-area networks. In *Proceedings of ACM Mobicom* (pp. 231–241).
9. Goff, T., Moronskim, J., & Phatak, D. S. (2000). Freeze-TCP: A true end-to-end TCP enhancement mechanism for mobile environments. In *Proceedings of IEEE infocom* (pp. 1537–1547).
10. Mondal, S. A. (2009). Improving performance of TCP over mobile wireless networks. *Wireless Networks Journal, 15*(3), 331–340.
11. Balakrishnan, H., Seshan, S., Amir, E., & Katz, R. H. (1995). Improving TCP/IP performance over wireless networks. In *Proceedings of ACM Mobicom* (pp. 2–11).
12. Barked, A., & Badrinath, B. R. (1995). I-TCP: Indirect TCP for mobile hosts. In *Proceedings of IEEE ICDCS* (pp. 136–143).
13. Bhandarkar, S., Sadry, N. E., Reddy, A. L. N., & Vaidya, N. H. (2005). TCP-DCR: A novel protocol for tolerating wireless channel errors. *IEEE Transactions on Mobile Computing, 4*(5), 517–529.
14. ElRakabawy, S. M., Klemn, A., & Lindemann, C. (2005). TCP with adaptive pacing for multihop wireless networks. In *Proceedings of ACM Mobihoc* (pp. 288–299).
15. Nahm, K., Helmy, A., & Kuo, C. J. (2005). TCP over Multihop 802.11 networks: Issues and performance enhancement. In *Proceedings of ACM Mobihoc* (pp. 277–287).
16. Venkata Ramana, B., Manoj, B. S., & Murthy, C. S. R. (2005). Learning-TCP: A novel learning automata based reliable transport protocol for ad hoc wireless networks. In *Proceedings of IEEE Broadnets* (pp. 521–530).
17. Xu, K., Tian, Y., & Ansari, N. (1995). Improving TCP performance in integrated wireless communications networks. In *Proceedings of IEEE ICDCS* (pp. 136–143).
18. Mascolo, S., Casetti, C., Gerla, M., Snadidi, M., & Wang, R. (2001). TCP westwood: Bandwidth estimation for enhanced transport over wireless links. In *Proceedings of ACM Mobicom* (pp. 287–297).
19. Akan, O. B., & Akyildiz, I. F. (2004). ATL: An adaptive transport layer suite for next-generation wireless internet. *IEEE Journal on Selected Areas in Communications, 22*(5), 802–817.
20. Yang, Y. R., & Lam, S. S. (2000). General AIMD congestion control. In *Proceedings of IEEE ICNP* (pp. 187–198).
21. Narendra, K. S., & Thathachar, M. A. L. (1989). *Learning automata: An introduction*. New Jersey: Prentice Hall.
22. Thathachar, M. A. L., & Sastry, P. S. (2004). *Networks of learning automata: Techniques for online stochastic optimization*. Kluwer: New Jersey.
23. Brakmo, L., & Peterson, L. (1995). TCP vegas: End-to-end congestion avoidance on global internet. In *IEEE Journal on Selected Areas in Communications, 13*(8), 1465–1480.

24. Samaraweera, N. K. G. (1999). Non-congestion packet loss detection for TCP error recovery using wireless links. *Proceedings of IEE Communications, 146*(4), 222–230.
25. Benveniste, A., Metivier, M., & Priouret, P. (1987). *Adaptive algorithms and stochastic approximations*. New York: Springer.
26. Kushner, H. J., & Yin, G. G. (1997). *Stochastic approximation algorithms and applications*. New York: Springer.
27. Chandrayana, K., Ramakrishnan, S., Sikdar, B., Kalyanaraman, S., Balan, A., & Tickoo, O. (2006). On randomizing the sending times in TCP and other window based algorithms. *Computer Networks Journal, 50*(3), 422–447.

Author Biographies

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 August 2006 – May 2007, he was a project officer at IIT, Madras, India. He is currently working as a research fellow at the Hamilton Institute, National University of Ireland Maynooth, Ireland. 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.