TCP-Probing: Towards an Error Control Schema with Energy and Throughput Performance Gains

Vassilios Tsaoussidis
College of Computer Science
Northeastern University
Boston, MA, USA

Hussein Badr
Computer Science
SUNY Stony Brook
NY, USA

Abstract

Today's universal communications increasingly involve mobile and battery-powered devices (e.g. hand-held, laptop) over wired and wireless networks. Energy efficiency, as well as throughput, are becoming service characteristics of dominant importance in communication protocols. Although standard TCP versions lack the functionality to efficiently adjust their error-control strategies to distinct characteristics of network environments and to specific constraints of communicating devices, the wide range of TCP-based applications have rendered TCP the de facto standard for reliable end-to-end communications. In this work we propose "grafting" two components of strategic significance onto standard TCP: a Probing mechanism and an Immediate Recovery strategy. Our results show that these enhancements yield higher throughput while maintaining lower levels of energy expenditure, and thus have the potential of promoting TCP's congestion control to a universal error-control schema for heterogeneous wired/wireless channels. Furthermore, the enhancements do not damage the end-to-end characteristics of the TCP, nor do they require changes to its semantics: the mechanisms are implemented as option extensions to the TCP header. We compare "TCP-Probing" with Tahoe, Reno, and New Reno, and show that it can be a protocol of choice for heterogeneous wired/wireless communications with respect to energy and throughput performance.

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, TCP [8] has evolved with a focus on its throughput efficiency in more-or-less "homogeneous" environments (e.g. wired vs. wireless); consequently, homogeneity was also reflected in the nature of the errors that were considered (e.g. congestion/transmission errors vs. random/burst/fading-channel errors). Jacobson [5] 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 & Henderson, for example, have shown that TCP throughput in wired environments can be made to improve using Partial Acknowledgments and Fast Recovery [4]. Other recent enhancements require intervention at the router or base-station level, and, in general, the splitting up of the end-to-end characteristic of TCP behavior. Ramakrishnan and Floyd [9], for example, propose an Explicit Congestion Notification to be added to the IP protocol (an approach similar to RED Gateways) 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 the fact 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. On the other hand, Balakrishnan et al. [2], Lakshman & Madhow [7], and others (e.g. [6]), have shown that TCP throughput degrades when the protocol is used over satellite or wireless links. The

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

- 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.
- The motivating force driving these modifications ignores energy efficiency, which is becoming a key performance issue.

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 only published studies of TCP energy consumption are [3, 13] and [10, 11]. The authors in [13] present results, based on a stochastic model of TCP behavior[1] while the authors in [10] present an energy-saving approach using "waves" and "probing" with an experimental transport level protocol.

The key to throughput *and* energy efficiency in a reliable transport protocol 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. 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 effective throughput ("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. Another source of energy wastage comes from the protocol's inability to efficiently monitor network conditions without incurring data-segment drops.

In order to enhance TCP throughput and energy efficiency with respect to the issues discussed above, we propose grafting a "probing" scheme onto the basic TCP error-control mechanism. In this scheme, a "Probe Cycle" consists of a structured exchange of "probe" segments between the sender and receiver. These segments carry no payload and are implemented using option extensions to the TCP header. 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 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 this paper we describe our implementation of TCP-Probing, and compare its throughput and energy efficiency with those of TCP Tahoe, Reno and New Reno. Although other versions of TCP have been recently proposed (e.g. TCP SACK), we have selected these three for two reasons:

(i) All three use the same acknowledgment strategy for successfully-transmitted segments, unlike TCP SACK for example. This enables uniform comparison of their mechanisms with TCP-Probing, which also uses the same strategy. Thus, differences in energy and throughput performance are a direct function of their distinct *congestion-control* strategies. We can therefore compare these strategies' ability to adjust transmission to varying error conditions, not the least of

[1] We have, however, been unable to corroborate some of their results and conclusions [11].

which is the ability to effectively utilize available bandwidth within the limits of both deteriorating *and* improving congestion levels. Any shortfall from this behavioral dynamic is reflected directly as additional energy consumption and/or degraded goodput.

