Improving TCP Performance over Networks with Wireless Components using "Probing Devices"

A. Lahanas and V. Tsaoussidis
College of Computer Science
Northeastern University
Boston, MA 02115

Abstract

TCP error control mechanism lacks the ability to detect with precision the nature of potential errors during communication. It is only capable of detecting the results of the errors, namely, that segments are dropped. As a result, the protocol lacks the ability to implement an appropriate error recovery strategy cognizant of current network conditions and responsive to the distinctive error characteristics of the communication channel. TCP sender always calls for the sending window to shrink. We show that probing mechanisms could enhance the error detection capabilities of the protocol. TCP could then flexibly adjust its window in a manner that permits the available bandwidth to be exploited without violating the requirements of stability, efficiency and fairness that need to be guaranteed during congestion.

Our experiments have three distinct goals: First, to demonstrate the potential contribution of probing mechanisms. A simple probing mechanism and an Immediate Recovery strategy are grafted into TCP-Tahoe and TCP-Reno. We show that, this way, standard TCP can improve its performance without requiring any further change. Second, to study the performance of adaptive strategies. An adaptive TCP with probing is used, that is responsive to the detected error conditions by alternating Slow Start, Fast Recovery and Immediate Recovery. An adaptive error recovery strategy can yield better performance. Third, to study the design limitations of the probing device itself. The aggressive or conservative nature of the probing mechanisms themselves can determine the aggressive or conservative behavior of the protocol and exploit accordingly the energy/throughput tradeoff.

1 Introduction

Error control mechanisms are the central component of reliable protocols. They affect a protocol's performance with respect to throughput, energy expenditure, and reliability.

Error control is usually a two-step process: error detection, followed by error recovery. TCP detects errors by monitoring the sequence of data segments received and/or acknowledged. When timeouts are correctly configured, a missing segment is taken to indicate an error, namely that the segment is lost. Reliable protocols usually implement an error recovery strategy based on two techniques: retransmission of missing segments; and downward adjustment of the sender's window size and readjustment of the timeout period.

In the standard TCP versions the receiver can accept segments out of sequence, but delivers them in order to the protocols above. The receiver advertises a window size and the sender ensures that the number of unacknowledged bytes does not exceed this size. For each segment correctly received, the receiver sends back an acknowledgment, which includes the sequence number identifying the next in-sequence byte expected. The transmitter implements a congestion window that defines the maximum number of transmitted-but-unacknowledged bytes permitted. This adaptive window can increase and decrease, but the actual "sending window" never exceeds the minimum of the advertised and congestion window. Standard TCP applies graduated multiplicative and additive adjustments to the sender's congestion window. Historically, TCP-Tahoe was the first modification to TCP. Tahoe's congestion-control algorithm includes Slow Start, Congestion Avoidance, and Fast Retransmit [1, 2, 3]. 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. TCP-Reno introduces Fast Recovery in conjunction with Fast Retransmit. The idea behind Fast Recovery is that a *DACK* is an indication of available channel bandwidth since a segment has been successfully delivered. The sender then halves the congestion window `cwnd` sets the congestion threshold to `cwnd`, and resets the *DACK* counter. In Fast Recovery, *cwnd* is effectively set to half its previous value in the presence of *DACKs*, rather than performing Slow Start.

TCP displays some undesirable patterns of behavior in the context of networks with wireless components. The error recovery mechanism 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 conservatively graduated increase to its reduced window size. During this phase of slow window expansion, opportunities for error-free transmissions are wasted and communication time is extended. In other words, in the presence of infrequent and transient errors, TCP's back-off strategy avoids only minor retransmission at the cost of unnecessary and significantly degraded throughput, and increases overall connection time. Yet, when an error occurs and TCP does back off, it continues to forcefully attempt transmissions within the confines of the reduced window size. In the presence of errors of a relatively persistent nature (fading channel, prolonged and frequent burst errors, congestion), this behavior does not favor energy-saving, since it might yield only minor throughput improvement at high cost in transmission energy. In summary, from the perspective of energy expenditure in the context of heterogeneous wired/wireless networks, TCP seems to possess an inherent tendency to back off too much when it should not, and too little when it should [4, 5, 6, 7, 8, 9, 10]. The central problem lies in the inability of TCP's mechanism to correctly detect the nature of the error, and so it is incapable of responding in an appropriate manner [8]. In addition, the protocol lacks the ability to efficiently monitor network conditions, rapidly readjust its window size in response to changes in these conditions[1], and detect congestion without inducing packet drops, thereby degrading overall performance through additional retransmission and wasted opportunities in maintaining the communication pipe full. The traditional schema of congestion control which uses backwards adjustment of the congestion window in the event of retransmission, and which is exemplified by the TCP paradigm, does not necessarily suffice.

Since the protocol is not optimized for a specific network type or application requirement, its mechanisms permit for several application- and network-specific improvements. TCP's behavior over wired networks, where congestion is a regular cause for packet loss, was initially studied by Jacobson [2]. Recently, TCP behavior over wireless/wired and satellite networks has become a focus of attention. Recent research results [11, 12, 13, 14, 15, 4, 16, 17, 18, 19, 20, 21, 22] have shown that TCP throughput degrades, in the presence of the kind of random and burst errors and long propagation delays typical of wireless and satellite environments, respectively.

Network- and application-specific modifications often involve tradeoffs that damage TCP operations on other networks and/or applications. As a result, some researchers have tended to focus on the development of architectures (e.g., wireless proxies) that assist the protocols operation over such specific networks in order to keep other applications and networks undamaged. For example, the enhancements being discussed in [13, 23, 16, 24] require intervention at the router or base-station level, and, in general, the splitting up of the end-to-end characteristic of TCP behavior. In particular, Ramakrishnan and Floyd [24] propose an Explicit Congestion Notification to be added to the IP protocol in order to trigger appropriate behavior in TCP congestion control and enhance its performance by avoiding retransmission caused by congestion. An obvious drawback of this proposal, as stated by the authors themselves, is that asymmetric routing will necessarily ensue. In addition, the end-to-end autonomy of TCP will be damaged, yet the problem will be only partially solved: the level of congestion will not be effectively estimated, since detection occurs only as a function of routers' threshold values which, moreover, might differ from router to router. Related arguments have been presented for a related approach: RED Gateways [23]. With the new patterns of traffic behavior of Internet routers which no longer match the traces reported in [25], the approach seems less effective. Recent work suggests more modest expectations than were initially predicted [26, 27]. The work performed by the authors in [13] made significant progress towards efficient operations over wireless links but does not overcome the above limitations nor deal with situations of encryption or full-duplex TCP traffic. This work suggested modifications at the base stations and also required changes to TCP semantics on the two hosts. Common experience with proxies leads us to accept that proxy-based solutions cannot have the wide applicability of end-to-end solutions and hence cannot guarantee the expected improvements globally but instead locally, where the modifications have been implemented completely. Acceptance of heterogeneity as the rule and not the exception, renders proxy-based solutions useful but with only limited success on the Internet.

Floyd and Henderson [28] propose a partial acknowledgment method to enhance performance of the TCP Fast Recovery algorithm which, under rather specific conditions, results in some improvement. New Reno addresses the problem of multiple segment drops. In effect, it can avoid many of the retransmit timeouts of Reno. A partial acknowledgment is defined as an *ACK* for new data which does not acknowledge all segments that were in flight at the point when Fast Recovery was initiated. When multiple segments are lost from a window of data, New Reno can

[1]Except for downward adjustment in response to congestion.

recover without waiting for a retransmission timeout.

Enhancements of the TCP acknowledgment strategy are discussed in [29, 30]. In [31] the authors propose an interesting modification which replaces the round trip delay measurements of TCP with estimations of delay along the forward path, and use of an operating point for the number of packets in the bottleneck. In [32] the authors make a significant contribution towards more accurate measurements of network congestion and avoid technical problems that arise from the clock granularity of different operating systems. A study of TCP performance over asymmetric links is presented in [33]. Authors in [33] 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 protocols discussed here; its contribution is focused on the acknowledgment strategy and not on the sender's adjustments[2] and decisions[3].

