The Journal of Supercomputing, 20, 115–135, 2001

The Wave & Probe Communication Mechanisms

VASSILIS TSAOUSSIDIS, A. LAHANAS AND C. ZHANG {vassilis,ladrian.czhang}@ccs.neu.edu
Computer Science, Northeastern University, Boston, MA 02115

Abstract. This paper is motivated by the modifications recently suggested to enhance TCP performance over wireless channels. We argue that TCP end-to-end error-control mechanism lacks the functionality to respond appropriately in situations where errors vary in nature, frequency, or duration. As a result, this mechanism could, under some circumstances, yield zero throughput achievements at high energy-cost, or degrade throughput performance without conserving energy. This incompetent behavior with respect to the energy/throughput tradeoff puts in question TCP's suitability as a universal, reliable transport protocol of choice, especially for battery-powered mobile devices for which energy is a critical resource and congestion is not the exclusive cause of errors.

We propose "Wave" and "Probing" communication and control mechanisms that permit end-to-end protocols to detect congestion without necessarily experiencing packet drops, to distinguish random and burst errors from congestion, and, as conditions vary, to rapidly adjust the transmission window upwards or downwards depending on the nature of the error. We report extensively on the performance of these new mechanisms to demonstrate their energy-conserving and high-throughput capabilities.

Keywords: transport protocols, energy saving, error control, mobile computing

1. Introduction

Mobile computing requires a universal solution for error control, combining energy efficiency and high throughput capacity over heterogeneous networks with both wireless and wired components. 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. Most transport protocols such as TCP detect 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. While the net outcome of the recovery process has to be the retransmission of the missing segments, the *nature* of the error actually should play a determining role in defining the recovery strategy to be used. When network conditions deteriorate to an extent that they become the ground for more-or-less persistent error conditions, a back-off strategy seems to be the correct choice for the sender. On the other hand, conditions of random and short or infrequent burst errors could require an aggressive behavior instead. In other words, error frequency and duration are two important characteristics that need to be ascertained during the error detection process in order to determine the appropriate recovery strategy.

This work addresses these concerns by proposing "Wave" and "Probing" communication and control mechanisms that permit detection of congestion without necessarily entailing packet drops, ascertain error frequency and duration, and, as conditions vary, rapidly adjust the transmission window upwards or downwards depending on the nature of the error. In Section 2 we review recent modifications and enhancements of TCP, and discuss the energy/throughput limitations inherent to its error control mechanism. In Section 3 we present our proposed mechanisms and describe their implementation in the Wave & Wait Protocol (WWP) [11, 12], an experimental protocol based on Waves and Probing. Section 4 presents extensive comparative results on TCP and WWP with respect to such characteristics as: performance under combinations of random, burst, short, and persistent errors; the impact of energy expenditure on throughput under various error conditions; and the energy/throughput tradeoff. Finally, Section 5 presents some concluding remarks.

2. TCP overview

Reliable protocols such as TCP [9] are tuned to perform well in traditional wired networks where transmission losses occur mostly due to congestion. 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 byte expected in correct sequence. 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 never exceeds the receiver's advertised window.

TCP has four more-or-less standard algorithms to perform the congestion control operations described above: Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery [1]. They differ from each other essentially in the way that the congestion window is manipulated in response to acknowledgments and timeouts, and the manner in which delivered or missing segments determine action. The initial single-segment congestion window effectively grows exponentially (slow start) until a threshold is reached. Beyond that point additive increase (congestion avoidance) takes over. When packets are lost and retransmission timeout event occurs, the cwnd is set to double the maximum segment size, and the congestion window threshold is set to half the window size that existed prior to the timeout event. In Fast Retransmit, a number (normally 3) of successive duplicate acknowledgements (dacks), a suggestion of segment loss, trigger off a retransmission without waiting for the associated timeout event to occur. After that, slow start is applied. TCP Reno introduces Fast Recovery [1] in conjunction with Fast Retransmit. Fast Recovery sets the congestion window to half the value prior to Fast Retransmit, rather than performing Slow Start, after the retransmitted segment gets acknowledged.