(ii) Tahoe, Reno and New Reno represent two different approaches to congestion control, the one conservative (Tahoe) and the other more aggressive (Reno/New Reno). TCP-Probing dynamically traverses a spectrum of conservative-through-to-aggressive behavior in response to varying error conditions. The effectiveness of this flexibility is highlighted by the contrast in performance between Tahoe and Reno/New Reno, given that these use *fixed* conservative and aggressive strategies, respectively.

"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. Our TCP-Probing serves to demonstrate the validity and effectiveness of the concept, and to indicate the direction of further research.

Section 2 gives an overview of TCP Tahoe, Reno and New Reno. The design and implementation of TCP-Probing is detailed in Section 3. Section 4 outlines our testing environment and methodology, and presents our results along with an analysis of the protocols' behavior. In Section 5, we briefly discuss some open issues for further research. Finally, we close with some concluding remarks in Section 6.

2. TCP Overview

Historically, TCP Tahoe was the first modification to TCP. The newer TCP Reno included the Fast Recovery algorithm [1]. This was followed by New Reno [4] and the Partial Acknowledgment mechanism for multiple losses in a single window of data. As noted, TCP Tahoe, Reno and New Reno, all use basically the same algorithm at the receiver, but implement different variations of the transmission process at the sender. 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 an acknowledgment which includes the sequence number identifying the next in-sequence segment (byte). The sender implements a congestion window that defines the maximum number of transmitted-but-unacknowledged bytes permitted. This adaptive window can increase and decrease, but never exceeds the receiver's advertised window. TCP applies graduated multiplicative and additive increases to the sender's congestion window. The versions of the protocol differ from each other essentially in the way that the congestion window is manipulated in response to acknowledgments and timeouts.

TCP error-control mechanism is primarily oriented towards congestion control. Congestion control can be beneficial also for the flow that experiences it, since avoiding unnecessary retransmission can lead to better throughput [5]. The basic idea is for each source to determine how much capacity is available in the network, so that it knows how many segments it can safely have in transit. TCP utilizes acknowledgments to pace the transmission of segments and interprets timeout events as indicating congestion. In response, the TCP sender reduces the transmission rate by shrinking its window. Tahoe and Reno are the two most common reference implementations for TCP. New Reno is a modified version of Reno that attempts to solve some of Reno's performance problems when multiple packets are dropped from a single window of data.

2.1 TCP Tahoe.

The congestion-control algorithm includes Slow Start, Congestion Avoidance, and Fast Retransmit [1, 5]. 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.

2.2 TCP Reno.

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. This, in turn, implies that the congestion window (`cwnd`) should actually be incremented. Receiving the threshold number of *dack*s triggers Fast Recovery: the sender retransmits the missing segment, then, instead of entering Slow Start as in Tahoe, increases `cwnd` by the *dack* threshold number. Thereafter, and for as long as the sender remains in Fast Recovery, `cwnd` is increased by one for each additional *dack* received. This procedure is called "inflating" `cwnd`. The Fast Recovery stage is

completed when an acknowledgment (*ack)* for new data is received. The sender then halves `cwnd` ("deflating" the window), sets the congestion threshold to `cwnd`, and resets the *dack* counter. In Fast Recovery, `cwnd` is thus effectively set to half its previous value in the presence of *dack*s, rather than performing Slow Start as for a general retransmission timeout. Reno, however, is not optimized for multiple segment drops from a single window.

2.3 TCP New Reno.

New Reno [4] addresses the problem of multiple segment drops. In effect, it can avoid many of the retransmit timeouts of Reno. The New Reno modification introduces a partial acknowledgment strategy in Fast Recovery. 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. It is thus an indication that not all data sent before entering Fast Recovery has been received. In Reno, a partial *ack* causes exit from Fast Recovery. In New Reno it is an indication that (at least) one segment is missing and needs to be retransmitted. This retransmission is effectuated and Fast Recovery continues. In this way, when multiple segments are lost from a window of data, New Reno can recover without waiting for a retransmission timeout. However, the retransmission triggered off by a partial *ack* might be for a delayed rather than lost segment; thus, the strategy risks making multiple successful transmissions for the segment, which can seriously impact its energy efficiency with no compensatory gain in goodput.