Today's TCP 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, since they do not flexibly adjust the recovery strategy to the variable nature of the errors. Moreover, a significant missing component from TCP is the mechanism to distinguish the nature of the error in heterogeneous wired/wireless networks. In [8] the authors propose grafting a probing mechanism onto standard TCP in order to enable the protocol with the ability to distinguish the nature of the error based on its frequency and duration, and to determine the recovery strategy accordingly.

Our experiments here have three distinct goals: First, to demonstrate the potential contribution of probing mechanisms. A simple probing mechanism and an Immediate Recovery strategy are grafted into TCP-Tahoe and TCP-Reno. We show that, this way, standard TCP can improve its performance without requiring any further change. Second, to study the performance of adaptive strategies. An adaptive TCP with probing is used, that is responsive to the detected error conditions by alternating Slow Start, Fast Recovery and Immediate Recovery. An adaptive error recovery strategy can yield better performance. Third, to study the design limitations of the probing device itself. The aggressive or conservative nature of the probing mechanisms themselves can determine the aggressive or conservative behavior of the protocol and exploit accordingly the energy/throughput tradeoff.

The remaining of this paper is organized as follows: Section 2 describes a simple design of Probing Mechanisms. In Section 3 we detail our testing methodology and our criteria for evaluating TCP changes. In Section 4 we present the results of grafting Probing into TCP Reno and Tahoe. We describe an experimental design of an Adaptive TCP with Probing in Section 4.1 and we present our observations from our experiments. A modified Probing Device is presented and tested in Section 4.2 and Section 5 presents our concluding remarks.

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

[3]The SACK mechanism is decoupled from the congestion window behavior (see also [29]), which is the point of interest in the present work.

2 Probe Cycles

A probe cycle consists of a structured exchange of very short control segments between sender and receiver, initiated by the sender so as to permit the receiver to make multiple, consecutive measurements from the network[4]. The sender initiates the cycle in response to the notification (from the receiver, the network or the timeout mechanism of the sender) that transmissions should be suspended for the present. The mechanism also provides the capability for sender and receiver to efficiently "checkpoint" with each other in the event of deviation from expected patterns of behavior (e.g., no feedback from the receiver and so on). When a data segment goes missing, the sender, instead of retransmitting and adjusting the congestion window and threshold, initiates a probe cycle during which data transmission is suspended and only probe segments are sent. In the event of persistent error conditions (e.g. congestion), 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.

"Probing" in the context of a reliable, transport-level protocol such as TCP is a fairly generic concept. It can be implemented in a variety of different ways, and further refined in several yet more directions. A critical part of the probing mechanism is the set of the decision rules that determine action at the end of the probe cycle. This action could be the full recovery or a downward adjustment of the window size and, in combination with the decision rules, can determine the aggressive or conservative behavior of the protocol.

2.1 A Simple Probing Device

The sender enters a "probe cycle" when one of the two situations applies:

[4]The prototype implementation measures RTTs. Throughput measurements in the forward direction are a subject of ongoing investigation.

1. A timeout event occurs. If the existent network conditions detected by the time 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 other words, neither of the 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 implemented. Note that here, however, Immediate Recovery will also expand the window in response to further DACKs that were received during the probing cycle. This is analogous to the congestion window expansion phase of Fast Retransmit in Reno and New 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 and New Reno's behavior after 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 back off strategy is more clearly indicated. A more detailed understanding of TCP-Probing mechanisms will be possible through a presentation of their implementation.

The option header extension for Probing includes an option type in order to distinguish between four probe-oriented segments: *PROBE1*, *PROBE2*, *PROBE1_ACK*, *PROBE2_ACK*. These segment types are composed of headers without payload. This is achieved by setting the variable `len`, which represents the length of payload in the segment, to 0 in the function `tcp_output`. The option header extension also includes a field header length, and a probe sequence number which is used to identify probe segments.

Figure 1 describes the state diagram of TCP combined with probing mechanisms. While in *Established* state, probing cycle is triggered either by a 3-DACK event or a timeout. During the probing cycle two RTTs are measured: `rtt1` and `rtt2`. If both measured RTTs are close to the best[5] RTT then TCP will perform Immediate Recovery. Otherwise the network conditions impel for congestion avoidance. To evaluate the impact of probing mechanisms on the error recovery strategies we have designed two protocols: Tahoe-Probing and Reno-Probing. If the measured RTTs do not call for Immediate Recovery then Tahoe-Probing recovers with Slow Start; Reno-Probing will enter Fast Recovery in case the probing cycle was triggered by a 3-DACK event or Slow Start otherwise (i.e., time-out).

[5]The best RTT is the minimum of the measured RTTs from the connection initiation time till the current probing cycle. Probing RTTs do not count for the best RTT selection.

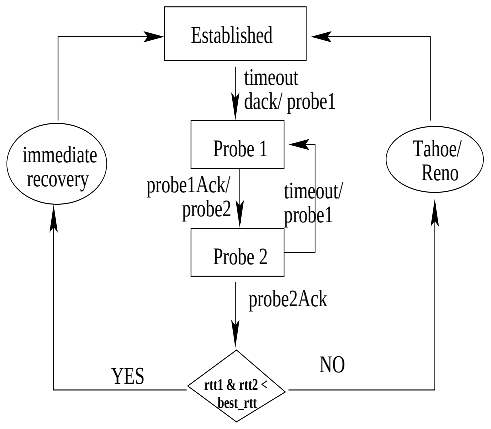

Figure 1. TCP with Probing devices

3 TCP Evaluation: Goals and Methodology

An important consideration of this work is the issue of the effective evaluation of TCP. The author in [34] discusses such principles, most of which are employed here. However, there are other significant details that we consider:

- The use of the appropriate performance metrics. In general, *Goodput*, *Overhead*, and *Time* need to be occasionally combined as performance metrics. A major transmission effort (associated with additional overhead) would not be efficient had a more conservative strategy yielded the same application throughput. The impact of the former strategy on the network (i.e., the amount of data that is injected into the network), and on battery-powered devices (i.e. handhelds) is significant indeed. *Throughput* is frequently used in papers as a network metric ignoring application throughput and overhead *per se*. *Goodput* is also used frequently in terms of `net data/transmitted data`, ignoring time as a performance metric. Finally, *Time* to complete a file transfer cannot be used as the sole performance metric since it does not capture the protocol's behavior; an aggressive behavior might have resulted in redundant retransmission, injecting unjustly packets into the network and expending energy on mobile devices. The measured performance of the protocols is given in terms of *Task Completion Time* and Overhead. Since our data file is of fixed size the Goodput is derived directly from Time.

- The arrangement for selecting versions that differ only at the component of evaluation. The versions used here are comparable with each other. The difference lies exclusively on the error control mechanism. For example, *pace* [34] we do not compare versions that employ different acknowledgment strategies like TCP-SACK.

This version was intentionally excluded for this reason; it is expected that its strategy will be beneficial for all the versions presented.

- Packets and Acknowledgments are dropped. We avoid the assumption that all acknowledgments are correctly delivered; this assumption has not proven to be valid especially in wireless and satellite networks (note that RFC2016 makes use of this assumption). The fact that acknowledgments are cumulative in TCP does not provide sufficient evidence in is own right; for example, it does not cancel the probability of timeouts. Furthermore, this assumption could trigger Fast Retransmit in simulated experiments - due to three DACKs - more frequently than it would do otherwise, had it been avoided.