Recently, TCP behavior over wireless and wired networks has been a focus of attention. New approaches or modifications have been proposed. Some use essentially the standard algorithm(s) at the receiver, but implement different variations of the transmission process at the sender. Others, however, propose variations of the acknowledgment strategy at the receiver. TCP's behavior over wired networks, where congestion is a regular cause for packet loss, was initially studied by Jacobson [5]. Later, modifications were suggested to upgrade the protocol's throughput performance by adopting occasionally more aggressive retransmissions, delaying acknowledgments when congestion appears to be the problem, applying a conservative retransmission algorithm when energy consumption is a significant concern, or modifying the functionality of intermediate devices (routers, base station) in order to assist the two ends in efficiently monitoring current network conditions without undergoing packet drops. Recent research results [2, 3, 6, 7] have, in fact, shown that TCP throughput degrades, in the presence of the kind of random and burst errors typical of wireless environments. However, the enhancements proposed [2, 10], 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 [10] propose an Explicit Congestion Notification capability to be added to the IP protocol (an approach similar to RED Gateways) in order to trigger appropriate behavior of TCP congestion control, and possibly enhance its performance by explicitly notifying it of the developing 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. Floyd and Henderson [4] propose a partial acknowledgment method to enhance performance of the TCP Fast Recovery algorithm which, under rather specific conditions, results in some improvement [13].

The work discussed in the preceding paragraph does not, however, address energy issues. Reference [15] is virtually the only research currently available, which explicitly does address TCP energy consumption. The authors investigate and report on the throughput and energy consumption of various versions of TCP under random and correlated (burst) errors. They conclude that, with the correct parameter settings for maximum window size and fast retransmit threshold, TCP Tahoe, in particular, does a good job at saving energy because it backs off in the presence of burst errors. Their work leaves for future investigation the important issue of the energy efficiency tradeoffs involved when backing off increases delays, and hence the overall connection time. Since the energy consumption of TCP has not been studied in any context outside of the comparative performance of the various versions of TCP itself, it is not yet quite clear how successful is TCP's job at saving energy. The basic question, after all, is not so much how energy can be saved in the context of this or that protocol, but rather how can it be efficiently expended to achieve high throughput. Although one might naively expect that expending more energy (in the form of transmission power) should result in better throughput, [11, 12, 13] show

that this is not always true. Energy and throughput do not necessarily constitute a zero-sum tradeoff.

TCP displays some undesirable patterns of behavior in the context of efficient energy expenditure that aspires to high throughput. The error recovery mechanism is not always efficient, especially for wireless networks, 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 retransmissions at the cost of unnecessary and significantly degraded throughput, and increases overall connection time, as our test results will show. Yet, when an error occurs and TCP does back off, it continues to forcefully attempt transmissions within the constraints of the reduced window size. In the presence of errors of a relatively persistent nature (fading channel, prolonged and frequent burst errors), this behavior does not favor energy-saving since it yields only minor throughput improvement at high cost in transmission energy. In summary, from the perspective of energy expenditure in the context of mobile networks, TCP would seem to possess an inherent tendency to back off too much when it should not, and too little when it should. 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. In addition, the protocol lacks the ability to efficiently monitor network conditions, rapidly readjust its window size in response to changes in these conditions, and detect congestion without inducing packet drops, thereby degrading overall performance through additional retransmissions and wasted opportunities in maintaining the communication pipe full. In order to efficiently expend energy that achieves high throughput, an end-to-end protocol would need to possess two broad characteristics: 1. The ability to efficiently monitor the network on an end-to-end basis, and determine the *nature* of errors. This capability, if it is to be energy-conserving, must depend on something other than the traditional approach of attempting costly data transmissions and seeing if they get through within a certain maximum round trip time (RTT). 2. The flexibility to rapidly adjust the sender's transmission window upwards or downwards in response to detected error conditions.