3. TCP-Probing

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.

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 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 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 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 Figure 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.

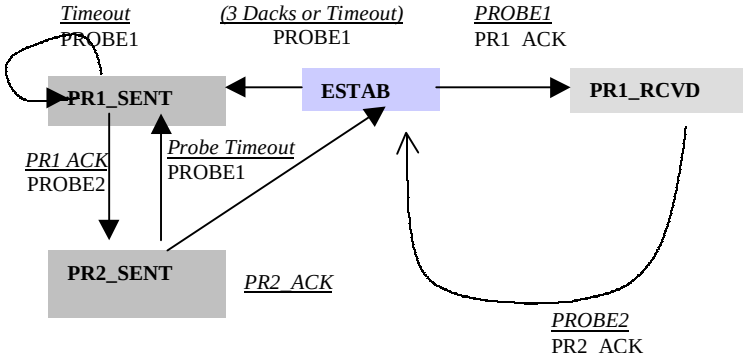


Figure 1: Probing State Transition Diagram

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.

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.

A critical part of the probing mechanism is the decision rules that determine action at the end of the probe cycle. The following simple algorithm proved sufficient to demonstrate the potential of probing in our experiments. Upon exiting the probe cycle, the two RTTs measured are compared. If both lie in the range [best RTT, last RTT], Immediate Recovery is applied. Last RTT is the estimated RTT value current at the time the probe cycle is triggered. Otherwise, the sender enters a Slow Start phase. More sophisticated decision criteria can be developed. For example, the value of each of the two RTTs with respect to the range [best RTT, last RTT], and the delay variation (jitter) between them, are potentially useful sources of information that we have not attempted to make use of.

Other issues need to be considered for the implementation of probing. For instance, the sender could receive *ack*s 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 *ack*s to the receiver's data. Duplicate acknowledgment delivery during a probe cycle is another issue that needs to be addressed. When a probe cycle is triggered because of a timeout, *dack*s are ignored. For a probe cycle triggered by *dack*s, however, we keep count of the total number of *dack*s. This count is used to increase the congestion window size if the ensuing phase is Immediate Recovery.

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.

4. Results and Discussion

The four versions of TCP were implemented (not simulated) as fully developed functioning protocol code using the x-kernel framework [12]. The tests were carried out 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 fairly low-bandwidth network environment since we are primarily interested in heterogeneous wired/wireless networks; and error conditions of different intensity and duration in order to evaluate the protocols' performance in response to changes in that environment.

In order to simulate the error conditions, we developed a new x-kernel "virtual protocol" [12], VDELDROP, which was configured between TCP and IP. VDELDROP'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.

In our experiments we configured the two states to have equal mean sojourn time. The value of this mean time varied from experiment to experiment, of course, but was always set equal for the two states. Furthermore, 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.

All tests were undertaken using 5-MByte (5,242,880 bytes) data sets for transmission. The results we report are based on the average of five replications for each test. 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 Table 1 (see Appendix). The table gives results for VDELDROP with mean On/Off phase duration set to 1 second (columns 1.1, 1.2, & 1.3) and 10 seconds (columns 2.1, 2.2, & 2.3). In order to represent the (transmission) energy expenditure overhead required to complete reliable transmission under different conditions, we use **Overhead** as a metric (columns 1.3 & 2.3). This is the total *extra* number of bytes the protocol transmits, expressed as a percentage, over and above the 5 MBytes 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 5 Mbytes data set for all tests.
- `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.

