ELSEVIER

Computer Communications xx (2002) xxx–xxx

computer communications

www.elsevier.com/locate/comcom

Exploiting the adaptive properties of a probing device for TCP in heterogenous networks

V. Tsaoussidis*, A. Lahanas

College of Computer Science, Northeastern University, Boston, MA 02115, USA

Received 26 November 2001; revised 2 May 2002; accepted 2 May 2002

Abstract

End-to-end protocols lack the functionality to efficiently adjust their error control strategies to the distinct characteristics of network environments and to specific constraints of communicating devices. In the context of heterogeneous networks, error control needs to be responsive to the nature of the errors spanning a conservative-through-to-aggressive behavioral spectrum. In the context of mobile, battery-powered devices, the recovery strategy should yield good performance with a minimal transmission effort. We exploit the potential of a probing device to implement an adaptive error control strategy efficiently by shaping data transmission to be assorted with the distinctive characteristics of the underlying wired or wireless network components. We graft our mechanism onto standard TCP; we present encouraging experimental results with wired, wireless and mobile networks. © 2002 Elsevier Science B.V. All rights reserved.

Keywords: Heterogeneous networks; TCP-probing; Adaptive properties; Wired/wireless networks; energy savings; Transmission Control Protocol

1. Introduction

Computer networking is rapidly evolving towards a physically heterogeneous but functionally integrated environment consisting of both wired and wireless components. Traditionally, throughput efficiency has been a central concern of reliable transport protocols. The increasing presence of battery-powered devices in today's networking environment makes energy efficiency an additional focus of concern.

By and large, end-to-end protocols such as TCP [20] have evolved with a focus on throughput efficiency in wired (vs wireless) environments. Consequently, the nature of the errors of wireless links was not considered in the recovery strategy (e.g. congestion vs random/burst/fading-channel errors). Jacobson [16] was the first to study the impact of retransmission on throughput, based on experiments with congested wired networks. More recently, others have also been devoting attention to TCP throughput and proposing modifications in order to enhance its performance. Floyd and Henderson, for example, have shown that TCP throughput in wireless environments can be made to improve using partial acknowledgments and fast recovery [15]. On the other hand, Balakrishnan et al. [7], Lakshman and Madhow [19], and others [11,18], have shown that TCP throughput degrades when the protocol is used over satellite or wireless links. The authors in Ref. [8] have proposed a caching mechanism for acknowledgments at the base station which intervenes between a fixed and a mobile host. This link-layer solution advances TCP throughput in such environments (between a mobile and a fixed host). Other proposals to improve the throughput efficiency of TCP over wireless networks include [5,6,10,21]. In Ref. [5] the transport protocol is split at the wireline–wireless border. The base station maintains the connection details although TCP semantics are not maintained. MTCP presents a similar approach with some modifications at the acknowledgment strategy. Authors in (RFC 2018/2883) discuss potential TCP throughput improvements when multiple losses occur within a single window of data, based on modifications of the current acknowledgment strategy. TCP with selective acknowledgments (SACKs) can be beneficial for the protocol discussed here, and vice versa. SACK's contribution is focused on the acknowledgment strategy and not on the sender's adjustments[1] and decisions[2]. It has been shown in Ref. [24] that, in cases of multiple losses within a

* Corresponding author. Tel.: +1-617-373-8169; fax: +1-617-373-5121.
E-mail address: vassilis@ccs.neu.edu (V. Tsaoussidis).

[1] TCP-SACK has no mechanism to distinguish the cause of packet drops in order to adjust the size of the sending window accordingly.

[2] TCP-SACK mechanism is decoupled from the congestion window behavior [13], which is the point of interest in the present work.

PII: S0140-3664(02)00134-2

single window of data, a conservative strategy could be more efficient. The authors in Refs. [24,25] conclude that an adaptive strategy can yield better energy and throughput performance, based on analytical and experimental results, respectively.

The direction in which today's network environments are evolving raises some critical issues:

- Today's applications are expected to run in physically heterogeneous environments composed of *both* wired and wireless components. The existing TCP mechanisms do not satisfy the need for *universal* functionality in such environments. A significant missing component from TCP is the mechanism to distinguish the nature of the error in heterogeneous wired/wireless networks and to flexibly adjust the recovery strategy.
- The motivating force driving most modifications ignores energy efficiency, which is becoming a key performance issue.

The related work discussed above raises the question 'where is the right place to add the required functionality of error control?'. This question does not have a clear answer nor is the contribution of the present work strictly confined within the context of transport protocols. Error control is not exclusively a management property of the router or the base station, nor is it exclusively assigned to the transport layer. A widely accepted approach, presented by the end-to-end argument [22], states that we can only implement a function at a lower layer, if that layer can perform the complete function. Since the lower level cannot have enough information about the applications' requirements and protocol parameters, it cannot implement the whole function of error control; it can only be used to optimize the function of the higher layer. There are two salient points to make: First, TCP does not have error control but congestion control; hence it does not have the required functionality already in place. Second, the proposed optimizations frequently exhibit a conflicting behavior with TCP's mechanisms; that is, a retransmission attempt at a lower level might result in timeout at the transport level. Furthermore, we cannot assume that all network devices or link-layer related modules would have the required functionality to support a defective TCP. That is, the transport layer seems to be a natural place for implementing the core functionality for error control. This does not exclude potential optimizations based on the more precise or accurate information that could be gathered from lower layers.

In this paper, we initially extend the well-known detecting property of probing with an adaptive cyclic scheme. In this scheme, a 'Probe Cycle' consists of a structured exchange of 'probe' segments between the sender and receiver that monitor network conditions. The cycle might be completed successfully when network conditions permit or otherwise be extended. Because of this additional property of the probing mechanism, we refer to it as a 'Probing Device'. We grafted this device onto standard TCP operating over wired, wireless, and mobile networks. We show that error control can be enhanced; error detection is more accurate and error-recovery is more responsive. Indeed, the properties of probing were combined with appropriate recovery tactics. We introduce here the 'Immediate Recovery' and 'Gentle Recovery' tactics that are activated upon error-free conditions depending on the level of present contention, to implement a weighted aggressive or a conservative strategy, respectively.

The remaining paper is organized as follows. In Section 2, we review the mechanisms of TCP from the perspective of the energy and throughput efficiency. The design of our Probing Device and the implementation of TCP-probing are detailed in Section 3. In Section 4 we outline our testing environment and methodology, and we discuss evaluation criteria, performance metrics and parameters of importance. We present our results along with an analysis of the protocols' behavior in Section 5. In Section 6, we briefly discuss some open issues for further research. Finally, we present some concluding remarks in Section 7.

2. Energy and throughput efficiency of transport protocols

Energy expenditure is device-, operation-, and application-specific. It can only be measured with precision on a specific device, running each protocol version separately, and reporting the battery power consumed per version. Energy is consumed at different rates during various stages of communication, and the amount of expenditure depends on the current operation. For example, the transmitter/receiver expend different amounts of energy when they, respectively, are sending or receiving segments, are waiting for segments to arrive, and are idle. Hence, estimation of energy expenditure, additional to that consumed by transmitting the original data, which is common to all protocols, cannot be based solely on the byte overhead due to retransmission, since overall connection time might have been extended while waiting or while attempting only a moderate rate of transmission. On the other hand, the estimation cannot be based solely on the overall connection time either, since the distinct operations performed during that time (e.g. transmission vs idle) consume different levels of energy. Nevertheless, a protocol's potential for energy saving may at least be gauged from the combination of byte overhead and time savings achieved. Indeed, if we consider that several devices are not optimized to adjust the power of the transmitting/receiving module whenever the device does not actually transmits data, and also that TCP's operation does not involve any 'idle' state by default[3] communication time seems to be the most significant factor. Determination

[3] Authors in Ref. [9] show that this is feasible in some cases.

of the energy function which is appropriate to describe TCP's operation on specific devices is an ongoing research work of the authors.

Though the energy-conserving capability of transport protocols can play an important role in determining the operational lifetime of battery-powered devices, the subject has not been adequately studied in the literature.

The key to throughput and energy efficiency in reliable transport protocols is the error control mechanism. TCP error control does have some energy-conserving capabilities: in response to segment drops, it reduces its window size and therefore conserves transmission effort. The aim here is not only to alleviate congested switches, but also to avoid unnecessary retransmission that degrades the protocol's performance.

Tahoe and Reno are two common reference implementations for TCP. New Reno and SACK are modified versions of Reno that attempt to solve some of Reno's performance problems when multiple packets are dropped from a single window of data, by introducing an enhanced acknowledgement strategy. The congestion control algorithm in Tahoe includes Slow Start, Congestion Avoidance, and Fast Retransmit [4]. It also implements an RTT-based estimation of the retransmission timeout. In the fast retransmit mechanism, a number of successive (the threshold is usually set at three), duplicate acknowledgments (*dacks*) carrying the same sequence number triggers off a retransmission without waiting for the associated timeout event to occur. The window adjustment strategy for this 'early timeout' is the same as for a regular timeout: Slow Start is applied. The problem, however, is that Slow Start is not always efficient, especially if the error was purely transient or random in nature, and not persistent. In such a case, the shrinkage of the congestion window is, in fact, unnecessary, and renders the protocol unable to fully utilize the available bandwidth of the communication channel during the subsequent phase of window re-expansion. Reno introduces Fast Recovery in conjunction with Fast Retransmit. The idea behind Fast Recovery is that by receiving a threshold number of dacks the sender retransmits the missing segment, and then, instead of entering Slow Start as in Tahoe, halves cwnd, sets the congestion threshold to cwnd, and resets the dack counter. In Fast Recovery, the congestion window is thus effectively set to half its previous value in the presence of three dacks.