3. Waves and probing

We present two mechanisms, "Waves" and "Probing," which together provide the necessary functionality outlined at the end of the previous section. We also describe the implementation of these mechanisms in the current version of the Wave and Wait Protocol (WWP). WWP is an experimental protocol we have developed as a test-bed transport-level protocol, running on top of IP and based on Wave and Probing mechanisms. Full details of WWP may be found in [11, 12]; here we shall focus specifically on the implementation of Waves and Probing in the protocol.

3.1. Waves

When network conditions appear acceptable and data transmission can be efficiently undertaken, the receiver could use the successive segments reaching it to effectively monitor network conditions, if it had some knowledge of the sender's transmission pattern of these segments. Furthermore, the receiver could then instruct the sender to readjust its transmission window upwards or downwards in response to what it has learned. The receiver could even request that the sender suspend transmissions altogether if it concludes that conditions have deteriorated beyond the point where energy-conserving transmission is possible for the present (*i.e.*, there is a high risk of too many data segments being lost, necessitating too many retransmissions). In order to provide this kind of capability, we propose that transmission between sender and receiver take place in "waves" of fixed-sized data segments.

A wave is group of fixed-sized data segments of predetermined number. The mechanism defines various levels of waves, where the higher the level, the more data segments that the wave comprises (*i.e.*, the larger the current sending window). The sender groups data segments into a wave at the appropriate level, determined by the receiver's experience of prevailing network conditions, and then transmits these segments one after the other, with no pause between one segment and the next. It then pauses, awaiting a response from the receiver. Cognizant of the current wave level, the receiver implicitly has full knowledge of the sender's transmission pattern and can proceed by notifying the sender of missing segments from the current wave, and, as outlined above, setting the next wave level based on monitored network conditions.

The wave mechanism provides a very flexible capability to rapidly adjust the sender's window in response to monitored conditions. It enables swift, dynamic readjustment along a wide spectrum of conservative-through-to-aggressive transmission behavior, depending on the number of wave levels implemented, the predetermined number of data segments at each wave level, the size adopted for the fixed-sized data segments, and the decision criteria according to which the receiver determines the next wave level.

3.2. Implementation of waves in WWP

As already mentioned, the receiver attempts to estimate prevailing congestion conditions by monitoring the throughput of the current wave and setting the level of the next wave accordingly. A wave at level i ($i >= 0$) is composed of a fixed number $W(i)$ of data segments. For $i = 0$, $W(0)$ is defined to be 0, signifying that no further data transmission is to be attempted for now.

In WWP all segments (data and control) have a fixed-sized 6-byte header. A data segment also carries a fixed-sized 1 Kbyte data payload. Once the first segment to reach the receiver from a new wave arrives, it is easy for the receiver, given the current wave level i (which is carried in the segment header), to calculate how long it would take the rest of the wave to reach it if the network were relatively uncongested, using a "*baseline*" throughput of BT KBytes per second for the

uncongested network. The time thus calculated is the "*baseline time*." The receiver measures how long it actually takes for the remaining segments in the wave to arrive. It then uses the baseline and measured times for the wave to set the level of the next wave.

The baseline throughput BT is a protocol parameter whose value can be determined empirically for the network on which the protocol is running, so as to maximize protocol performance in line with the application's throughput needs and the amount of energy-saving that it is willing to trade off in return for higher throughput. BT would typically be set at some fraction of the (average) maximum effective throughput the network is capable of under congestion-free conditions. The exact value would, in general, depend on how stable general network conditions are, as well as on the inherent throughput capabilities of the network. The closer the value set for BT is to the network's maximum effective throughput, the less aggressive the protocol will be in attempting transmissions, since it will attempt transmissions at higher wave levels only when current network throughput gets close to the maximum possible, and there will be a strong bias towards selecting lower wave levels, including wave level zero, as the current throughput falls below that maximum. In a relatively stable and uncongested network environment, it would probably pay off to be somewhat more aggressive, and so BT should be set lower. In a relatively congested network with rapidly varying conditions, BT should be set higher to make the protocol's behavior more conservative and less aggressive. However, a wide range of wave levels would allow more flexibility than a single parameter such as BT is, by itself, capable of providing, permitting the protocol to automatically adjust its behavior across a broader variety of network environments, as well as to accommodate more variability in the operational characteristics of any single environment. Our current WWP implementation calls for four wave levels, $i = 0, 1, 2, 3$, with the number of segments in a wave set at $W(i) = (12 \times i)$ for $i = 0, 1, 2, 3$; fixed-sized segment payload is set to 1 KByte. The following simple algorithm is used by the receiver to set the next wave level: Suppose the current wave is at level i, $i = 1, 2, 3$.