The purpose of our tests was to evaluate the error control behavior of the TCP versions presented here in response to changes in the network environment. The varying error rate for a selected range of "error phases" was a choice that enabled us to test the protocol behavior in response to duration and error pattern changes. Other factors could also affect the relative performance of the protocols. For example, the window size, could have different impact on their measured performance. More specifically, Probing requires a fixed number of RTTs (currently 2) when conditions are clear and it will probably recover faster than Tahoe a large window of data in this case. Tahoe and Reno will exhibit a more comprehensive behavior with small windows: recovery with exponential growth starting from a single segment might even take less RTTs. Hence, although it appears that with small windows Probing introduces an additional cost, this potential advantage is cancelled[6] for larger windows. Our experiments used a default delay small enough to be beneficial for Tahoe and Reno since the *Delay* $\times$ *Bandwidth* product was limited to a maximum of 10 KB.

The probing protocols presented here, were implemented using the x-kernel protocol framework [35]. Each implementation was individually tested on a single session with two dedicated hosts connected over a local area network. In the experiments carried out there were no other TCP competing flows. The protocols tested here are fully-functional, implemented protocols. A virtual protocols that is configured between the transport and the IP layer implement the error models for the experiments. Although our model is rather simple, it is oriented towards link-level error patterns instead of congestion. The default *Delay* here is the network propagation and transmission delay, plus the x-kernel processing delay. The "virtual protocol", VDELDROP [7], has a core mechanism that consists of a two state (On/Off) continuous time Markov chain. One state was always configured with a zero error 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 varying density, persistence and duration could thus be simulated, depending on the choice of the drop rate and phase duration. The combination of those allows for tests with errors of varying density and duration. Note, that the error rate does not report the number of packet drops. Instead, it reports the dropping intention. This setting for a time-based model allows for capturing better[7] the behavior of the protocols.

The application protocol had a fixed task: to send 5 MB (5,242,880 bytes) data sets for transmission. The size of the transmission message was selected to be sufficiently large to accommodate several phase changes which, in turn, had to be sufficiently large to permit for full recovery of the window size. We took measurements of the *Task Completion Time* (TCT) and of the *Total* number of bytes transmitted (i.e. including protocol control overhead transmissions, data segment retransmission, etc.). Those results are presented in the tables in the Appendix.

The Error Rate in the tables and charts denotes the dropping rate during the On phase of the VDELDROP protocol (see [7] for more details). Therefore, the dropping rate reported refers to segments during the On phases, not the averaged overall drop rate across On/Off phases. The error protocol was configured in the protocol stack of both participating hosts, thereby, data packets as well as acknowledgments were dropped during the On phases. Our tests were repeated a sufficient number of times; that was indicated by the Standard Deviation measured after 10 experiments. The average is reported here.

4 TCP Reno and Tahoe with Probing

Figures 2, 3 and 4 draw the performance of the four protocols for transient errors which are typical for heterogeneous wired and wireless networks, and relatively short bad-phase duration. For small error rates and windows the results show that the probing mechanisms could slightly impact negatively the time and overhead of Tahoe and Reno. As noted above, larger windows (i.e., buffers, and/or Delay $\times$ Bandwidth products) cancel this effect[8]. However, the present example is the worst case scenario for the probing device.

For small rates of transient errors, where probing can be vulnerable to multiple 3-DACK events, TCP with Probing

[6]Experimental evidence of our claim is not presented here. Due to space limitations, a webpage with additional results will be provided presently.

[7]An arrangement for dropping a specified data rate from a single flow could be more appropriate for simulating a RED Gateway.

[8]Our experiments with larger windows drop the level of TCT 4% and the level of overhead 2%.

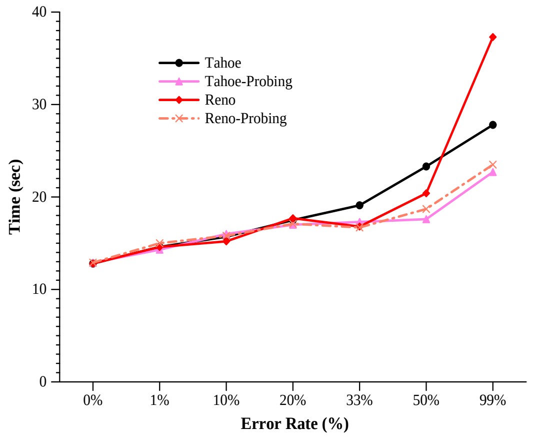

Figure 2. Performance with 1-5 sec On/Off phase.

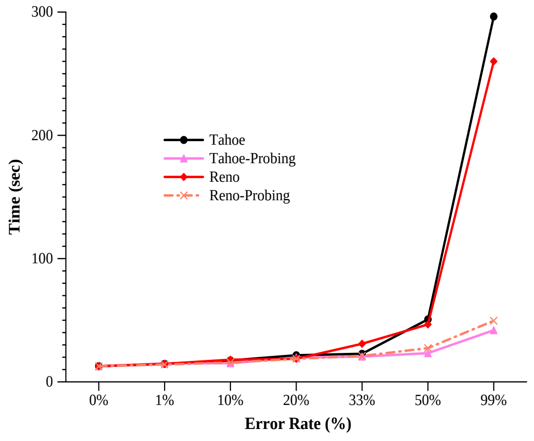

Figure 4. Performance with 4-5 sec On/Off phase.

imposes an improvement. It can be observed that the performance of Tahoe-Probing is significantly improved. For example, Figure 2 shows that Tahoe-Probing is 2% faster than Tahoe at 1% error rate; 3% faster at 20% error rate and 9% faster at 33% error rate. Reno-Probing starts to improve at 20% error rate (it is 3.5% faster than Reno).

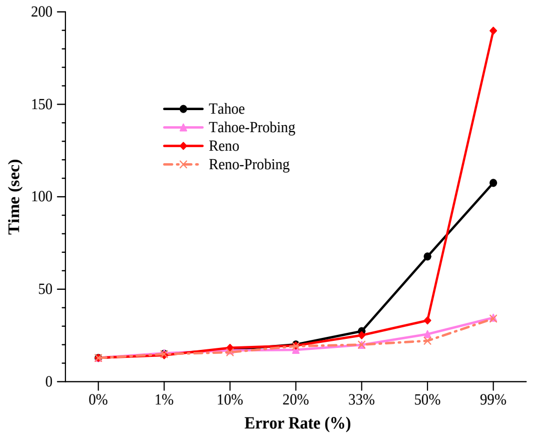

Figure 3. Performance with 3-5 sec On/Off phase.

The relative improvement of probing protocols become more evident at high error rates. Consecutive timeouts at this stage of the experiment cause TCP to loose its ability to rapidly detect error-free channels, thereby wasting opportunities of error- and congestion-free transmissions. Probing mechanisms entail a dual promotion to TCP's error-detection capabilities: (i) they enable a distinction of the nature of the error, and (ii) they work against an unduly-delayed detection. An extended time-out will not permit the protocol to exploit efficiently the capacity of an error free link.

Our results affirm that both Tahoe-Probing and Reno-Probing experience significant improvement at high error rates. Tahoe-Probing is 9% faster than Tahoe at 33% error rates (see Figure 2); 24% faster at 50% error rate and 18% faster than Tahoe at 99% error rate. Reno-Probing is 1.2% faster than Reno at 33% error rate, 8% faster at 50% error rate and 36% faster at 99% error rate (see Figure 2). For the same error duration the overhead bears a significant improvement for the probing protocols. Tahoe-Probing has 4% less overhead than Tahoe at 1% error rate; 1% less overhead at 20% error rate; 6% less overhead at 33% error rate, 5% less overhead at 50% and 8% less overhead at 99% error rate (see Figure 5). Note that a 99% error rate represents an occasion of a correlated error; a situation that appears frequently in fading channels or during handoffs. In absolute terms the latter represents 8908 bytes. This number grows sharply with long-lasting errors and its importance is crucial for mobile devices that operate on battery-based energy sources. Reno-Probing has also a slight improvement in the overhead which is 3% less than that of Reno at 50% error rate and is also 4% lower at 99% error rates. It is important to note that the protocol behavior under heavy errors is the most challenging design goal: a fault or inefficient operation at these stages will have distorting effects on the overall energy and throughput performance. Otherwise, the efficiency will be damaged to some extend which might or might not be acceptable for the user but it will not determine the operational lifetime of a battery-powered device.