The `Connection Time` required to complete reliable transmission under different conditions, from connection initiation through to connection termination, is given in columns **Time** in seconds (1.1 & 2.1). The measured performance of the protocols is given in columns **Goodput** (1.2 & 2.2) in **bits**/sec., using the formula: `Goodput = Original Data / Connection Time`. For VDELDROP, the **DROP Rate** reported is the dropping rate for segments during the On phases, not the averaged overall drop rate across On/Off phases. An entry of **0** in the **DROP rate** column signifies error-free conditions. All charts presented in Sections 4.1 and 4.2 below are based on the data given in Table 1.

4.1 Effective Throughput Performance (Goodput)

As can be seen from Chart 1 below, Tahoe, Reno, New Reno and Probing exhibit somewhat similar behavior for frequent On/Off phase changes (1-second mean phase duration). Due to the short phase periods, the protocols do not get the opportunity to demonstrate the full range of their behavioral characteristics by expanding their window sizes and applying their congestion-/error-control algorithms over a sufficiently prolonged period of time. In particular, for example, when TCP-Probing enters Immediate Recovery, it does so with a window size that is not significantly different from Slow Start.

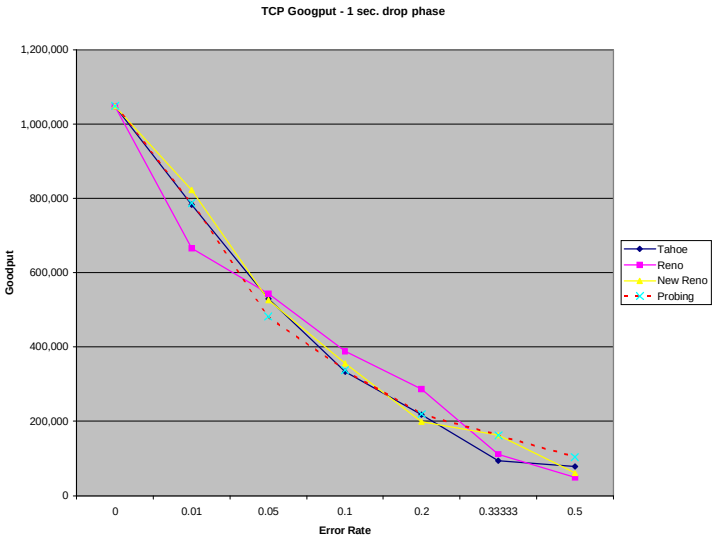


Chart 1: Goodput with mean On/Off phase 1 second

The situation, however, changes when the mean duration of the On/Off phase grows to 10 seconds. Chart 2 shows that none of the three standard versions of TCP uniformly outperforms the other two with respect to goodput across the full range of error rates from 0 to 50%. Reno's more aggressive strategy yields better results at lower error rates, while Tahoe's conservative strategy proves its worth at higher rates.