Let $\mathbf{T(i)}$ be the measured time for a level i wave.
Let $\mathbf{B(i)}$ be the baseline time for a level i wave.
For j = 1, 2, 3:
  if $\mathbf{T(i)}$ is in the range $(\mathbf{j-1}, \mathbf{j}] \times \mathbf{B(i)}$,
  then next wave level is set to $(\mathbf{4-j})$;
  else set wave-level to **0**.

The algorithm essentially implies that when the receiver sets the new wave level to $\mathbf{k}$, $k = 1, 2, 3$, it is estimating the current network throughput to be no worse than approximately a fraction $1/(4-k)$ of the baseline throughput value of BT KBytes per second (and no better than approximately a fraction $1/(3-k)$, for $k = 1$ or 2). The number of segments in the new wave is then adjusted proportionately. If the throughput appears to be less than 1/3 of the baseline throughput, we go to level 0, deeming it better to pause for a while than risk expending energy transmitting

even a small wave that might not have a sufficiently good chance of getting through undamaged.

Upon receiving a complete wave, the receiver responds by transmitting a "N-S_ACK" segment to the sender. N-S_ACKs are control segments whose variable-length payload field identifies all data segments to date that need retransmitting. The header identifies the level set by the receiver for the next wave. A timeout mechanism at the receiver ensures that the N-S_ACK is still sent when unduly delayed or lost data segments make for only an incomplete wave reaching the receiver. The current implementation of WWP calls for the next wave level to be set to 0 in this case.

The sender does not start transmitting data segments until it has a sufficient number to make up a complete wave. When it receives a N-S_ACK setting the wave level, and if it does not have sufficient old (needing retransmission) and new data up to the specified number of segments in the wave, it will transmit the segments it has at the highest wave level for which it has enough segments. Timeout mechanisms at the sender side take care of instances where the N-S_ACK is excessively delayed, and possibly lost, as described in 3.4 below. In the event the receiver sets the next wave level at 0, the sender will immediately probe the receiver. The receiver uses the RTTs (Round Trip Times) measured during the probe cycle to set the new wave level in a N-S_ACK segment it sends to the sender at the end of the cycle. The probing mechanism is dealt with in the next two subsections.

3.3. 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 RTT measurements from the network. The sender would initiate the cycle in response to the receiver's notification 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 in response to the last wave sent, and so on).

Probe cycles provide an "energy-saving" mechanism whereby instances in which good prevailing network conditions appear to be deteriorating can be investigated. They enable continuous, efficient, end-to-end monitoring of the network in order to exploit windows of opportunity of improved conditions during which transmission of data at an appropriate wave level can be successfully resumed. They are "energy-saving" in contrast to the energy that would otherwise be expended on the transmission of data segments that do not have a good chance of getting through during periods of degraded network conditions. Operating at even aggressive wave levels, and pausing to check with probes in the event of unduly delayed, and possibly lost segments, the receiver can decide whether to have data transmissions continue at an appropriately aggressive level, adjust the wave level downwards, or even temporarily back off data transmission altogether.

3.4. WWP implementation of probe cycles

A probe cycle in the current version of WWP aims at permitting the receiver to measure two successive RTTs from the network. It makes use of three short probing (control) segments (PROBE1, PROBE2 and PROBE3) and acknowledgments corresponding to the first two of these (PR1_ACK and PR2_ACK). All five segments carry no payload, consisting only of the 6-byte segment headers. A N-S_ACK is used as the acknowledgment to the PROBE3 segment, and signifies the successful termination of the probe cycle.