We conclude that longer error duration deteriorate further the performance of TCP Tahoe and Reno since they suffer from consecutive time-outs which are provenly harmful for their retransmission capabilities. On the contrary, TCP

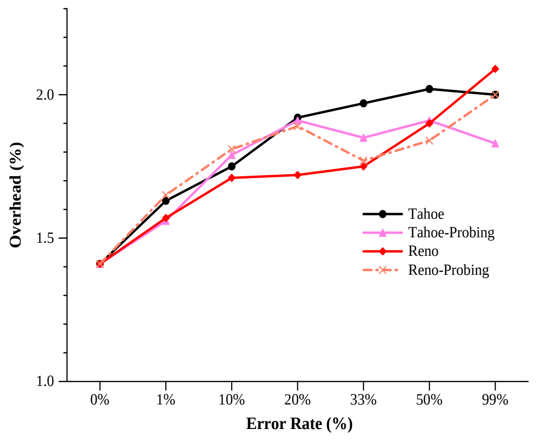

Figure 5. Protocols' overhead with 1-5 sec On/Off phase.

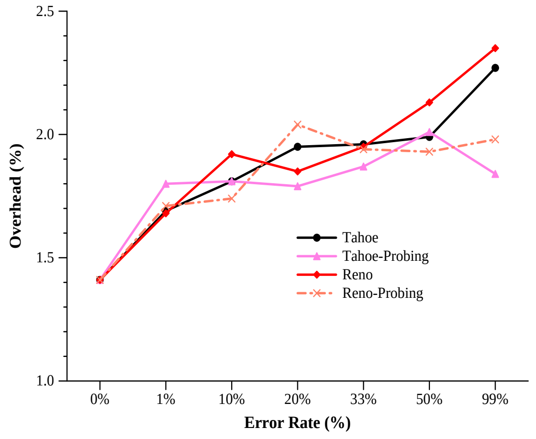

Figure 6. Protocols' overhead with 3-5 sec On/Off phase.

with probing will start a probing cycle that will probably not terminate until the error level has dropped or cleared. During this time, the congestion control parameters (time-out value, congestion window and threshold value) are not affected. As soon as probing completes its cycle, the protocol resumes data transmission. Figure 3 shows the performance of protocols for a representative error-phase duration of 3 seconds. We note again that for small error rates probing mechanisms might cause a slight increase on the protocol task completion time and overhead. We measured the average sending window at this stage to be 2KB. It is easier to recover from Slow Start a window size of 2KB than it would be for a window of 64K. Recall that Immediate Recovery from uninterrupted probing takes 2RTTs. However, we can observe that Tahoe-Probing is 2.3%, 14%, 27%, 62% and 3 times faster than Tahoe at 10%, 20%, 33%, 50% and 99%

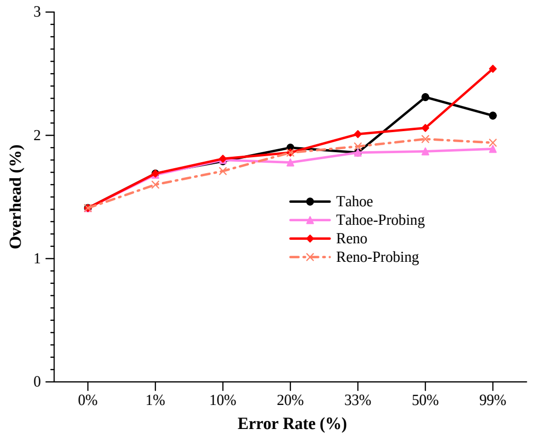

Figure 7. Protocols' overhead with 4-5 sec On/Off phase.

error rates respectively. Reno-probing is also 13%, 1.5%, 20%, 33% and 5.5 times faster than Reno at 20%, 33%, 50% and 99% error rates respectively (see Figure 3). Probing also reduces significantly the protocols' overhead. Both TCP versions with probing reduce their overhead from 1% to 18% for Tahoe and from 1% to 15% for Reno, respectively (see Figure 6). For 2 and 4 seconds error duration the relative performance gain of probing protocols is almost the same as in 3 seconds error duration. Details of these results are listed in tables 1, 2, 3 and 4 in the Appendix.

4.1 Adaptive TCP with Probing

Observing TCP behavior from another perspective, we realize that the relative performance gain of aggressive or conservative error recovery strategies varies as the error rate changes. Based on this observation an interesting issue would be to combine the error recovery mechanisms of TCP Tahoe and Reno into one protocol and study its performance. *Adaptive-TCP* addresses this issue: it combines the aggressive and conservative error recovery strategies of Reno and Tahoe, respectively. The results of Probing are used to enable a decision about the choice of the appropriate recovery strategy.

Figure 8 presents the state transition diagram of A-TCP. The protocol enters into a probing cycle either after a 3-DACK event or a time-out. If both measured RTTs during the probing cycle are smaller than the best RTT then A-TCP will enter Immediate Recovery. In response to the varying nature of the errors (congestion or wireless errors) A-TCP performs a feedback-based [9] recovery. The current window recovery phase (i.e., Slow Start or Congestion Avoidance)

[9]Feedback is provided by the probing cycle

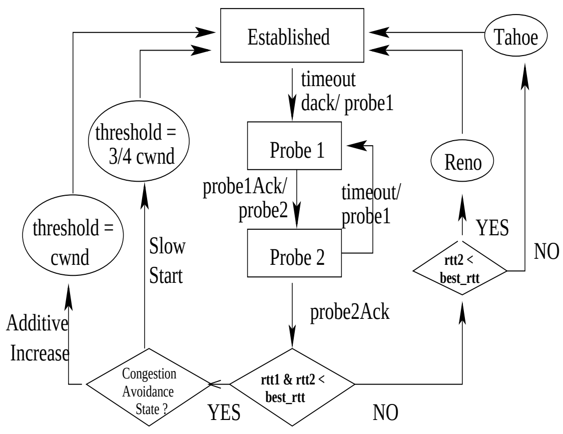

Figure 8. Adaptive TCP state diagram

is also taken into account. More precisely, if the probing cycle interrupted the Slow Start phase, and A-TCP determines action with Immediate Recovery, then the congestion window and threshold are set at 3/4 the value of the congestion window prior to the error. Traditionally, packets that are lost due to congestion during Slow Start are treated more conservatively. Stability and fairness are becoming most important concerns in this case. However, an error during Additive Increase calls for the Immediate Recovery to perform the same actions as Reno- or Tahoe-Probing.

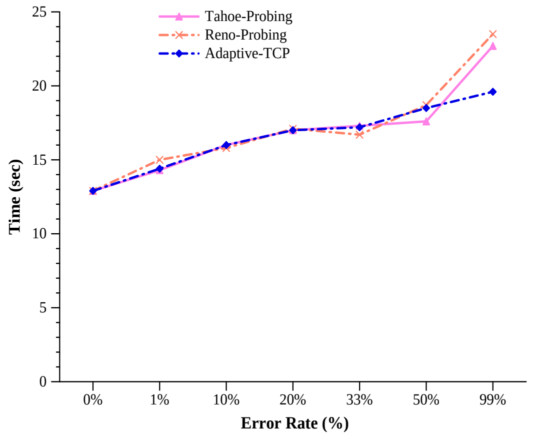

Figure 9. Time performance of TCP-Probing and Adaptive-TCP with probing. 1/5 seconds On/Off phase

The measurements taken during the probing cycle determine action for the recovery. If they both are worse than the best RTT, the protocol reacts conservatively and enters Slow Start. If only the second measured RTT is better than the best RTT, A-TCP risks the assumption that congestion level is improving and enters Fast Recovery. Two fine measurements during the Probe Cycle call for Immediate Recovery.