When network conditions deteriorate to an extent that they become the ground for more-or-less persistent error conditions (e.g. congestion, prolonged burst errors, fading channel), this kind of back-off strategy seems to be the correct choice. The error-recovery mechanism, however, is not always efficient, especially when the error pattern changes, since packet loss is invariably interpreted by the protocol as resulting from congestion. For example, when relatively infrequent random or short burst errors occur, the sender backs off and then applies a graduated increase to its reduced window size. During this phase of window expansion, opportunities for error-free transmission are wasted and communication time is extended. In other words, in the presence of infrequent and transient errors, TCP back-off strategy, at best, avoids only minor retransmission at the cost of unnecessary and significantly degraded goodput, and increases in overall connection time. Conditions of random and short or infrequent burst errors might actually call for an aggressive behavior instead, which could enhance throughput and reduce overall connection time in a combined dynamic that produces energy saving.

3. A probing device for TCP

A probing device models two properties: (i) it inspects the network load whenever an error is detected and enables error classification (i.e. congestion or not), (ii) it suspends data transmission for as long as the error persists, thereby forcing the sender to adapt its data transmission rate to the actual conditions of the channel. Using the TCP paradigm, we show that such a device could shape a transmission strategy that conforms to the load and error characteristics of the network. We call our version of TCP with probing 'TCP-probing'.

When a data segment is unduly delayed and possibly lost, the sender, rather than immediately retransmitting the segment and adjusting the congestion window and threshold downward, suspends data transmission and initiates a probe cycle instead. Since probe segments are composed of only segment headers, this enables the sender to efficiently monitor the network on an end-to-end basis, at much less cost in transmission effort (and hence energy cost) than would otherwise be expended on the (re)transmission of data segments that might not have a good chance of getting through during periods of degraded network conditions. It also contributes more effectively to alleviating congestion, should that happen to be the cause of the error, than would retransmitting a full data segment. The probe cycle terminates when network conditions have improved sufficiently that the sender can make two successive round-trip-time (RTT) measurements from the network, at which point it will have more information on which to base its error-correction response than does 'standard' TCP. In the event that persistent error conditions are detected, the sender backs off by adjusting the congestion window and threshold downwards as would standard TCP. On the other hand, if the conditions detected indicate only a transient random error that did not impact the network's effective throughput capacity, the sender could immediately resume transmission at a level that makes appropriate use of available network bandwidth.

In the event of persistent error conditions (e.g. congestion, burst error, handoff), the duration of the probe cycle will be naturally extended and is likely to be commensurate with that of the error condition, since probe segments will be

lost. The data transmission process is thus effectively 'sitting out' these error conditions awaiting successful completion of the probe cycle. In the case of random loss, however, the probe cycle will complete much more quickly, in proportion to the prevailing density of occurrence for the random errors.

The sender enters a probe cycle when either of two situations apply:

1. A timeout event occurs. If network conditions detected when the probe cycle completes are sufficiently good, then instead of entering Slow Start, TCP-probing simply picks up from the point where the timeout event occurred (in fact, some adjustments in the window might take place—this scheme is explained in detail later). In other words, neither congestion window nor threshold is adjusted downwards. We call this 'Immediate Recovery'. Otherwise, Slow Start is entered.
2. Three dacks are received. Again, if prevailing network conditions at the end of the probe cycle are sufficiently good, Immediate Recovery is executed. Note that here, however, Immediate Recovery will also expand the congestion window in response to all dacks received by the time the probe cycle terminates. This is analogous to the window inflation phase of Fast Retransmit in Reno. Alternatively, if deteriorated network conditions are detected at the end of the probe cycle, the sender enters Slow Start. This is in marked distinction to Reno behavior at the end of Fast Retransmit. The logic here is that having sat out the error condition during the probe cycle and finding that network throughput is nevertheless still poor at the end of the cycle, a conservative transmission strategy is more clearly indicated.

3.1. Implementation

A probe cycle uses two probing segments (PROBE1, PROBE2) and their corresponding acknowledgments (PR1_ACK and PR2_ACK), implemented as option extensions to the TCP header. The segments carry no payload. The option header extension consists of fields: (i) *type*, in order to distinguish between the four probe segments (this is effectively the *option code* field); (i) (*options*) *length*; (iii) *id number*, used to identify an exchange of probe segments.

The sender initiates a probe cycle by transmitting a PROBE1 segment to which the receiver immediately responds with a PR1_ACK, upon receipt of which the sender transmits a PROBE2. The receiver acknowledges this second probing with a PR2_ACK and returns to the ESTAB state (see Fig. 1). The sender makes an RTT measurement based on the time delay between sending the PROBE1 and receiving the PR1_ACK, and another based on the exchange of PROBE2 and PR2_ACK.

The sender makes use of two timers during probing. The first is a *probe timer*, used to determine if a PROBE1 or its corresponding PR1_ACK segment are missing, and the same again for the PROBE2/PR2_ACK segments. The second is a *measurement timer*, used to measure each of the two RTTs from the probe cycle, in turn. The probe timer is set to the estimated RTT value current at the time the probe cycle is triggered.

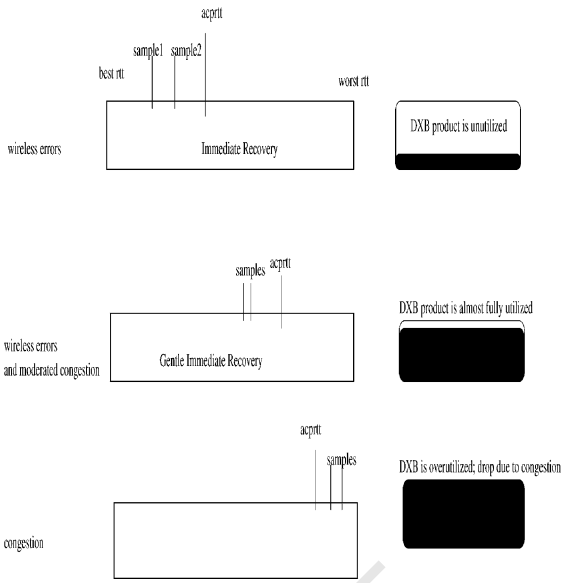

Fig. 1. TCP-probing recovery strategy.

The value in the option extension id number identifies a full exchange of PROBE1, PR1_ACK, PROBE2 and PR2_ACK segments, rather than individual segments within that exchange. Thus, in the event that the PROBE1 or its PR1_ACK is lost (i.e. the probe timer expires), the sender reinitializes the probe and measurement timers, and retransmits PROBE1 with a new id number. Similarly, if a PROBE2 or its PR2_ACK is lost, the sender reinitiates the exchange of probe segments from the beginning by retransmitting a PROBE1 with a new id number. A PR1_ACK carries the same id number as the corresponding PROBE1 that it is acknowledging; this is also the id number used by the subsequent PROBE2 and PR2_ACK segments. The receiver moves to the ESTAB state after sending the PR2_ACK that should terminate the probe cycle. In this state, and should the PR2_ACK be lost, the receiver would receive—instead of data segments—a retransmitted PROBE1 that is reinitiating the exchange of probe segments since the sender's probe timer, in the meantime, will have expired.

3.2. TCP-probing decision process

A critical part of the probing mechanism is the decision rules that determine action at the end of the probe cycle. Hence, a ruling is needed upon a packet drop to determine its cause (i.e. congestion or not). In case of drop due to congestion the protocol backs off as in Tahoe without violating the congestion control principles. In case of a random, transient error that is no longer affecting the

communication channel, the protocol resumes transmission with Immediate Recovery. In case of a burst error caused by some transmission deficiency or a handoff, the probe cycle will probably be extended. Then, data transmission will be suspended and a decision on the cause of the drops will not be made until conditions permit the probing cycle to be completed.

Immediate Recovery overcomes TCP's inefficiency to exploit the capacity by means of growing the window in the presence of frequent channel errors. Standard TCP would back off in the presence of such errors adjusting the slow start threshold downwards; depending on the frequency of the error and the *Delay* × *Bandwidth* (D × B) product, standard TCP might not be capable of exploiting the available bandwidth of a large D × B product. That is, having the slow start threshold adjusted to small values, additive increase will dominate the protocol's strategy quite early. Consequently, a potentially large number of RTT's will be needed to reach a full window size and, depending on the frequency of the errors, this might never happen with standard TCP. We demonstrate this case in more detail in Section 5.

It is possible to experience a random drop during a phase of moderated congestion. Although an Immediate recovery at the level of the previous window followed by Additive Increase would be the action that best matches TCP operation[4], TCP-probing takes advantage of the recent information gathered by the probe cycle and follows a 'Gentle Recovery'. That is, timeout and threshold values are not adjusted but the window size is taking a value from the range (half previous window, full previous window) and in reverse proportion to the level of detected congestion. Note that Gentle Recovery constitutes an improvement only in the presence of random errors that trigger the probing mechanism.