A probe cycle is initiated under one of two conditions. Firstly, if the N-S_ACK segment for the data wave just transmitted sets the next wave level at 0, the sender initiates a probe cycle by transmitting a PROBE1 segment. Secondly and alternatively, if the N-S_ACK segment for the data wave goes absent (see Figure 1). When the sender finishes transmission of the data wave, it sets a "SEND_T" timeout which is canceled upon receipt of the wave's N-S_ACK. If the SEND_T timer expires, the sender initiates the probe cycle by transmitting a PROBE1 segment. Either way, the receiver responds to the PROBE1 with a PR1_ACK, upon receipt of which the sender transmits a PROBE2. The receiver acknowledges this second probe with a PR2_ACK and enters a state where it waits for a PROBE3. It makes an RTT measurement based on the time delay between sending the PR1_ACK and receiving the PROBE2. Upon receipt of PROBE3 it makes the second RTT measurement based on the time delay between the PR2_ACK and the PROBE3. The receiver then determines the level for the next wave and informs the sender by means of a N-S_ACK, whose payload also identifies all data segments up to this point of time that need retransmitting. In setting the next wave level, the receiver takes into

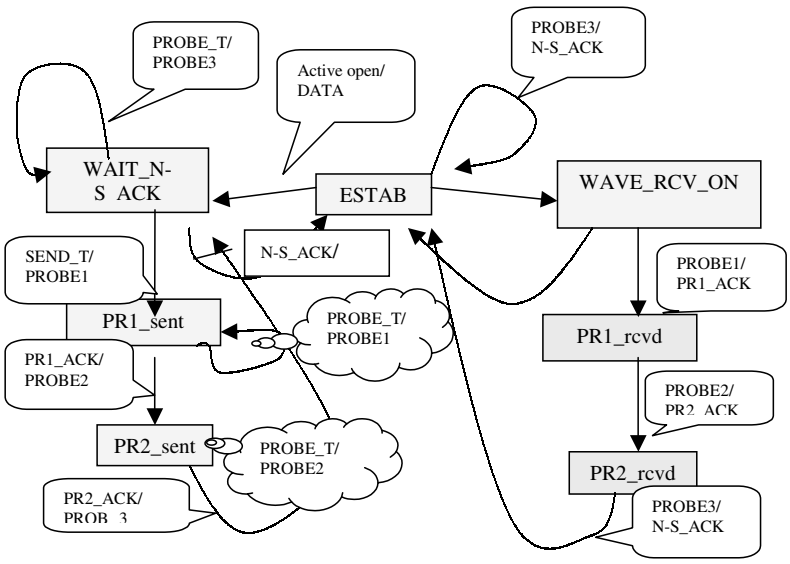

Figure 1. Probing: state transition diagram.

account the network conditions that had been detected at the time the current wave level was determined, as well as the values of the two RTT measurements just made and the delay variation (jitter) between them. Other applications with bursty flows that are currently sharing the same links of the network that our application is being routed along will induce measurable jitter between the two RTTs. Contrariwise, an error free environment, or links that are being shared with normalized/smoothed-out data flows, will induce no jitter. The receiver can take all this into account in setting the level for the next wave. The full set of decision-making rules has not yet been completely standardized, but a set of rules has been developed, implemented and calibrated in the WWP versions used in our tests.