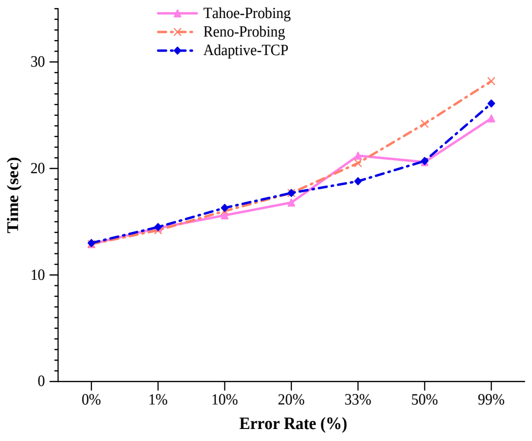

Figure 10. Time performance of TCP-Probing and Adaptive-TCP with probing. 2/5 seconds On/Off phase

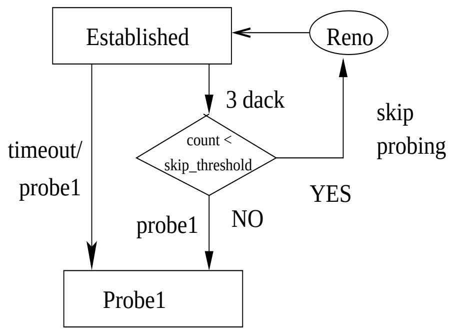

Figure 11. Adaptive TCP with Skip Probing state diagram

Figures 9, 10, 12, 14, 15 and 16 plot the time performance and the overhead of A-TCP, Tahoe-Probing and Reno-Probing. Even though these protocols have slightly different Immediate Recovery mechanism, it is not difficult to observe the impact of adaptive error recovery strategies on protocol's performance. Figures 9, 10, 12 show that the Task Completion Time of A-TCP has an upper and lower limit the time of Tahoe-Probing and Reno-Probing. Likewise, the overhead of A-TCP has an upper and lower limit the overhead of Tahoe-Probing and Reno-Probing. At high error rates, however, the A-TCP might outperform the non-adaptive protocols. For example Figure 9) shows that A-TCP is 13% faster than Tahoe-Probing and is 16% faster than Reno-Probing at 99% error rate. Figure 10) shows A-TCP 8% faster than Reno-Probing and 11% faster than

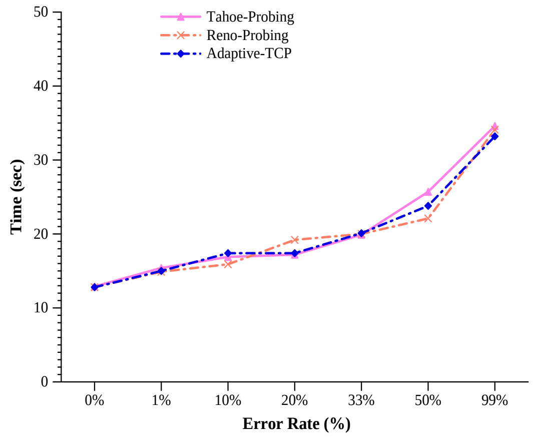

Figure 12. Time performance of TCP-Probing and Adaptive-TCP with probing. 3/5 seconds On/Off phase

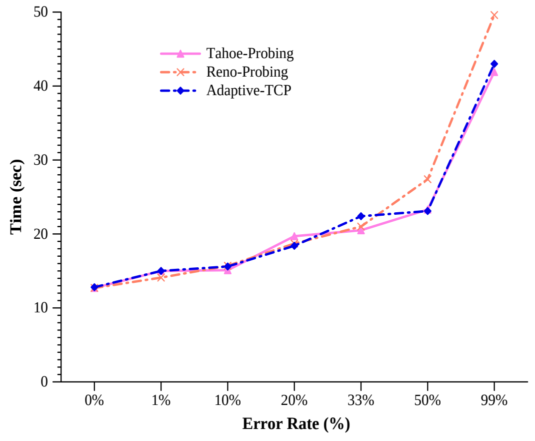

Figure 13. Time performance of TCP-Probing and Adaptive-TCP with probing. 4/5 seconds On/Off phase

Tahoe-Probing at 33% error rate. As the error duration increases, the time of A-TCP tends to outmatch the time of the fastest protocol. At high error rates the possibility of Immediate Recovery is rare; if this happens it probably causes most of the transmitted data to be dropped again. Since A-TCP applies a more conservative recovery than both other protocols, it will require less retransmission after the ”bad phase”. In fact, since it will also experience less time-outs, it will be capable of detecting faster any potential change. Good performance of A-TCP can also be observed from figures 12 and 13 which present the performance of the protocol for error durations of 3 and 4 seconds respectively. Figure 12 shows that at 20% and 33% the A-TCP time equals to that of Tahoe-Probing (which is the fastest at these error rates); at 50% and 99% error rates A-TCP time tends towards the time of Reno-Probing. Figure 13 shows that the time of A-TCP at 10% error rates is smaller and close to the time of Tahoe-Probing; at 20% and 33% the A-TCP time is close to that of Reno-Probing and at 99% it tends towards the time of Tahoe-Probing.

4.2 Selective Probing

Since the probing cycle itself adds at least two RTTs to the connection time, it appears to have greater cost during errors that are relatively frequent and transient. The reason is that the strategy after the probing is likely to call for Immediate Recovery; however, the two RTT’s will be wasted.

Behind the idea of Selective Probing stands the observation that a heavy error condition will extend a probing cycle anyway and actual data transmission will be avoided. Hence, it could be feasible to apply a probing scheme that is not triggered every single time a packet goes missing. Instead, this could happen once every small time intervals; small enough to allow for detecting congestion levels that potentially build up. As noted, heavy error rates will be captured indeed, since the probing cycle will be extended. This can be viewed from another perspective as one probing cycle per window of data, for sufficiently large windows.

We call this experimental protocol *Selective-Probing* (SP-TCP) and present its state transition diagram in Figure 11. Selective probing has the same adaptive error recovery strategy as described in Section 4.1. Its transition diagram is the same as that of A-TCP (see Figure 8) with exception the transition from state *‘Established’* to *‘Probe1’* triggered by a 3-DACK event. SP-TCP uses a new *counter* and *skip-threshold* variable. For each 3-DACK event, SP-TCP compares the counter with the skip-threshold value and does not start Probe Cycle unless the value of the counter exceeds the skip-threshold value or the specified time-interval. Fast Retransmit and Fast Recovery are executed if the counter is smaller than the skip-threshold value. The value of the counter is incremented every time TCP estimates the RTT value and it is reset every time the protocol enters Probe Cycle. In our experiments we used small values for both time and counter thresholds. In response to a time-out event SP-TCP enters probing cycle just like Adaptive-TCP.