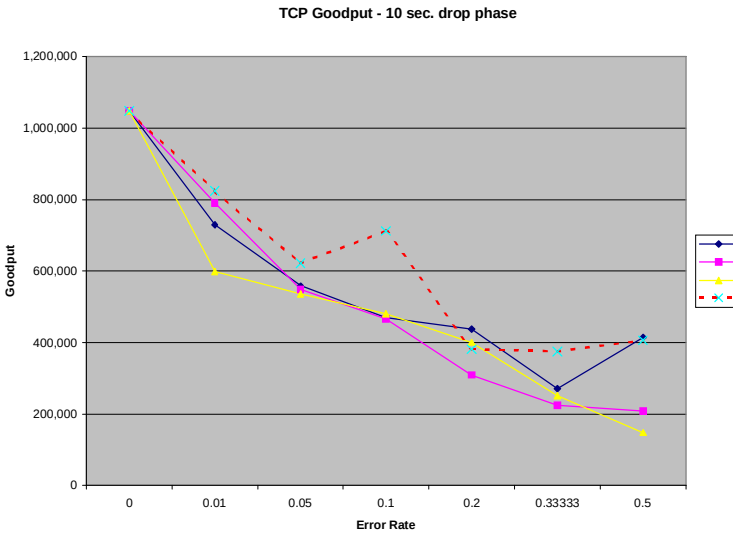


Chart 2: Goodput with mean On/Off phase 10 seconds

The salient point to note here is that, with only one exception at rate 20%, TCP-Probing's goodput is uniformly no worse than the *best* of the three standard versions across the entire range of error rates. As such, TCP-Probing yields an upper bound - sometimes a loose upper bound - to the standard versions' 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 (e.g. 1 second and 10 seconds). 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 appreciably under-perform the best of the three standard versions, and outperforms them for longer phase duration, across the range of configurations for the error model shown in Charts 1 and 2, gains further significance.

It is instructive to consider how the probing and Immediate Recovery mechanisms provide TCP-Probing with the behavioral flexibility underlying its performance in Chart 2. At relatively low error rates (0.01/1% to 0.1/10%), 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 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, Tahoe, Reno and New Reno's mechanisms are 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.

Probing gradually induces a distinctly different pattern of behavior from the above 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 leads to Slow Start rather than Immediate Recovery. TCP-Probing behavior thus becomes more conservative. This conservative bias is appropriate under the circumstances, as may be seen by contrasting it against Reno's and New Reno's more aggressive, fixed policies. Consider, for example, Reno at error rates 0.33 and 0.5 (Chart 2). It responds to some errors by entering a Fast Recovery phase and continuing relatively aggressive data transmission; it goes to Slow Start only in response to timeout events. Its window will have been expanded significantly during the Off phase. When it enters the On phase, relatively aggressive transmission could go on for a while, since graduated downward adjustments in Fast Recovery allow for data transmission until the window shrinks. Under the same scenario, New Reno could sustain aggressive retransmission with a large window for even longer than Reno does, since, in the absence of a timeout, it interprets *ack*s during Fast Recovery as partial acknowledgments. Such attempts to recover from multiple losses before readjusting its window are unlikely to be successful as the error phase persists. Extending the scenario to Tahoe, its conservative policy calls for the window to shrink immediately; it will not effectively re-expand for the remainder of the heavy error phase. As the error rate increases beyond 0.33, Reno and New Reno goodput continues to deteriorate, while Tahoe goodput improves, finally matching that of TCP-Probing at 0.5. Despite more bytes being injected into the network by Reno and New Reno, goodput achieved is barely half that of Tahoe and TCP-Probing (Table 1, column 2.2, rows 6.1-6.4). This confirms our expectation that a conservative mode of transmission favors goodput when error conditions are rather intensive, while an aggressive bias results in worse goodput performance due to retransmission and timeout adjustments. A consideration of protocol overhead in light of achieved goodput further confirms the validity of this point.

The overhead expended by Reno is 3.07% and 2.82% for 33% and 50% error rates, respectively (Table 1, column 2.3, rows 5.2 & 6.2). New Reno wastes even more effort on retransmission: 3.86% and 3.71%, respectively (column 2.3, rows 5.3 & 6.3).

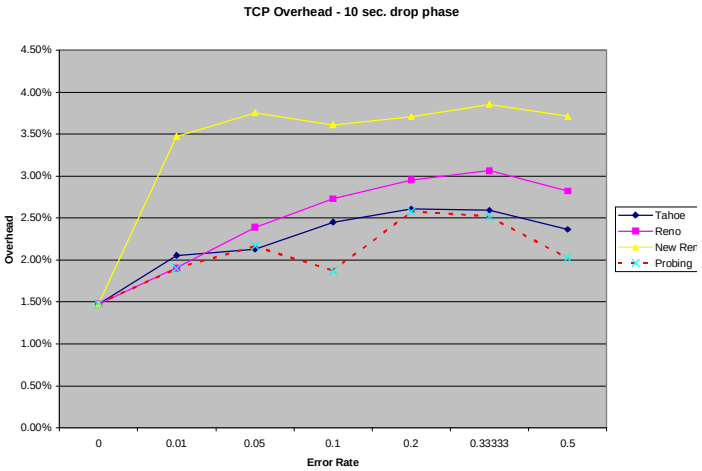


Chart 3: Overhead with mean phase 10 seconds

Tahoe's conservative retransmission policy reduces the overhead to 2.59% and 2.36%, respectively (column 2.3, rows 5.1 & 6.1). TCP-Probing, despite the transmission effort expended on probes, comes in even lower, with 2.52% and 2.02%, respectively (column 2.3, rows 5.4 & 6.4). Overhead expenditure across the full range of error rates is shown in Chart 3 above.