We implement the above plan as follows. Upon exiting the probe cycle, the two RTTs measured are compared. If the measured Probe RTTs (`mprtt`) are less than an *active threshold* that identifies the *Average Current Probe RTT* (`acprtt`), the protocol enters Immediate Recovery; the threshold used here is built as an active average that relies heavily on recent probe RTT samples. Otherwise the protocol enters Slow Start (like TCP Tahoe). If `mprtt` is less than `acprtt`, we consider the measurement successful and this helps us make the decision if a drop is due to congestion or not. We also compare the `acprtt` with the best and worst measurement of Probe RTT (`bestprtt`, `worstprtt`) (that is, the smallest and largest Probe RTT during the communication, respectively). The more the distance from the `bestprtt` grows the more conservatively we will recover. For example, if the active threshold (`acprtt`) goes closer to `bestprtt` we will use Immediate Recovery with full window size; else we will recover gently. For example, if the `mprtt` < `acprtt` > > `(worst + best)/2` which indicates that we are approaching the upper threshold of available capacity, we will recover with half window. Thus, we will not adjust the `timeout` and `ssthresh` but we will limit the window size during Immediate Recovery (in reverse proportion to the location of the `acprtt` in the space [bestprobertt, worstprobertt]. Note that only probes participate in the calculation of `acprtt`.

Even more sophisticated decision criteria can be developed. For example, the value of each of the two RTTs with respect to the active threshold, and the delay variation (jitter) between them, are potentially useful sources of information that we have not attempted to make use of[5].

```
acprtt = a * acprtt + b * sampleprobeRTT
/* (a = 0.2, b = 08) */
if (acprtt < (bestprobe + worstprobe)/
2)
then /* we are at the left edge of the
space */
if both samplertt < acprtt /* no conges-
tion is indicated */
cwnd = old cwnd; /* Immediate Recovery
*/
else /* Things are now worse than it was
previously */
Reno
if          acprtt > (bestprobertt +
worstprobertt)/2
then /* we are closer to the right edge of
the window in fig1 */
if (both samplertt < acprtt)
/* the error was not due to congestion */
/* Gentle Recovery */
cwnd = oldcongw/2 + [(worst-sample)/
(worst-best)] * old cwnd
/* We adjust the window in reverse pro-
portion to the available bandwidth */
/*  although the error was not due to
congestion */
else
Slow Start
```

Other issues need to be considered for the implementation of probing. For instance, the sender could receive *acks* during the probe cycle, in which case it updates its sending window as would standard TCP, but does not send out new data before the completion of the probe cycle. Of course, in a full duplex connection the sender might nevertheless need to respond with acks to the receiver's data. Duplicate acknowledgment delivery during a probe

[4] That is standard TCP would have kept growing its window under that level, if a drop was not detected.

[5] Note that the algorithm presented here, and hence the results are significantly improved from those reported in Ref. [23].

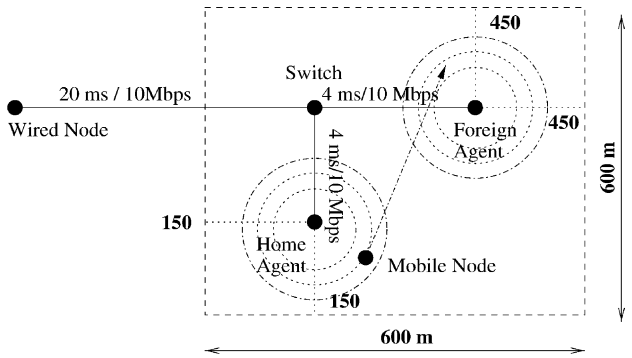

Fig. 2. Hand-off layout with NS-2 simulator.

cycle is another issue that needs to be addressed. When a probe cycle is triggered because of a timeout, dacks are ignored. For a probe cycle triggered by dacks, however, we keep count of the total number of dacks. This count is used to increase the congestion window size if the ensuing phase is Immediate Recovery. Therefore, TCP Probing respects the ACK-based clocking of TCP even during the probing phase.

Probing proves to be a more useful device than would be sending data that is likely to be dropped, on the one hand; or reducing the window size and degrading the connection throughput, possibly for no good reason, on the other. The first option would negatively impact energy expenditure. The second would needlessly degrade the goodput and also, by unnecessarily prolonging the connection time, impact energy consumption. However, grafting probing mechanism onto TCP does not retract the original additive increase multiplicative decrease (AIMD) fairness strategy [12], which is used whenever congestion is detected as the cause of packet drops. TCP-probing converges faster to maximum efficiency when errors are transient; it also adopts TCP fairness scheme since it does not over-utilize the overall available bandwidth but stays within the confines of the detected level of bandwidth. In the contrary, it assists fairness in terms of consuming the assigned bandwidth to the particular flow; under-utilization of the available bandwidth in such cases, causes the competing flows to gradually consume 'extra' bandwidth through additive increase.

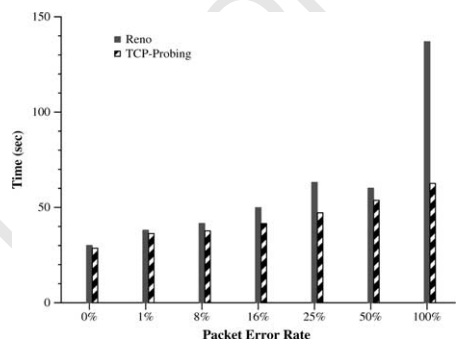

Fig. 3. Task completion time performance of TCP-probing and Reno (see Table A1).

4. Testing plan and methodology

4.1. Parameters

The versions of TCP discussed here were implemented as fully developed functioning protocol code using the X-kernel framework [1]. In order to capture the dynamics of real networks, tests were carried out over a real network, using the X-kernel platform. In order to observe the protocol behavior under different topologies or conditions (e.g. user speed) in a cellular network we carried out separate tests in a simulated environment using the network simulator (ns2). Tests were undertaken with the X-kernel using different sizes of data for transmission, up to 5-MB (5,242,880 bytes). However, the different sizes or experiment duration indicate similar relative results; the 5-MB data size was selected for presentation here since it was sufficiently large to avoid significant deviations of measurements. Most experiments reported here are well beyond the range of 95% confidence interval and within the $(-5\%, +5\%)$ space in reference to the means of the samples. However, the deviation appears to be more significant when the error rate is low. We outline this behavior in Fig. 13 and we discuss it further in Section 5. Note, that the purpose of the experiments was to investigate the *relative gains* of TCP-probing; accordingly, the data size needed to be selected so that protocol behavior could be efficiently and correctly investigated.

The TCP max sending window was set to 64. The number is selected to be sufficient to fill in a relatively large $D \times B$ product. Although we ran the experiments on low-throughput channels additional propagation and queuing delays were added during the experiments. The results we report are based on the average of several replications for each test. Standard deviation was our criterion for the number of replications.

4.2. Testing plan

The tests were carried out in three stages.

Initially in a single session, with the client and the server running on two directly-connected dedicated hosts, so as to avoid unpredictable conditions with distorting effects on the protocol's performance. Each version of the protocol was tested by itself, separately from the others, so that its error control mechanism can demonstrate its capability without being influenced by the presence of other flows in each channel. We simulated a error conditions of different intensity and duration in order to evaluate the protocols' performance in response to changes in that environment.

At the next stage, we have used an external application protocol to generate traffic in order to observe the protocols behavior in an environment with varying error and delay patterns. During these experiments drops were caused by congestion and/or by the simulated link deficiencies. This set of experiment allowed us for conclusions on the capabilities of TCP-probing to distinguish the nature of

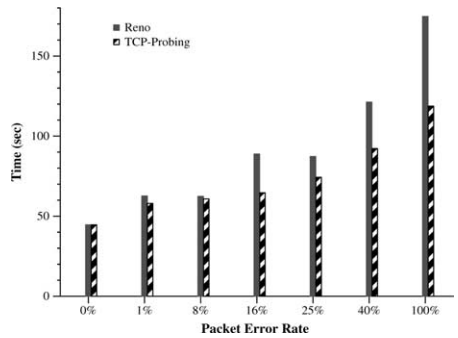

Fig. 4. Task completion time performance with 50 ms propagation delay (see Table A2).

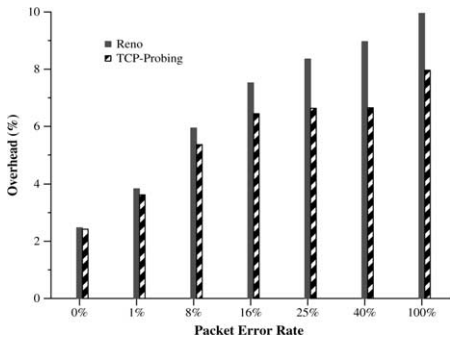

Fig. 6. Overhead of protocols with 100 ms propagation delay (see Table A3).

the error and to respond appropriately. More precisely, the aim here was to test the capability of TCP-probing to respond aggressively whenever the error rate permitted an aggressive behavior and more conservatively, whenever congestion was present, even if a detected drop was not due to congestion.