Figures 18, 19, 20, 21, 22, 23, 24, and 25 present the time and overhead performance of SP-TCP and A-TCP. It can be observed that the improvements appear more significant with lower error rates. For example, Figures 18, 19 show that A-TCP yields worse performance than SP-TCP especially at lower error rates (1%, 10%, 20%, and 33%) where 3-DACK events are more likely to happen than at high error rates. The protocols that enter probing cycle for every loss

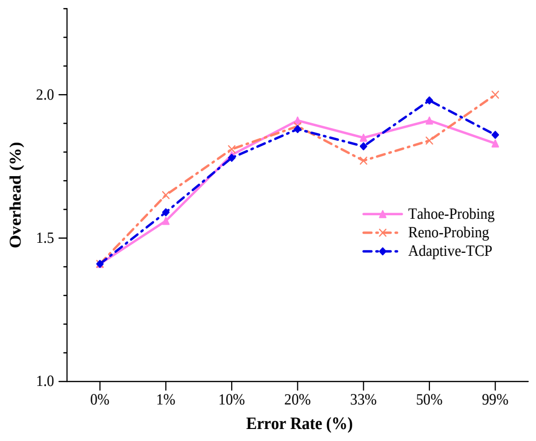

Figure 14. Overhead of TCP-Probing and Adaptive-TCP with probing. 1/5 seconds On/Off phase

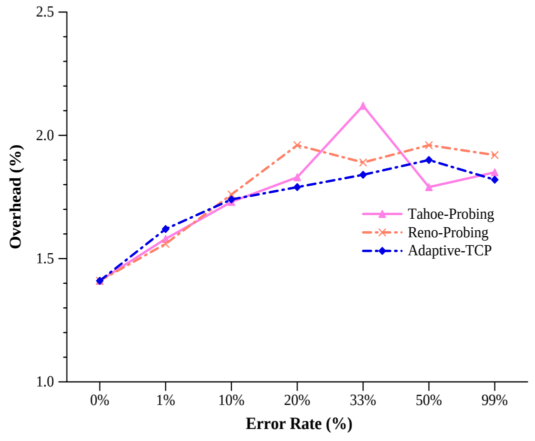

Figure 15. Overhead of TCP-Probing and Adaptive-TCP with probing. 2/5 seconds On/Off phase

event (3-DACK or time-out) will waste some RTTs. The Selective-Probing strategy avoids the vulnerability of consecutive probing cycles by adopting a Fast Recovery mechanism during frequent 3-DACK events and improves significantly the performance of the protocol especially at low error rates where the 3 DACKs is more likely to occur.

For long error durations the protocols most probably suffer from time-outs. The SP-TCP adopts probing strategies after a time-out event and has the same performance as that of probing TCPs or Adaptive-TCP. The Fast Recovery, however, has its own drawbacks; this was evident from our experiments in section 3 with high error rates (see also [7]). Thereby, SP-TCP could suffer, for the same reasons, at high error rates. This can be seen by the degraded performance of SP-TCP at short error durations and high error rates (50% and 99% error rates - see figures 18, 19). The degraded time performance is reflected in the overhead also. Fast Recovery strategy while prolonged network congestion or link errors will result in further packet loss and an increase in the protocol's overhead (see figures 22, 23).

5 Conclusion and Future Work

Probing constitutes a research topic in its own right. Several improvements can be made. Our experiments indicate a potential direction of further research towards both adaptive protocols and probing devices. Indeed, probing devices can also be adaptive themselves; be commensurate with the measured RTTs, behave aggressively or conservatively. Clearly, an adaptive protocol design would require further experience with alternative designs. Although probing schemes have been used in the past, the authors are not aware of any published results that discuss such design issues and their corresponding impact on transport protocols. The topic is becoming important due to recent advances of wireless Internet.

Furthermore, we have shown that Probing enables TCP to go beyond a circumscribed functionality exclusively focused on congestion control, and to move towards a *universal* error control. Its self-adjusting strategy is responsive to the nature of the errors and achieves a significant performance gain compared to Standard TCP. The better the adjusting strategy matches the network conditions, the more efficient the probing device would be. Our experiments demonstrate the validity of this concept and provide directions for further research.

References

[1] M. Allman, V. Paxson, and W. Stevens, "TCP Congestion Control," *RFC 2581*, April 1999.

[2] V. Jacobson, "Congestion Avoidance and Control," in *Proceedings of the ACM SIGCOMM '88*, August 1988.

[3] M. Mathis, J. Semke, J. Mahdavi, and T. Ott, "The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm," *Computer Communication Review*, vol. 27, July 1997.

[4] S. Gorinsky and H. Vin, "Additive Increase Appears Inferior," tech. rep., University of Texas, Austin, 2000.

[5] V. Tsaoussidis, H. Badr, and R. Verma, "Wave & Wait Protocol (WWP) An Energy-Saving Transport Protocol for Mobile IP-Devices," in *Proceedings of the*

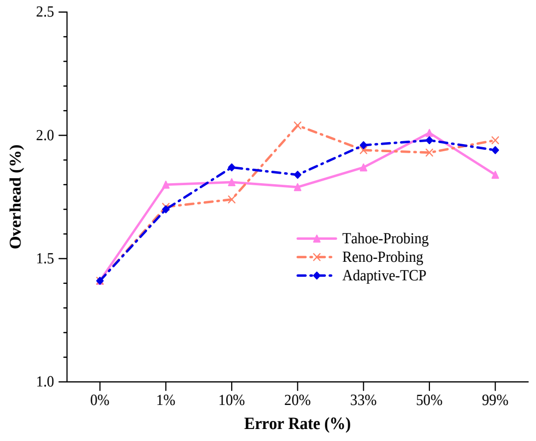

Figure 16. Overhead of TCP-Probing and Adaptive-TCP with probing. 3/5 seconds On/Off phase

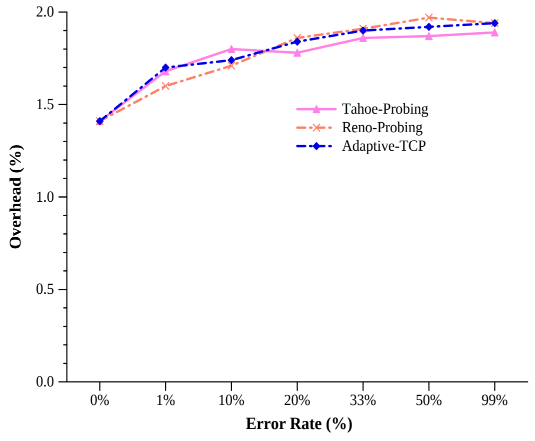

Figure 17. Overhead of TCP-Probing and Adaptive-TCP with probing. 4/5 seconds On/Off phase

7th International Conference on Network Protocols., 1999.

[6] V. Tsaoussidis, A. Lahanas, and H. Badr, "The Wave and Wait Protocol: High Throughput and Low Energy Expenditure for Mobile-IP Devices," in *Proceedings of the 8th IEEE Conference on Networks, IEEE ICON 2000, Singapore*, 2000.

[7] V. Tsaoussidis, H. Badr, G. Xin, and K. Pentikousis, "Energy / Throughput Tradeoffs of TCP Error Control Strategies," in *Proceedings of the 5th IEEE Symposium on Computers and Communications, ISCC*, 2000.

[8] V. Tsaoussidis and H. Badr, "TCP-Probing: Towards an Error Control Schema with Energy and Throughput Performance Gains," in *Proceedings of the 8th IEEE Conference on Network Protocols*, 2000.

[9] G. Xylomenos and G. Polyzos, "TCP and UDP Performance over a Wireless LAN," in *Proceedings of the IEEE INFOCOM*, 1999.

[10] M. Zorzi and R. Rao, "Energy Efficiency of TCP," in *Proceedings of the MoMUC '99, San Diego, California*, 1999.

[11] M. Allman, D. Glover, and L. Sanchez, "Enhancing TCP Over Satellite Channels using Standard Mechanisms," *RFC 2488*, January 1999.

[12] B. Bakshi, P. Krishna, N. Vaidya, and D. Pradhan, "Improving Performance of TCP over Wireless Networks," in *Proceedings of the IEEE 17th ICDCS'97*, pp. 365–373, 1997.

[13] H. Balakrishnan, V. Padmanabhan, S. Seshan, and R. Katz, "A Comparison of Mechanisms for Improving TCP Performance Over Wireless Links," *ACM/IEEE Transactions on Networking*, December 1997.

[14] H. Balakrishnan, S. Seshan, E. Amir, and R. Katz, "Improving TCP/IP Performance over Wireless Networks," in *Proceedings of the 1st ACM Int'l Conf. On Mobile Computing and Networking (Mobicom)*, November 1995.

[15] A. Chockalingam, M. Zorzi, and R. Rao, "Performance of TCP on Wireless Fading Links with Memory," in *Proceedings of the IEEE ICC'98, Atlanta, GA*, June 1998.