In the event that the PROBE1 or its acknowledgment is lost, the sender, using a PROBE_T timeout, retransmits the PROBE1. The receiver reinitializes its measurement timer upon the receipt of the retransmitted PROBE1 in order to take the correct RTT measurement. Distinct sequence numbers are used to distinguish between different instances of multiply-retransmitted PROBE1 segments. A PR1_ACK carries the same sequence number as the corresponding PROBE1 instance that it is acknowledging. This sequence number is echoed back in the corresponding PROBE2. A similar process takes place with respect to: (i) using PROBE_T timeouts on the sender side for retransmissions of PROBE2 and PROBE3 segments in the event that either or both, or their corresponding PR2_ACKs and N-S_ACKs, are lost; and (ii) use of distinct sequence numbers for retransmissions, which are echoed back and forth in the various exchanges between sender and receiver so that associated segments can be correctly identified and paired off, and RTTs correctly measured. The receiver moves to the ESTAB state after sending the N-S_ACK that should terminate the probe cycle. In this state, and should the N-S_ACK be lost, the receiver would receive—instead of data segments from the next wave, or a PROBE1 initiating the next probe cycle if the receiver had specified wave level 0 in the N-S_ACK—it would receive a retransmitted PROBE3. The N-S_ACK was lost (or excessively delayed), and the sender, which is in state WAIT_N-S_ACK, timed out on the PROBE_T timer and resent the PROBE3. The receiver would then retransmit the N-S_ACK, echoing back the sequence number of the retransmitted PROBE3.

Although probing is a fairly complicated mechanism and adds additional RTTs to the protocol's progress, it 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 (*i.e.*, reducing the wave level) 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 effective throughput and also, by unnecessarily prolonging the connection time, impact energy consumption.

4. Testing environment and methodology

We ran tests simulating a fairly low-bandwidth environment using the *x*-kernel protocol framework [14] and the recommendations of [8]. The tests were carried out in a single session, with both client and server running on a single network

segment, so as to avoid unpredictable conditions with distorting effects on TCP's and WWP's performance. TCP Reno was used in the experiments.

Error conditions were simulated by dropping and delaying segments using modified x-kernel protocols. A "virtual protocol," VDELDROP, was configured between the transport protocols (TCP and WWP) and IP. VDELDROP's core mechanism consists of a two state (On/Off) continuous time Markov chain. One state was always configured with a zero error rate. The protocol PHASE_DROP (P_D) drops segments at a constant rate specified for the duration of a test, and causes different delays for each segment. PHASE_DROP also has the capability of alternating On/Off phases during which its drop actions are in effect and are suspended, respectively. Error conditions of varying intensity, persistence and duration could thus be simulated, depending on the choice of the drop rate and phase duration. Thus, during a connection period, WWP and TCP would experience phases that are error free and others with simulated error effects. This modification of the original VDROP protocol of the x-kernel enabled us to test their behavior in response to sudden changes in the simulated environment, and their ability to rapidly re-adapt to varying error conditions. Such conditions are typical of mobile networks where the user is "on the move": communication with the access points will have variable characteristics during the connection time. The high-level testing ("application") protocol configured above that sends messages of 1024 bytes to the underlying transport layer.

Note that the WWP implementation makes no comprehensive attempt to calibrate the various protocol parameters (data segment size, number of segments per wave at each wave level, baseline throughput BT, *etc*.) for optimal performance with respect to the overall characteristics of the protocol's operational environment. The data segment payload was 1 KByte; $W(i) = 12 \times i$, $i = 0, 1, 2, 3$; BT was set at 40 KByte/second (which is about 32% of the best effective throughput achieved under error-free conditions). All this probably causes WWP to understate its potential somewhat, though some effort was put into calibrating the sender timeout values SEND_T and PROBE_T, and the decision-making process by which the next wave level was determined at the end of a probe cycle, in order to enhance performance.

4.1. Test results

Tests are undertaken using 2- and 5-MByte data sets for transmission. The purpose of the tests was to evaluate the behavior of the two protocols in response to changes in the simulated network environment, such as congestion and transmission errors at different rates and of different duration. We took measurements of the total connection time and the total number of bytes transmitted (*i.e.*, including protocol control overhead transmissions, data segment retransmissions, *etc.*). Both of these factors significantly affect energy expenditure as well as throughput. Error conditions have various distinct characteristics—transient random errors, short burst errors, persistent and sustained errors, excessive congestion—but one and the same result: segments are lost. A challenge for both protocols was whether, in the presence of