In order to simulate the wireless error conditions, we developed a new X-kernel 'virtual protocol' VLINK which was configured between TCP and IP. VLINK's core mechanism consists of a 2-state continuous-time Markov chain. Each state has a mean sojourn time m_i and a drop rate $r_i (i = 1, 2)$ whose values are set by the user. The drop rate r_i takes a value between 0 and 1, and determines the proportion of segments to be dropped during state i. Thus, when it visits state i, the mechanism remains there for an exponentially distributed amount of time with mean m_i, during which it randomly drops a proportion r_i of segments being transmitted, and then transits to the other state. Hence, when the channel leaves one state, it enters the other state with probability one. One state was always configured with a zero drop rate. Thus, simulated error conditions during a given experiment alternated between 'On' and 'Off' phases during which drop actions were in effect and were suspended, respectively. Error conditions of various intensity, persistence and duration could thus be simulated, depending on the choice of mean state-sojourn time and drop rate for the On state. In order to have wireless error patterns incorporated into the experiments (i.e. path loss, slow fading, fast fading, noise/interference from other networks/devices) we have selected a range of error rates that correspond to packet error rates (PER) of 1, 8, 16, 25, and 40%.

The states were configured to have equal sojourn times for the duration of the experiments with the exception of burst errors during fading. Here, we have used a similar protocol parameters based on the Gilbert model discussed in Ref. [2]. For this set of experiments we used a dropping rate of 1 for the 'On' Phase. The dropping rate during the Off phase was again 0. Following this pattern (i.e. all segments are dropped during the 'On' state) we have simulated heavy burst errors. We have also embedded a similar model in a separate application that was used to generate external TCP-based traffic. The application protocol alternates ON/OFF phases that generate and suspend data transmission. This way we were able to effectively combine congestion characteristics with link errors. Although error modeling of such conditions constitutes an active research area, we were able to eliminate results of dubious precision by combining the dynamics of a real network and the properties of our error model.

It is essential to consider that the experiments were carried out over a real network with one process running per peer. The VLINK at the receiver was dropping packets while the VLINK at the sender was dropping acknowledgments.

At the third stage a mobile network is presented and the protocol behavior is evaluated in the context of goodput and

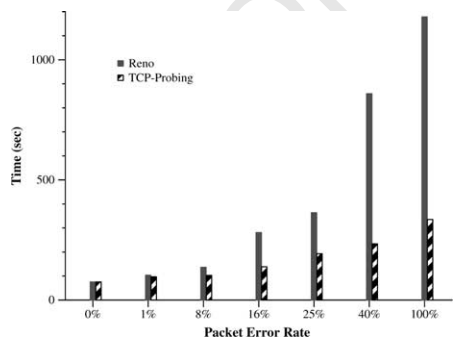

Fig. 5. Task completion time performance with 100 ms propagation delay (see Table A3).

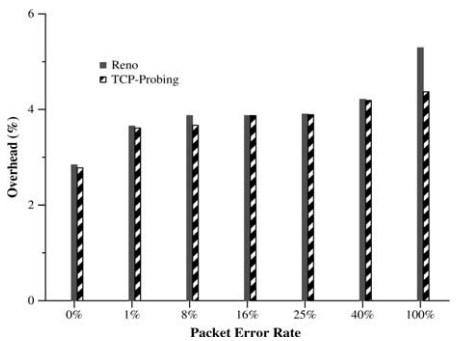

Fig. 7. Overhead of Reno and TCP-probing (see Table A1).

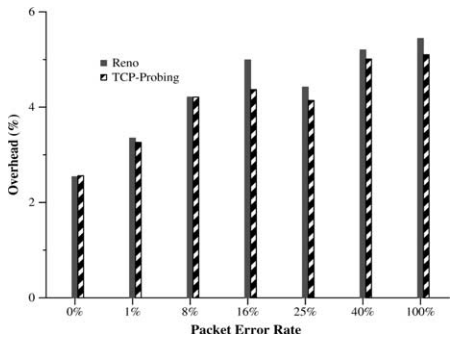

Fig. 8. Overhead of protocols with 50 ms propagation delay (see Table A2).

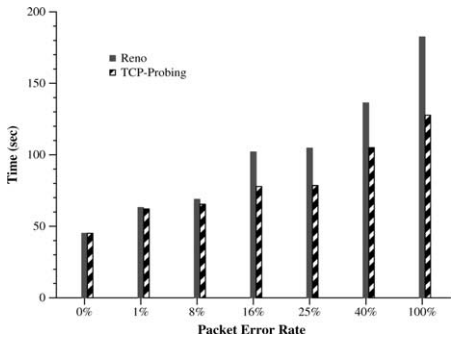

Fig. 10. Task completion time performance with 50 ms propagation delay and congestion (see Table A5).

overhead expenditure when the user speed or node distribution changes. NS-2 provides the Mobile-IP implementation of SUN Microsystems and an IEEE802.11 wireless LAN. With these modules, we build a simple wired-cum-wireless scenario as shown in Fig. 2. The link parameters (bandwidth and propagation delay between the wired node and the base-station) were chosen so that the delay bandwidth product equals 64K; the queue size at the router was set twice the delay bandwidth product. TCP-probing and TCP-Reno were configured on both end nodes. The buffer size was 120K and the time-out granularity was 100 ms. FTP was configured on top of each TCP and its task was to send continuously data for a period of 120 s; the deviation we observed with the aforementioned duration was within a 98% confidence and 5% precision. Note that randomness with the ns simulator was present mainly due to the random placement of the mobile device in the terrain and the point in time at which movement was initiated.

The wireless node was placed randomly in the terrain near the Home Agent in an area with radius 100 m (see Fig. 2). At randomly selected points of time (between the 10th and the 100th second) the mobile node starts moving with a constant speed towards a randomly selected destination in the Foreign Agent area. When the wireless node moves from the area of the Home Agent to that of the Foreign Agent a handoff occurs. To examine different handoff scenarios we experimented with several base station placements on the terrain and different moving speeds of the wireless node. In the experiments the base stations were placed 200, 300, and 400 m apart; the moving speed of the wireless node was set at 2, 4, 8, 16, and 20 m/s. The wireless nodes were configured with the Mobile-IP to enable re-routing of packets when the node moves from the domain of one base-station to another. Destination-sequenced distance vector (DSDV) was chosen as the routing protocol for the wireless portion of the network.

It is notable that probing or similar techniques (i.e. packet pair) in general is not a new concept and it has been proposed in the past as a mechanism to measure current network conditions [14]. Probing here is used with a combined property of self-adjusting phase-duration enabling an adaptive protocol strategy.

We compare TCP-probing with TCP Reno, since probing has been grafted onto TCP-Reno.

4.3. Performance metrics

We took measurements of the total connection time and of the total number of bytes transmitted (i.e. including protocol control overhead transmissions, data segment retransmission, etc.). Both factors significantly affect energy expenditure as well as throughput. Detailed results are presented in Tables A1–A9 in Appendix A.

The *Task completion time* is the time required to complete reliable transmission of 5MB under different conditions, from connection initiation through to connection termination. The Goodput performance of the protocols can be directly calculated using the formula: `Goodput = Original Data/Task Completion Time`. All

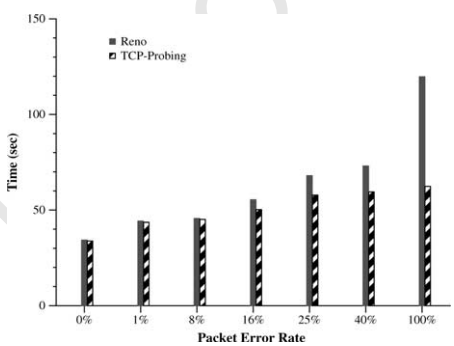

Fig. 9. Task completion time performance with congestion (see Table A4).

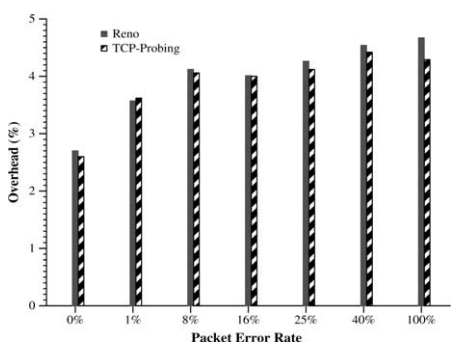

Fig. 11. Overhead of protocols with congestion (see Table A4).

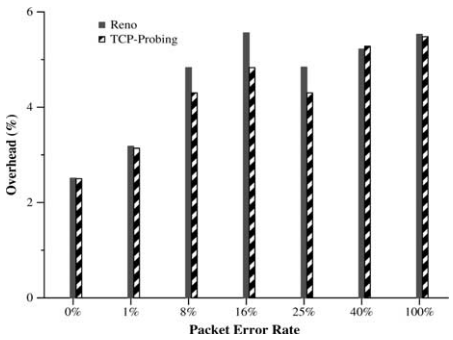

Fig. 12. Overhead of protocols with 50 ms propagation delay and congestion (see Table A5).

figures presented in Section 5 below are based on the data given in Tables A1–A9 in Appendix A.