[16] Z. Haas and P. Agrawal, "Mobile-TCP: An Asymmetric Transport Protocol Design for Mobile Systems," in *Proceedings of the IEEE International Conference on Communications (ICC'97)*, 1997.

[17] A. Kumar, "Comparative Performance Analysis of Versions of TCP in a Local Network with a Lossy Link," in *ACM/IEEE Transactions on Networking*, August 1998.

[18] A. Lahanas, D. Vardalis, and V. Tsaoussidis, "On the Performance of Reliable Transport Protocols over Wide Area Networks," in *Proceedings of the International Conference on Internet Computing, IC 2000*, CSREA Press, Las Vegas, Nevada, June 2000.

[19] T. Lakshman and U. Madhow, "The Performance of TCP/IP for Networks with High Bandwidth-Delay

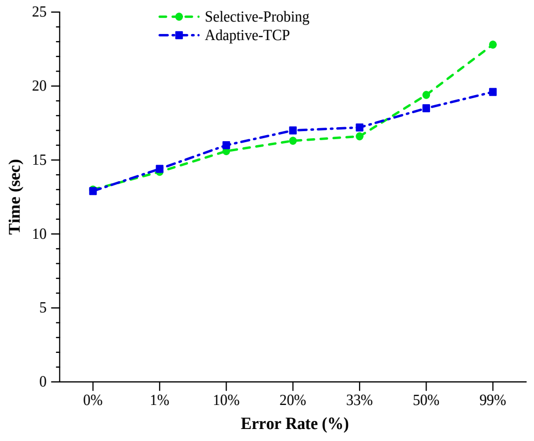

Figure 18. Time performance of Selective-Probing TCP. 1/5 seconds On/Off phase

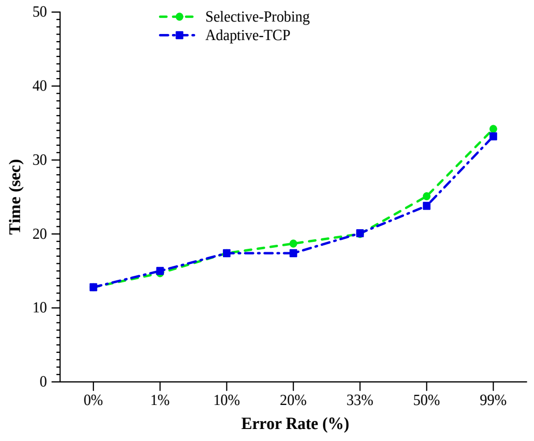

Figure 20. Time performance of Selective-Probing TCP. 3/5 seconds On/Off phase

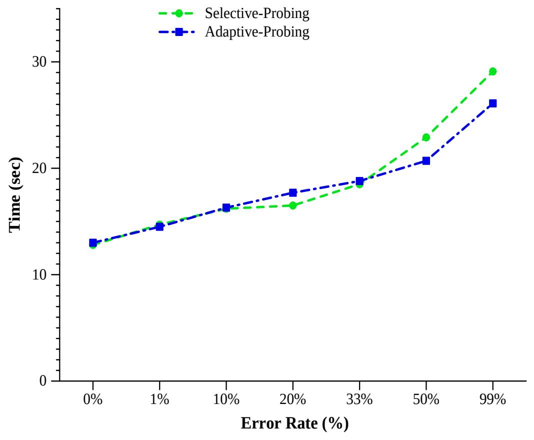

Figure 19. Time performance of Selective-Probing TCP. 2/5 seconds On/Off phase

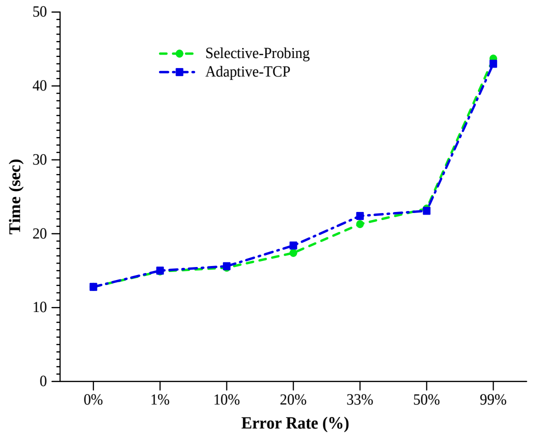

Figure 21. Time performance of Selective-Probing TCP. 4/5 seconds On/Off phase

Products and Random Loss," *IEEE/ACM Transactions on Networking*, pp. 336–350, June 1997.

[20] T. Lakshman and U. Madhow, "TCP/IP Performance with Random Loss and Bidirectional Congestion," *IEEE/ACM Transactions on Networking*, vol. 8, pp. 541–555, October 2000.

[21] C. Partridge and T. Shepard, "TCP Performance over Satellite Links," *IEEE Network*, vol. 11, pp. 44–99, September 1997.

[22] M. Zorzi and R. Rao, "Perspective on the Impact of Error Statistics on Protocols for Wireless Networks," *IEEE Personal Communications Magazine*, October 1999.

[23] S. Floyd and V. Jacobson, "Random Early Detection Gateways for Congestion Avoidance," *IEEE/ACM Transactions on Networking*, vol. 1, pp. 397–413, August 1993.

[24] K. Ramakrishnan and S. Floyd, "A Proposal to add Explicit Congestion Notification (ECN) to IP," *RFC 2481*, January 1999.

[25] V. Paxson, "End-to-End Routing Behavior in the Internet," *IEEE/ACM Transactions on Networking*, vol. 5, pp. 601–615, October 1997.

[26] M. May, J. Bolot, C. Diot, and B. Lyles, "Reasons not to Deploy RED," tech. rep., INRIA, France, June 1999.