In conclusion, TCP-Probing outmatches Reno and New Reno in aggressive bias when an aggressive strategy yields better goodput. It successfully competes with Tahoe in being conservative when aggressiveness damages goodput. Yet it manages to demonstrate this flexible functionality using less transmission effort, which is a significant factor for energy expenditure.

4.2 Energy Issues

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 four 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,

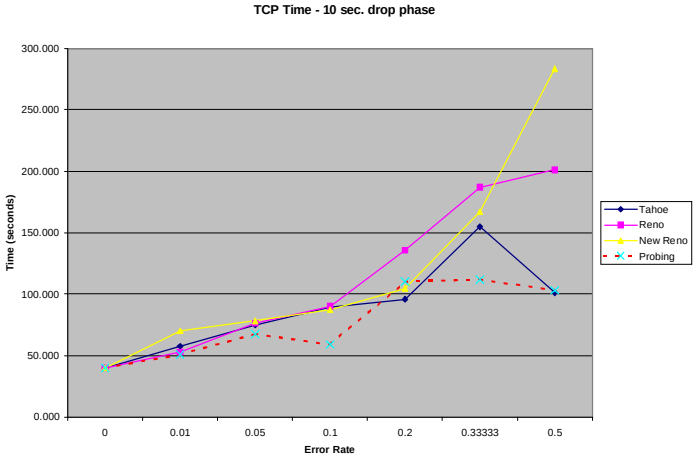
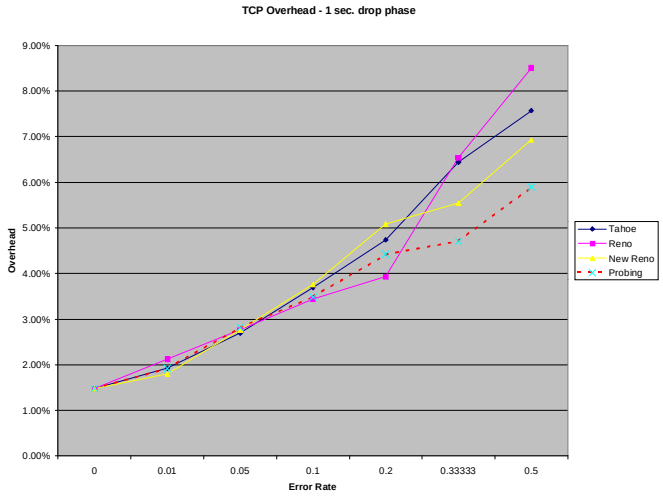


Chart 4: Connection time with mean On/Off phase 10 seconds

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 (discussed in the preceding section) and time savings achieved.

Chart 4 plots the data of column 2.1 in Table 1, giving the behavior of the protocols with respect to connection time for mean On/Off phase duration 10 seconds. TCP-Probing expends less time than the best of the three standard TCP versions, almost with no exception. The differences reach extremes of: only about 2/3 of the time needed by Reno, New Reno, and Tahoe at 33% error rate; and at 50% error rate, only about 1/2 and 1/3 of Reno's and New Reno's times, respectively. Even in the case of 1-second mean error phases, where TCP-Probing does not generally outperform Tahoe, Reno and New Reno, its associated overhead transmission effort and connection times (Charts 5 below), and hence goodput (Chart 1), are better at higher error rates (around 33% and up).