We also use *Overhead* as a metric to realize the protocol transmission effort in order to achieve the presented time performance. Overhead is the total *extra* number of bytes the protocol transmits, expressed as a percentage, over and above the 5MB delivered to the application at the receiver, from connection initiation through to connection termination. The Overhead is thus given by the formula:

```
Overhead = 100 * (Total − Original)/5Mbytes
```

where,

- `Original Data` is the number of bytes delivered to the high-level protocol at the receiver. It is a fixed 5MB data set for all tests presented in Section 5.
- `Total` is the total of all bytes transmitted by the sender and receiver transport layers. This includes protocol control overhead, data segment retransmission, as well as the delivered data.

For each scenario with the netework simulator we measure the number of packets sent and received by each TCP (Probing and Reno). We present these measurements in terms of *Packet-Goodput* (Total Packets Received) and *Packet-Throughput* (total packets sent). The difference

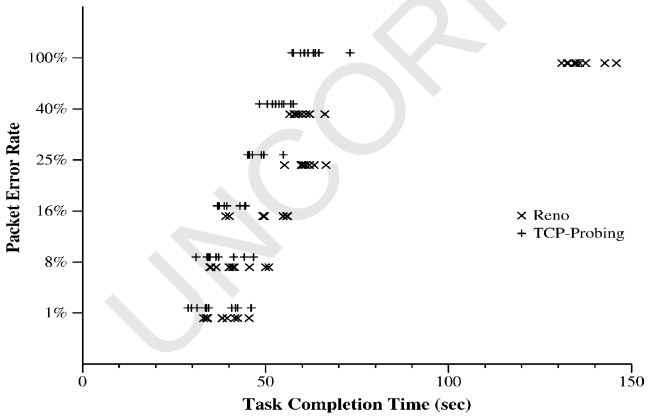

Fig. 13. Task completion time distribution of the protocols. The average value is presented in Table A1).

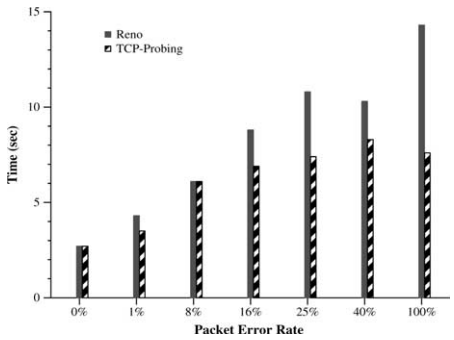

Fig. 14. Task completion time with small file transfer (300KB, see Table A6).

between the Packet-Throughput and Packet-Goodput gives the retransmission overhead of the protocol.

5. Results and discussion

5.1. Performance with one flow per test

Table A1 presents the measurements of the time performance of Reno and Probing. It can be observed that the higher the error rate, the more the relative improvement with TCP-probing. At small error rates (i.e. PER 1–8%) the protocols do not face significant obstacles but some performance difference is already indicated. Reno applies Fast Recovery instead of Slow Start (unlike Tahoe, for example) and Probing applies Immediate Recovery. The test does not involve any congestion other than the one created by the flow itself. The performance difference between Probing and Reno can be justified by the additional RTTs that are required by Reno to reach the max window size. At these error rates TCP-probing demonstrates an advantage in the range of 4.5–9.7% (see Table A1). It is instructive to consider how the probing and Immediate Recovery mechanisms provide TCP-probing with the behavioral flexibility underlying its performance in Fig. 3. At relatively low error rates (1–8%), it is able to expand its window in the Off phase. During the On phase its probing mechanism allows it to explore windows of opportunity for error-free transmissions, which are then exploited by Immediate

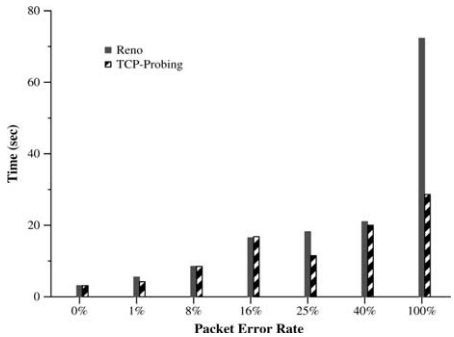

Fig. 15. Task completion time with small file transfer (300KB) and 50 ms propagation delay (see Table A7).

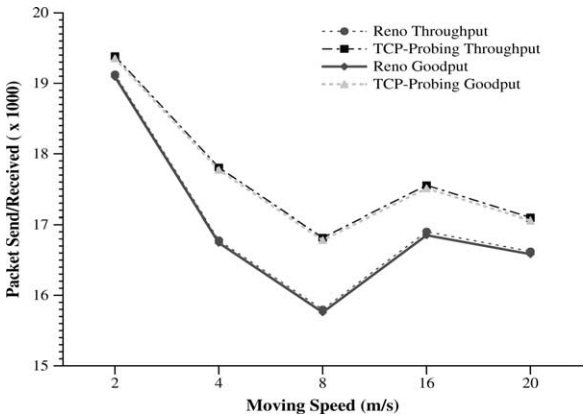

Fig. 16. Goodput vs Throughput. Base-stations distance: 200 m.

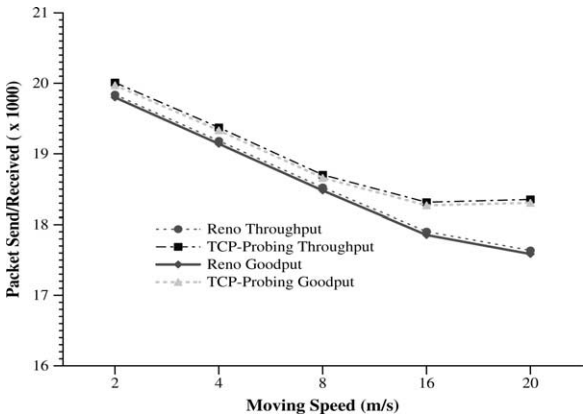

Fig. 18. Goodput vs Throughput. Base-stations distance: 400 m.

Recovery. It is effectively backing off for the duration of the probe cycle, but is also capable of rapid window adjustments with Immediate Recovery where appropriate. In contrast, Reno's mechanism is exclusively focused on congestion control. The idea is to alleviate congested routers and avoid flooding the network, so their mechanisms do not allow for rapid window adjustments as a recovery strategy after backing off. This is not necessarily the appropriate course of action in every instance of random and/or transient errors. Such errors are not always symptomatic of degraded network capacity, and so graduated adjustments could needlessly degrade overall throughput performance.

As the error rate increases consecutive Fast Recoveries are triggered; the `ssthresh` is reduced for Reno although congestion is not present. As indicated by the results at the error rate of 16 and 25% the relative performance difference between Reno and Probing grows in favor of Probing to a level of 16.8 and 25.1%, respectively. An extreme situation of 40% PER has also been tested. There, TCP-probing falls into consecutive probe cycles; it gains its slight advantage over Reno due to the avoidance of the unnecessary `ssthresh` adjustments.

The salient point to note here is that TCP-probing's goodput is uniformly no worse than Reno's, across the entire range of error rates. As such, TCP-probing yields an upper bound—sometimes a loose upper bound—to the standard version's goodput performance across the range of error rates. Recall that our On/Off error model simulates exponentially distributed On and Off phase duration with a specified mean value. During On periods, segments are randomly dropped at a specified rate. Thus, despite its stochastic nature, any single configuration of the model undoubtedly yields a narrower dynamic range of error patterns than would be encountered in a real network environment over a sufficiently long connection time. In other words, errors during a specific connection are more accurately exemplified by some pattern of varying configurations of our error model. Consequently, the fact that TCP-probing does not under-perform the best result of the standard version, and outperforms them for high error rates, across the range of configurations for the error model, gains further significance.

Someone might expect that the cost of probing is higher when the RTT duration increases.

In Figs. 4 and 5 we present results with the two protocols running on top of links with high propagation delay (the conclusion could be extended to Wide Area Networks). Notably, the relative performance gains of TCP-probing are now even higher. Indeed, the impact of delay is more significant on standard TCP since it requires more RTT's to

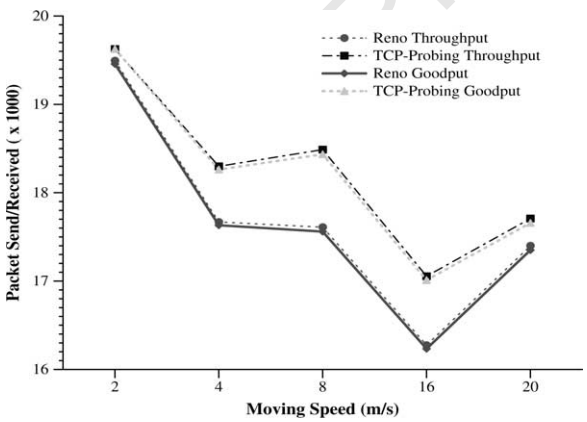

Fig. 17. Goodput vs Throughput. Base-stations distance: 300 m.

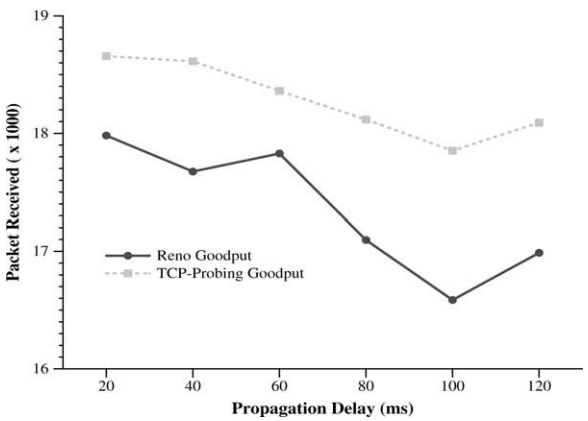

Fig. 19. Goodput vs Propagation delay. Moving speed 16 m/s.