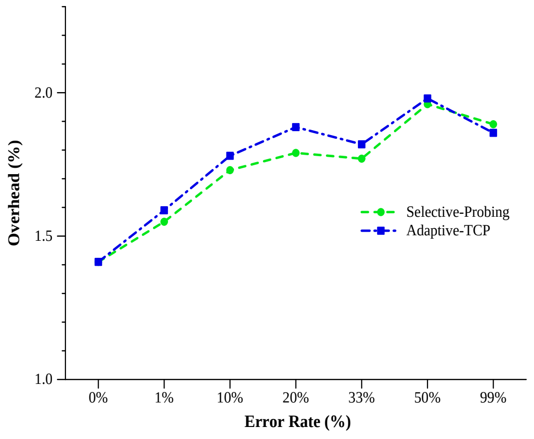

Figure 22. Overhead of Selective-Probing TCP. 1/5 seconds On/Off phase

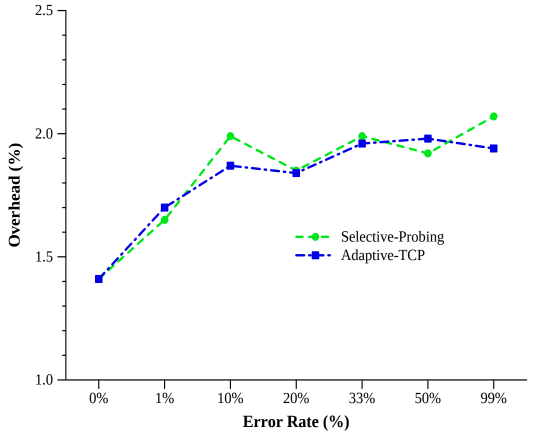

Figure 24. Overhead of Selective-Probing TCP. 3/5 seconds On/Off phase

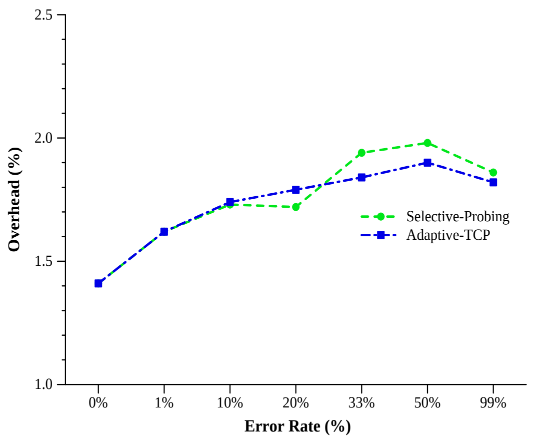

Figure 23. Overhead of Selective-Probing TCP. 2/5 seconds On/Off phase

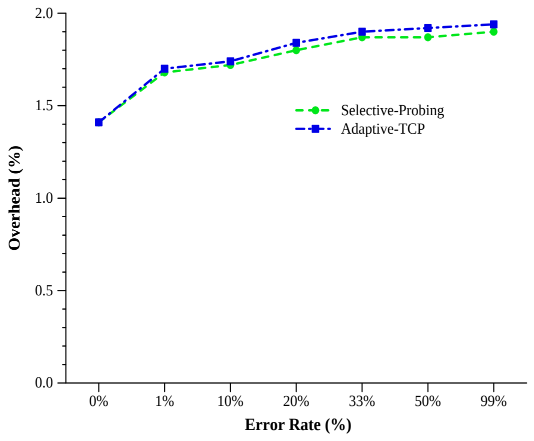

Figure 25. Overhead of Selective-Probing TCP. 4/5 seconds On/Off phase

[27] M. May, T. Bonald, and J. Bolot, “Analytic Evaluation of RED Performance,” in *INFOCOM 2000*, August 2000.

[28] S. Floyd and T. Henderson, “The New-Reno Modification to TCP’s Fast Recovery Algorithm,” *RFC 2582*, April 1999.

[29] M. Allman, “On the Generation and Use of TCP Acknowledgments,” *ACM Computer Communication Review*, October 1998.

[30] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, “TCP Selective Acknowledgement Options,” *RFC 2018*, Aprill 1996.

[31] C. Parsa and L. Aceves, “Improving TCP Congestion Control over Internets with Heterogeneous Transmission Media,” *IEEE Conference on Network Protocols, ICNP ’99, Toronto*, 1999.

[32] L. Brakmo and L. Peterson, “Tcp Vegas: End to End Congestion Avoidance on a Global Internet,” *IEEE Journal on Selected Areas of Communications*, October 1995.

[33] H. Balakrishnan, V. Padmanabhan, and R. Katz, “The Effects of Asymmetry in TCP Performance,” in *Proceedings of the 3rd ACM/IEEE Mobicom Conference*, September 1997.

[34] M. Allman, “On the Effective Evaluation of TCP,” *CCR*, 1999.

[35] “The X-kernel,” www.cs.princeton.edu/xkernel.

Appendix

Error Rate:	0%	1%	10%	20%	33%	50%	99%
Protocol	Task Completion Time (sec)						
Tahoe	12.8	14.6	15.7	17.5	19.1	23.3	27.8
Tahoe-Probing	12.9	14.3	16.0	17.0	17.3	17.6	22.7
Reno	12.8	14.6	15.2	17.7	16.8	20.4	37.3
Reno-Probing	12.9	15.0	15.8	17.1	16.7	18.7	23.5
Adaptive-TCP	12.9	14.4	16.0	17.0	17.2	18.5	19.6
Skip-Probing	13.0	14.2	15.6	16.3	16.6	19.4	22.8
	Overhead (%)						
Tahoe	1.41	1.63	1.75	1.92	1.97	2.02	2.00
Tahoe-Probing	1.41	1.56	1.79	1.91	1.85	1.91	1.83
Reno	1.41	1.57	1.71	1.72	1.75	1.90	2.09
Reno-Probing	1.41	1.65	1.81	1.89	1.77	1.84	2.00
Adaptive-TCP	1.41	1.59	1.78	1.88	1.82	1.98	1.86
Skip-Probing	1.41	1.55	1.73	1.79	1.77	1.96	1.89

Table 1. Performance of protocols with 1/5 sec on/off phase

Error Rate:	0%	1%	10%	20%	33%	50%	99%
Protocol	Task Completion Time (sec)						
Tahoe	12.9	14.6	16.9	17.4	20.4	24.7	65.4
Tahoe-Probing	12.9	14.4	15.6	16.8	21.2	20.6	24.7
Reno	12.8	14.3	17.0	16.8	20.4	23.9	69.8
Reno-Probing	12.9	14.2	16.0	17.7	20.5	24.2	28.2
Adaptive-TCP	13.0	14.5	16.3	17.7	18.8	20.7	26.1
Skip-Probing	12.8	14.7	16.2	16.5	18.5	22.9	29.1
	Overhead (%)						
Tahoe	1.41	1.64	1.92	1.82	1.85	2.01	2.21
Tahoe-Probing	1.41	1.58	1.73	1.83	2.12	1.79	1.85
Reno	1.41	1.58	1.80	1.67	1.93	1.94	2.16
Reno-Probing	1.41	1.56	1.76	1.96	1.89	1.96	1.92
Adaptive-TCP	1.41	1.62	1.74	1.79	1.84	1.90	1.82
Skip-Probing	1.41	1.62	1.73	1.72	1.94	1.98	1.86

Table 2. Performance of protocols with 2/5 sec on/off phase

Error Rate:	0%	1%	10%	20%	33%	50%	99%
Protocol	Task Completion Time (sec)						
Tahoe	12.8	15.1	17.3	20.1	27.3	67.7	107.5
Tahoe-Probing	12.9	15.4	16.9	17.2	19.9	25.7	34.6
Reno	12.9	14.3	18.3	19.5	25.1	33.1	189.8
Reno-Probing	12.8	14.9	15.9	19.2	20.0	22.1	34.1
Adaptive-TCP	12.8	15.0	17.4	17.4	20.1	23.8	33.2
Skip-Probing	12.8	14.7	17.4	18.7	20.0	25.1	34.2
	Overhead (%)						
Tahoe	1.41	1.69	1.81	1.95	1.96	1.99	2.27
Tahoe-Probing	1.41	1.80	1.81	1.79	1.87	2.01	1.84
Reno	1.41	1.68	1.92	1.85	1.95	2.13	2.35
Reno-Probing	1.41	1.71	1.74	2.04	1.94	1.93	1.98
Adaptive-TCP	1.41	1.70	1.87	1.84	1.96	1.98	1.94
Skip-Probing	1.41	1.65	1.99	1.85	1.99	1.92	2.07

Table 3. Performance of protocols with 3/5 sec on/off phase

Error Rate:	0%	1%	10%	20%	33%	50%	99%
Protocol	Task Completion Time (sec)						
Tahoe	12.7	14.6	17.4	21.6	22.8	50.7	296.4
Tahoe-Probing	12.7	15.0	15.1	19.7	20.5	23.3	41.9
Reno	12.7	14.6	18.0	18.7	30.9	46.6	260.0
Reno-Probing	12.7	14.1	15.7	18.7	21.0	27.4	49.6
Adaptive-TCP	12.8	15.0	15.6	18.4	22.4	23.1	43.0
Skip-Probing	12.8	14.9	15.4	17.4	21.3	23.4	43.7
	Overhead (%)						
Tahoe	1.41	1.69	1.79	1.90	1.86	2.31	2.16
Tahoe-Probing	1.41	1.68	1.80	1.78	1.86	1.87	1.89
Reno	1.41	1.69	1.81	1.86	2.01	2.06	2.54
Reno-Probing	1.41	1.60	1.71	1.86	1.91	1.97	1.94
Adaptive-TCP	1.41	1.70	1.74	1.84	1.90	1.92	1.94
Skip-Probing	1.41	1.68	1.72	1.80	1.87	1.87	1.90

Table 4. Performance of protocols with 4-5 sec on/off phase