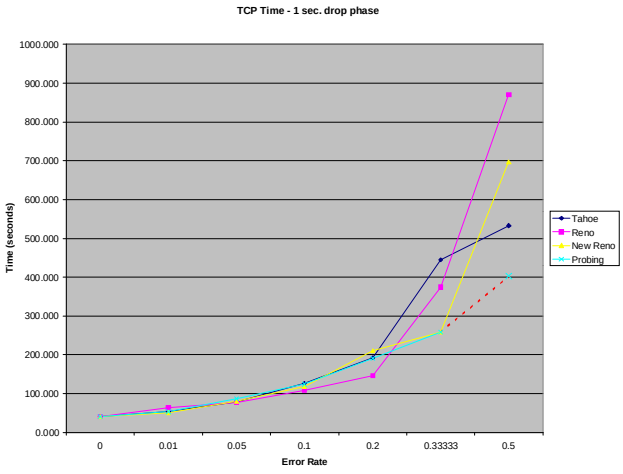


Chart 5: Overhead and Time with mean On/Off phase 1 second

At these higher rates, more energy needs to be expended due to the increasing number of lost segments. By the same token, any relative gains at these higher expenditure levels translates into greater energy savings in absolute terms. Clearly, TCP-Probing's performance under more intensive and/or prolonged error patterns renders it the energy-conserving protocol of choice.

5. 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 1-second phase tests indicate that the decision-making component of the probing mechanism is amenable to localized, heuristic improvement. The two tests at error rates 10% and 20% are noteworthy for the fact that TCP-Probing performance falls well below the best of the three standard TCP versions, which happens to be Reno in this case. An examination of the data in the Table shows that the shortfall comes not so much from extra overhead expended by TCP-Probing, but rather from extended connection times: TCP-Probing connection times come in very close to Tahoe's in both cases. At these error rates probing cycles are not particularly extended, so their impact on both overhead and connection times is minimal. The problem seems to be the unduly conservative decision-making criteria whereby Immediate Recovery is entered at the end of probing only if both probe-cycle RTTs are less than the last estimated RTT. Simply stated, TCP-Probing behavior is insufficiently aggressive under the circumstances. This is currently the subject of ongoing research.

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 10-second phase test at 20% error rate indicates that the protocol's adjustment mechanisms could be better calibrated. The dynamics of the energy/throughput tradeoff need to be further investigated.

Finally, TCP behavior under extended RTTs is worth investigating. It should be noted that TCP response is strictly governed by events that occur on an RTT time scale. As such, one would not anticipate dramatic changes in the relative performance of the four versions under consideration. Nevertheless, TCP with long RTTs might usefully serve to indicate the latitudes within which probing mechanisms could be amenable to further refinement and development.

6. 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 energy-conserving capabilities. The results presented in this paper serve to demonstrate the validity of the concept, and to provide directions for further research.

References

1. M. Allman, V. Paxson, W. Stevens, "TCP Congestion Control", RFC 2581, April 1999
2. H. Balakrishnan, V. Padmanabhan, S. Seshan, R. Katz, " A comparison of mechanisms for improving TCP performance over wireless links", *ACM/IEEE Transactions on Networking*, Dec. 1997.
3. A. Chockalingam, M. Zorzi, R. R. Rao, "Performance of TCP on Wireless Fading Links with memory", in *Proc. of IEEE ICC'98*, June 1998
4. S. Floyd, T. Henderson, "The New Reno Modification to TCP's Fast Recovery Algorithm", RFC 2582, April 1999.
5. V. Jacobson, "Congestion avoidance and control" in *Proc. of ACM SIGCOMM '88*, August 1988.
6. A. Kumar, "Comparative performance analysis of versions of TCP in a local network with a lossy link", *ACM/IEEE Transactions on Networking*, August 1998.
7. T. Lakshman, U. Madhow, "The performance of TCP/IP for networks with high bandwidth-delay products and random loss", *IEEE/ACM Transactions on Networking*, pp. 336-350, June 1997.
8. J. Postel, Transmission Control Protocol, RFC 793, September 1981