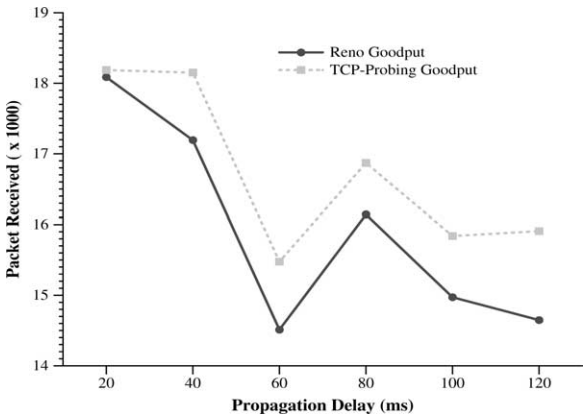

Fig. 20. Goodput vs Propagation delay. Moving speed 20 m/s.

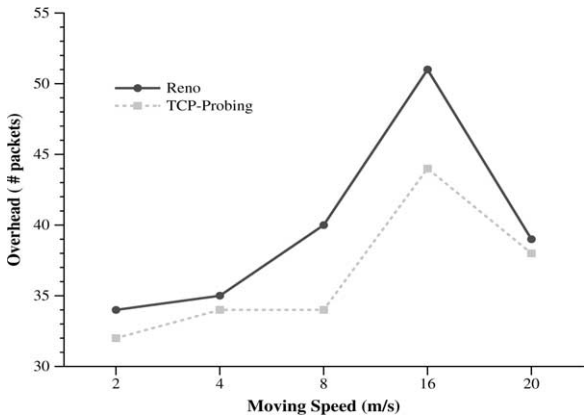

Fig. 21. Overhead: Simulator. Base-stations distance 200 m.

readjust to a full window. This situation is clearly indicated by the results presented in Fig. 4. We can observe that TCP-probing demonstrates an improvement of more than 50% at all high error rates and reaches up to a level of 72.7% for the packet error rate of 40%. It is important to note that TCP-probing yields these performance gains without applying a more aggressive behavior than Reno. This can be confirmed by the overhead of the two protocols as presented in Figs. 6–8. TCP-probing can be adjusted to produce even less overhead (i.e. adjusting the Immediate Recovery Strategy, extending the probe cycle, or adjusting the `acprtt` to trigger a more conservative recovery using a 'device factor') at the expense of time and depending on the particular device[6]. We observe from Figs. 6–8 that overhead with Probing is reduced. The improvement reaches a level of 6.3% at the 25% error rate with 50 ms propagation delay and grows even higher when the delay increases, as indicated by Fig. 8.

In conclusion, TCP-probing outmatches Reno in aggressive bias when an aggressive strategy yields better goodput and the detected drops are not due to congestion. Yet it manages to demonstrate this functionality using less transmission effort, which could be another significant factor for energy expenditure.

5.2. Protocol behavior in the presence of losses due to congestion and channel errors

Probing gradually induces a distinctly different pattern of behavior as error-frequency becomes more dense. The probe cycle naturally becomes more extended, during which no data segments at all are transmitted. Furthermore, in the event of congestion with deteriorating RTTs, the prolonged probe cycle should lead occasionally to Slow Start rather than Immediate Recovery.

Hence, an important design property of TCP-probing is its ability to distinguish the nature of the error. That is, to

[6] As noted in Section 2, for most devices, time is the most significant factor that determines the level of energy expenditure using TCP.

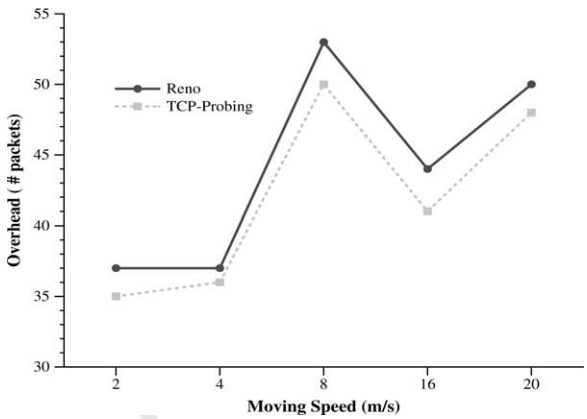

Fig. 22. Overhead: Simulator. Base-stations distance 300 m.

distinguish the drops due to congestion from the channel errors. This situation involves actually two interesting and distinct scenarios: (i) both errors (i.e. congestion vs channel) might happen separately but within the same connection phase or (ii) channel errors might happen in the presence of moderated congestion. The latter case, calls for a more conservative strategy which is implemented using the Gentle Recovery.

Fig. 9 is based on the measurements of the two protocols with mixed congestion and channel errors. Recall from

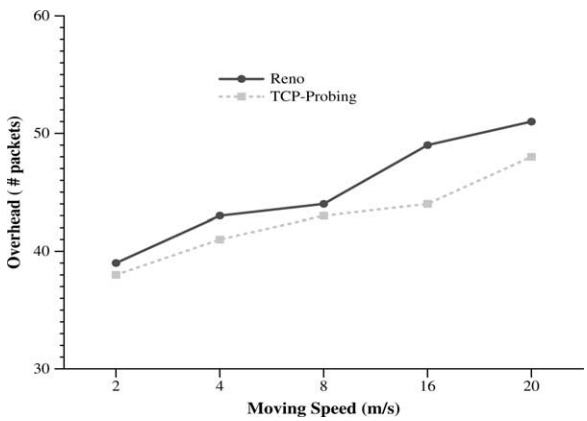

Fig. 23. Overhead: Simulator. Base-stations distance 400 m.

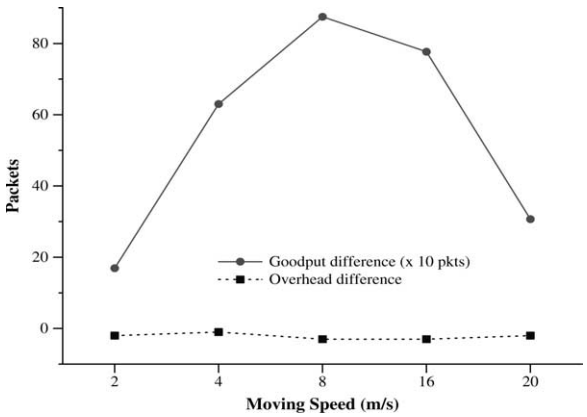

Fig. 24. Impact of handoff on goodput and overhead. Base-stations distance 300 m. *y*-Axis presents the difference between the goodput and overhead of TCP-probing and TCP-Reno.

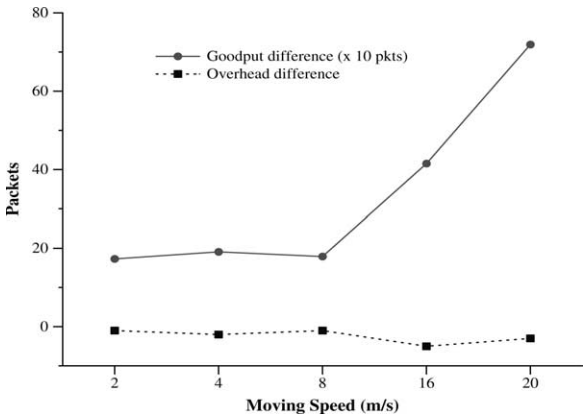

Fig. 25. Impact of handoff on goodput and overhead. Base-stations distance 400 m. *y*-Axis presents the difference between the goodput and overhead of TCP-probing and TCP-Reno.

Section 4.3 that congestion is created by a tcp-based application that alternates on/off phases of data transmission. Hence, during the experiment congestion was heavy and mitigated periodically. Due to the random nature of the error model, the protocols were phased with both the scenarios presented above. It appears from the results that the problem of TCP in such environments is twofold: first, when congestion is not present, standard TCP backs off unnecessarily as was indicated by the previous subsection[7]. Second, whenever congestion is present simultaneously with channel errors standard TCP experiences consecutive adjustments of the slow start threshold (`ssthresh`) too frequently and hence it does not have the opportunity to exploit the available bandwidth since several RTTs are required; probably the channel error and congestion drops could not permit a continuous window growth for that long.

This conclusion can be confirmed by the results presented in Fig. 10. Channels with packet error rates of more than 16% constitute an abrasive way for TCP Reno to exploit the available bandwidth. The performance gains of TCP-probing are comparable to those presented in Section 5.1 although the recovery now is frequently based on the Gentle Recovery described in Section 3.2. Recall that Gentle Recovery is activated whenever congestion is moderated and does not justify a packet drop; then, the threshold is not adjusted backwards and the timeout is not extended. Fig. 9 confirms that the impact of delay appears to be harsher on TCP Reno. A contrast of the Figs. 9 and 10 shows the *relative* improvement of TCP-probing over Reno at all error rates when the delay increases. Also note from Figs. 11 and 12 that TCP-probing does not yield better performance because it is more aggressive. Instead, it uses less retransmission overhead than Reno.

Fig. 13 shows some sample statistical details of the data we gathered. We observed that, although significant deviation was expected, in cases with high error rate where communication time is extended, not only the

[7] Whenever a drop is due to congestion both protocols back off similarly.

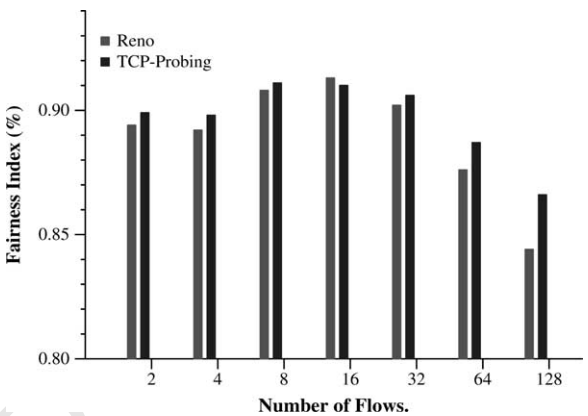

Fig. 26. Fairness over a 10 Mbps link.

average but also the median and the mode present similar evidence; this justifies the use of average for the presentation of data. Error rates with significant deviation have been repeated several more times. Tables A1–A9 in Appendix A incorporate these extra measurements over the regular ten runs per protocol and per error rate. We have mentioned in the introduction of Section 4 that we have tested the protocols with variable data sizes. The deviation of measurements was significant during the experiments although the results indicated similar behavior. It is expected that due to the random nature of the error model the impact of packet losses might be significant when it happens while the sending window is large. It can be

Table A1
Performance of Reno and TCP-probing

PER (%)	0	1	8	16	25	40
Protocol	*Task completion time (s)*					
Reno	30.0	38.0	41.6	49.9	63.1	60.1
TCP-probing	28.6	36.3	37.7	41.5	47.2	53.7
	Overhead (%)					
Reno	2.84	3.65	3.87	3.87	3.90	4.21
TCP-probing	2.78	3.61	3.67	3.87	3.89	4.19

Table A2
Performance of protocols with propagation delay 50 ms

PER (%)	0	1	8	16	25	40
Protocol	*Task completion time (s)*					
Reno	44.6	62.6	62.4	88.8	87.3	121.2
TCP-probing	44.6	58.0	60.8	64.6	74.3	92.2
	Overhead (%)					
Reno	2.54	3.35	4.21	4.99	4.42	5.20
TCP-probing	2.56	3.26	4.21	4.37	4.14	5.01

Table A3
Performance of protocols with propagation delay 100 ms

PER (%)	0	1	8	16	25	40
Protocol	*Task completion time (s)*					
Reno	75.4	103.2	136.3	281.1	363.4	859.1
TCP-probing	75.6	96.7	102.7	138.5	192.8	233.8
	Overhead (%)					
Reno	2.47	3.83	5.94	7.52	8.35	8.96
TCP-probing	2.42	3.62	5.36	6.44	6.63	6.65

Table A4
Performance of protocols with random congestion

PER (%)	0	1	8	16	25	40
Protocol	*Task completion time (s)*					
Reno	34.3	44.2	45.6	55.4	68.0	73.0
TCP-probing	33.8	43.6	45.1	50.2	57.8	59.5
	Overhead (%)					
Reno	2.70	3.57	4.12	4.01	4.26	4.54
TCP-probing	2.60	3.62	4.06	4.00	4.12	4.42

significantly different when the error 'hits' a small window; then the RTTs required for recovery are significantly less. Due to mechanism of TCP-probing to enter a two-RTT cycle whenever a loss is detected and to recover with Immediate Recovery whenever the RTT measurements indicate clear or improved network conditions the impact of randomness appears less significant on deviation. We present here (Figs. 14 and 15) a sample of our measurements with smaller data sizes (i.e. 300KB) for the purpose of completeness.

5.3. Protocol behavior in mobile environments

A common operation in mobile environments is the handoff process when a user moves across cells. Under this scenario the protocol's 3-DACK mechanism will likely[8] not be activated due to the absence of any correctly delivered segments. Consequently, the protocols do not recover with Fast Recovery but rather with slow start. For the duration of the handoff, Reno attempts to retransmit the missing segment and extends the timeout. By the time the handoff has been completed, the sender not only has not been able to recover from the loss, but indeed, it has degraded its ability to detect immediately the error-free channel and to adjust the congestion window rapidly upwards. With the enhancement of probing the protocol overcomes this deficiency: upon the detection of loss, it enters a probe cycle which cannot be completed for the entire duration of the handoff, hence saving on retransmission overhead. Furthermore, it adjusts the window rapidly with Immediate Recovery upon successful completion of the probe cycle. Since Reno attempts retransmission with single segment (which happens to be 1KB in our experiments) the overhead saving is not that significant for TCP-probing; however, the avoidance of an unreasonable graduated adjustment results in enhanced goodput performance. This conclusion is confirmed by the results presented in Fig. 16.

Figs. 16–18 display the goodput and the throughput of TCP-Reno and TCP-probing when the base-stations are placed 200, 300, and 400 m apart; the figures outline the protocols' relative efficiency when the wireless node moves with different speeds.

The experiments show that the distance between the base-stations does not have a major impact on the performance of the protocols but it can also not be considered insignificant. For the same moving speed, the performance of the protocols is slightly higher when the base-stations are placed 400 m apart rather than when they are placed 300 or 200 m apart. For example, at 400 m, the goodput of the protocols is about 4% higher than that achieved at 300 or 200 m.

In addition to the hand-off, the motion itself is associated with other phenomena such as the Rayleigh fading channel and the Doppler effect. The specific impact of Rayleigh fading channel and Doppler effect on the performance of standard TCP is analyzed in detail in Ref. [13]. As the speed of the node increases from 2 to 4 m/s the goodput of the protocols drop by 10% and when the speed increases to 20 m/s the goodput drops by 15% (see Figs. 19 and 20). The performance of both TCPs degrades, however, TCP-probing maintains its relative advantage achieving 3–6% higher goodput than Reno.

Figs. 19 and 20 present the protocols' goodput performance when propagation delay increases. The moving speed of the wireless node in each chart is constant and equal to 16 and 20 m/s, respectively. The goodput performance of the protocols degrades when the

Table A5
Performance of protocols with random congestion. Propagation delay 50 ms

PER (%)	0	1	8	16	25	40
Protocol	*Task completion time (s)*					
Reno	45.1	63.1	68.8	101.9	104.6	136.3
TCP-probing	45.1	62.1	65.4	77.8	78.5	105.0
	Overhead (%)					
Reno	2.51	3.18	4.83	5.56	4.84	5.22
TCP-probing	2.50	3.14	4.30	4.83	4.30	5.28

[8] This depends on the duration of the handoff and the size of the window.

Table A6
Performance of protocols with small file transfer (300KB) and congestion

PER (%)	0	1	8	16	25	40
Protocol	*Task completion time (s)*					
Reno	2.7	4.3	6.1	8.8	10.8	10.3
TCP-probing	2.7	3.5	6.1	6.9	7.4	8.3
	Overhead (%)					
Reno	2.14	3.55	10.15	14.45	15.33	16.02
TCP-probing	2.14	3.11	9.63	13.13	17.11	15.46

propagation delay increases. However, we observe in Fig. 19 that the performance of TCP-Reno is occasionally lower. In the results presented above one can see that the behavior of the protocols with regard to their goodput and overhead performance is non-monotonic. Although this looks as an anomaly or, from a statistical perspective, it may signal the need for further experiments, a close examination of the traces reveals that the non-monotonic behavior of goodput and overhead was due to TCP's behavior. More precisely, goodput is associated with the duration of the handoff, the duration and strength of 'fading', and the propagation delay. When propagation delay increases, the RTT increases and the timeout of TCP is extended (since it is a function of the RTT). Similarly, when the speed increases, the associated characteristics of handoffs and fading channels also change. For TCP this means that a handoff might cause one or more timeouts [11] and in turn, depending on the specifics of the mobile network, an extended RTT or a higher speed might cause less timeouts and better goodput. We have confirmed this phenomenon in an earlier work [23] and in a similar context: when error rate increases, goodput may increase. Our confidence regarding our reasoning, however, arises from the close examination of the traces.

Figs. 21–23 present the overhead, as a function of moving speed. It appears that the distance between the base stations doesn't affect the overhead of the protocols. Unlike the distance, the speed does have some impact on overhead which appears to increase as the speed of the wireless nodes increases. Furthermore, we observe that TCP-probing completes its task by expending less effort. The overhead of TCP-probing is from 6% (for 2 m/s) to 15% (at 8 m/s) lower than Reno's. Recalling the results of goodput, we conjecture that Reno's aggressiveness was not invested in goodput.

Figs. 24 and 25 show the impact of handoff delay on goodput and overhead. The results present the comparative difference of the goodput and overhead of TCP-probing and TCP-Reno; that is, the Figs. 24 and 25 show the extra number of packets (downscaled at a factor of 10) achieved by TCP-probing combined with the associated gain in overhead. The charts use Reno as a reference point for both goodput and overhead, which assumed to lie in the line $y = 0$. We can observe that Probing gains significantly in goodput and expends slightly less effort. Notice, for example, that the overhead line is below zero; one can reasonably expect that normalizing the results based on goodput, the difference in overhead would appear more significant.