9. K. Ramakrishnan, S. Floyd, " A Proposal to add Explicit Congestion Notification (ECN) to IP", RFC 2481, January 1999.
10. V. Tsaoussidis, H. Badr, R. Verma, "Wave and Wait: An Energy-saving Transport Protocol for Mobile IP-Devices", in *Proc. of IEEE ICNP '99*, Toronto, Canada, Oct. 1999.
11. V. Tsaoussidis, H. Badr, "Energy / Throughput Tradeoffs of TCP Error Control Strategies", 5th IEEE Symposium on Computers and Communications, IEEE ISCC 2000, France, 2000.
12. TheX-kernel: http://www.cs.princeton.edu/xkernel
13. M. Zorzi, R. Rao, "Energy Efficiency of TCP", ACM Monet, Special Issue on Energy Conserving Protocols, January/February 2000.

Appendix: Tables: *Test Results with mean On/Off phase duration 1 second (columns 1.1, 1.2,& 1.3) and 10 seconds (columns 2.1, 2.2, & 2.3)*

Test	Protocol	Drop rate	Time	Goodput	Overhead	Time	Goodput	Overhead
			1.1	1.2	1.3	2.1	2.2	2.3
0.1	Tahoe	0	40.000	1,048,576	1.48%	40.00	1,048,576	1.48%
0.2	Reno	0	40.000	1,048,576	1.48%	40.00	1,048,576	1.48%
0.3	New Reno	0	40.000	1,048,576	1.48%	40.00	1,048,576	1.48%
0.4	Probing	0	40.000	1,048,576	1.48%	40.00	1,048,576	1.48%
1.1	Tahoe	0.01	53.591	782,650	1.92%	57.52	729,215	2.06%
1.2	Reno	0.01	62.970	666,079	2.13%	53.02	791,020	1.91%
1.3	New Reno	0.01	51.003	822,369	1.80%	70.02	598,996	3.48%
1.4	Probing	0.01	53.260	787,426	1.92%	50.87	824,514	1.90%
2.1	Tahoe	0.05	79.198	529,598	2.69%	75.06	558,766	2.13%
2.2	Reno	0.05	77.127	543,820	2.76%	76.42	548,857	2.39%
2.3	New Reno	0.05	79.734	526,037	2.76%	78.18	536,524	3.76%
2.4	Probing	0.05	81.002	517,802	2.82%	67.47	621,654	2.17%
3.1	Tahoe	0.1	126.023	332,822	3.70%	89.16	470,415	2.45%
3.2	Reno	0.1	107.944	388,561	3.44%	90.05	465,783	2.73%
3.3	New Reno	0.1	118.079	355,213	3.76%	87.35	480,162	3.61%
3.4	Probing	0.1	124.086	338,015	3.49%	58.87	712,468	1.87%
4.1	Tahoe	0.2	192.219	218,205	4.74%	95.81	437,796	2.61%
4.2	Reno	0.2	146.059	287,164	3.93%	135.55	309,421	2.96%
4.3	New Reno	0.2	210.476	199,277	5.09%	104.56	401,155	3.71%
4.4	Probing	0.2	191.146	219,482	4.42%	110.12	381,137	2.58%
5.1	Tahoe	0.33333	445.242	94,203	6.44%	154.67	271,176	2.59%
5.2	Reno	0.33333	374.421	112,021	6.54%	186.83	224,498	3.07%
5.3	New Reno	0.33333	257.884	162,643	5.55%	166.89	251,326	3.86%
5.4	Probing	0.33333	258.228	162,443	4.71%	111.88	374,893	2.52%
6.1	Tahoe	0.5	532.897	78,708	7.57%	101.01	415,247	2.36%
6.2	Reno	0.5	870.334	48,192	8.51%	201.18	208,488	2.82%
6.3	New Reno	0.5	697.130	60,165	6.93%	283.44	147,977	3.71%
6.4	Probing	0.5	402.877	104,128	5.90%	103.15	406,621	2.02%