Table A7
Performance of protocols with small file transfer (300KB) and congestion. Propagation delay 50 ms

PER (%)	0	1	8	16	25	40
Protocol	*Task completion time (s)*					
Reno	3.1	5.5	8.5	16.5	18.2	21.0
TCP-probing	3.1	4.2	8.5	16.8	11.5	20.0
	Overhead (%)					
Reno	2.13	2.90	6.71	10.89	12.74	15.66
TCP-probing	2.13	3.01	5.15	11.75	10.83	16.28

Table A8
Performance of protocols with handoffs (PER 15%)

Protocol	Congestion free		Congestion		Short transfer	
Prop. delay (ms)	20	50	20	50	20	50
Reno	136.9	174.7	119.7	182.4	14.3	72.3
TCP-probing	62.6	118.7	62.4	127.7	7.6	28.6
	Overhead (%)					
Reno	5.29	5.44	4.67	5.53	13.45	17.90
TCP-probing	4.37	5.10	4.29	5.48	12.16	17.05

6. Open issues

Probing represents a strategy wherein transmission effort and time are invested in probe cycles in order to determine the nature of prevailing error conditions. This 'cost of probing' is recouped and made to yield effective returns by adopting appropriately conservative and aggressive transmission tactics in response to the conditions detected. The results of 'Overhead' indicate that the decision-making component of the probing mechanism is amenable to localized, heuristic improvement. For example, when the protocol is applied to more 'sophisticated' devices that adjust the power consumption to the corresponding state of the transmission protocol(s)[9] the expenditure on overhead might become a more significant factor. Thus, TCP-probing could be adjusted to yield more savings on overhead at the expense of communication time. The adjustment mechanism appears to be already in place: The decision-making process includes an active average of the probe RTTs and determines the protocols recovery. That is, the `acprtt` can determine the protocol's conservative or aggressive behavior and trade off time versus overhead.

Fairness of TCP-probing is certainly an issue of interest. Fig. 26 outlines the fairness [17] of TCP-Reno and TCP-probing using a dumbbell topology with 10 Mbps link bottleneck. For a small number of flows the fairness index[10] shows similar performance.

[9] Whenever this is possible.
[10] Fairness index used was the standard index proposed in Ref. [17].

Table A9
Performance of protocols with handoffs of different PER

Prop. delay (ms)	20	30	50	20	30	50	20	30	50
Handoff PER (%)	5			10			15		
Protocol	*Task completion time (s)*								
Reno	31.9	52.1	60.2	57.0	73.2	153.9	136.9	156.2	174.7
TCP-probing	25.4	32.7	45.4	36.5	50.4	114.5	62.6	87.5	118.7
	Overhead (%)								
Reno	34.7	3.65	3.74	3.51	4.49	5.00	5.29	5.32	5.44
TCP-probing	3.40	3.49	3.58	3.47	4.11	4.74	4.37	4.87	5.10

Both protocols apply identical AIMD [12] mechanisms and fairness performance appears, in general, quite similar. When the number of flows increases, fairness of TCP-Reno appears to be smaller than the corresponding fairness of TCP-probing. The reason is that with multiple flows congestion is likely to happen more frequently and present a more persistent nature; probing appears to be an alleviating mechanism in its own right, triggering better performance. Although the fairness results are quite interesting and call for further attention, we did not elaborate further presently since fairness in wired networks is relevant but beyond our central focus in this paper.

Another interesting issue is the energy/throughput tradeoff as TCP-probing behavior scales down from a net aggressive bias at low error rates, where expenditure of extra transmission effort yields improved goodput, to a net conservative one at high rates, where goodput efficiency is attained by adjusting transmission rates downwards. At some 'intermediate' level of error rates aggressive tactical choices are counterbalanced by conservative ones, yielding a mix that displays neither conservative nor aggressive bias overall. The dynamics of the energy/throughput tradeoff need to be further investigated.

7. Conclusion

In today's heterogeneous wired/wireless internets, with proliferating battery-powered devices, TCP exhibits two shortcomings. It does not integrate energy efficiency as a focus of concern; and its error-recovery mechanism is not always efficient. Underlying both deficiencies is TCP's inability to distinguish between different types of errors and apply a flexible strategy for error-recovery. TCP-probing achieves energy and throughput efficiency by implementing a self-adjusting strategy, which is responsive to the nature of errors. Probing enables TCP to go beyond a circumscribed functionality exclusively focused on congestion control, and to move towards a *universal* error control with potential energy-conserving capabilities. The results presented in this paper serve to demonstrate the validity of the concept, and to provide directions for further research. Our experiments highlighted a property of 'probing devices' beyond the scope of TCP: to regulate data transmission in accordance with the protocol aggressive or conservative strategy in response to the distinctive error characteristics of networks with both wired and wireless components.

Appendix A

Tables A1–A9.

References

[1] The X-kernel, Technical report, www.cs.arizona.edu/xkernel.
[2] A.A. Abouzeid, S. Roy, M. Azizoglou, Stochastic modeling of TCP over lossy links, INFOCOM (2000).
[3] M. Allman, On the effective evaluation of TCP, ACM Computer Communication Review (1999) 34–39.
[4] M. Allman, V. Paxson, W. Stevens, TCP congestion control, RFC 2581, 1999.
[5] A. Bakre, B. Badrinath, I-TCP: Indirect TCP for Mobile Hosts, Proceedings of the IEEE ICDCS'95, 1995, pp. 136–143.
[6] A. Bakre, B. Badrinath, Implementation and performance evaluation of indirect TCP, IEEE Transactions on Computers 46 (3) (1997) 260–278.
[7] H. Balakrishnan, V. Padmanabhan, S. Seshan, R. Katz, A comparisons of mechanisms for improving TCP performance over wireless links, ACM/IEEE Transactions on Networking 5 (6) (1997) 756–769.
[8] H. Balakrishnan, S. Seshan, E. Amir, R. Katz, Improving TCP/IP Performance Over Wireless Networks, Proceedings of the First ACM International Conference On Mobile Computing and Networking (Mobicom), 1995, November, 1995; pp. 98–104.
[9] I. Batsiolas, I. Nikolaidis, Selective-idling: an Experiment in Transport-layer Power-Efficient Protocol Implementation, Proceedings of the International Conference on Internet Computing, 2000, June, 2000.
[10] K. Brown, S. Singh, M-TCP: TCP for Mobile Cellular Networks, Proceedings of the ACM SIGCOMM CCR, 1997, pp. 19–43.
[11] R. Caceres, L. Iftode, Improving the performance of reliable transport protocols in mobile computing environments, IEEE Journal on Selected Areas in Communications 13 (1995) 5.
[12] D. Chiu, R. Jain, Analysis of the increase/decrease algorithms for congestion avoidance in computer networks, Journal of Computer Networks and ISDN 17 (1) (1989) 1–14.
[13] A. Chockalingam, M. Zorzi, R. Rao, Performance of TCP on Wireless Fading Links with Memory, Proceedings of the IEEE ICC'98, Atlanta, GA, 1998, June, 1998; pp. 201–206.
[14] A. Chockalingam, M. Zorzi, V. Tralli, Wireless TCP Performance with Link Layer FEC/ARQ, Proceedings of the IEEE ICC'99, 1999, June, 1999.
[15] S. Floyd, T. Henderson, The new-reno modification to TCP's fast recovery algorithm, RFC 2582, 1999.

[16] V. Jacobson, Congestion Avoidance and Control, Proceedings of the ACM SIGCOMM'88, 1988, August, 1988; pp. 314–329.
[17] R. Jain, D.M. Chiu, H. Hawe, A quantitative measure of fairness and discrimination for resource allocation in shared systems, Technical Report DEC-TR-301, Digital Equipment Corporation, 1984.
[18] A. Kumar, Comparative performance analysis of versions of TCP in a local network with a lossy link, ACM/IEEE Transactions on Networking (1998) 25–31. August, 1998.
[19] T. Lakshman, U. Madhow, The performance of TCP/IP for networks with high bandwidth-delay products and random loss, IEEE/ACM Transactions on Networking (1997) 336–350.
[20] J. Postel, Transmission control protocol, RFC 793, 1981.
[21] K. Ratnam, I. Matta, WTCP: an Efficient Mechanisms for Improving TCP Performance Over Wireless Links, Proceedings of the Third IEEE Symposium on Computer and Communication (ISCC'98), 1998, June, 1998.
[22] J.H. Saltzer, D. Reed, D. Clark, End-to-end arguments in system design, ACM Transactions on Computer Systems (1984).
[23] V. Tsaoussidis, H. Badr, TCP-probing: Towards and Error Control Schema with Energy and Throughput Performance Gains, Proceedings of the Eighth IEEE Conference on Network Protocols, 2000, November, 2000; pp. 137–202.
[24] V. Tsaoussidis, H. Badr, G. Xin, K. Pentikousis, Energy/Throughput Tradeoffs of TCP Error Control Strategies, Proceedings of the Fifth IEEE Symposium on Computers and Communications, ISCC, 2000, July, 2000; pp. 150–155.
[25] M. Zorzi, R. Rao, Is TCP Energy Efficient, Proceedings of the MoMUC'99, San Diego, California, 1999.