Table 1. 10-second On/Off error phases

Test #	Prtcl	P_D Rate (%)	Time Ran	Total Bytes	Time Overhead (sec)	Tr. En. Wastage	Effective Throughput
1.1	TCP	0	69	5319120	—	76240B	75983
1.2	WWP	0	40.5	5268495	—	25615B	129453
2.1	TCP	5	117.4	5395782	48.4	152902B	44685
2.2	WWP	5	71	5290519	30.5	47639	73843
3.1	TCP	10	132.9	5409384	63.9	166504	39449
3.2	WWP	10	81.9	5295859	41.4	52979	64015
4.1	TCP	20	136.63	5410822	67.63	167942	38372
4.2	WWP	20	84.08	5319556	43.58	76676	62355
5.1	TCP	33	258.89	5392682	189.89	149802	20251
5.2	WWP	33	85.9	5324601	45.4	81721	61034
6.1	TCP	50	259.38	5388348	190.38	145468	20213
6.2	WWP	50	86.7	5275294	46.2	32414	60471

errors, they respond in a manner appropriate to, and compatible with, the *nature* of the error.

Below we present five tables of results from three distinct sets of tests. The first set reports on the impact of the protocols' error-control mechanisms on energy and throughput (Tables 1 and 2). Time periods for error-free and error-prone phases were sufficiently large so as to permit both protocols to stabilize their behavior when conditions change. Measurements of energy and throughput efficiency are summarized in Table 3.

Results from the second test set, presented in Table 4, are based on shorter On/Off periods in order to assess the ability of the error/control mechanisms to rapidly re-adjust to network changes and exploit windows of opportunities for error-free transmissions.

The last test set explores the energy/throughput tradeoff (Table 5). Two versions of WWP are used: a conservative version (which is the one used in the tests of Tables 1–4), and a more aggressive one. The aim here is to investigate conditions under which: an aggressive transmission strategy yields only minor throughput

Table 2. 4-second On/Off error phases

Test#	Prtcl	P_D Rate (%)	P_D Phase (sec)	Time Ran	Total Bytes	Time Overhead	Tr. En. Wastage	Effective Throughput
1.1	TCP	0	0	69	5319120	—	76240B	75983
1.2	WWP	0	0	40.5	5268495	—	25615B	129453
2.1	TCP	10	4	135.67	5409384	66.67	166504	38644
2.2	WWP	10	4	81.3	5306219	40.8	63339	64488
3.1	TCP	33	4	368.95	5502300	299.05	259420	14210
3.2	WWP	33	4	87.56	5344200	47.06	101320	59877
4.1	TCP	50	4	385.79	5514338	316.79	271458	13589
4.2	WWP	50	4	93.9	5378366	53.4	135486	55834

Table 3. Comparative energy & throughput results for TCP and WWP

T #	Prtcl	P_D Rate (%)	Time Ran (sec)	Energy Savings	Time Savings (sec)	Energy Expend. Ratio (TCP/WWP)	Throughput Ratio (TCP/WWP) (%)	Time Ratio (TCP/WWP)
1.1	TCP	0	69	50 KB	28.5	2.97/1	58.6	1.70/1
1.2	WWP	0	40.5					
2.1	TCP	10	132–135	103–113 KB	19.6–22.5	2.62–3.14/1	59–60	1.62–1.64/1
2.2	WWP	10	81.3–81.9					
3.1	TCP	33	258–368	68–158 KB	173–281	1.83–2.56/1	23–32	3.0–4.2/1
3.2	WWP	33	85.9–87.6					
4.1	TCP	50	259–385	113–135 KB	172–291	2.0–4.48/1	24–33	2.98–4.1/1
4.2	WWP	50	86.7–93.7					

Table 4. Protocol behavior under rapid network changes

Rate (%)	On/Off (sec)	TCP time	WWP time	TCP Bytes	WWP Bytes	Time Ratio
						WWP/TCP=
0	0	25.6	15.6	2133636	2107402	0.61
10	1	63.2	28.58	2192524	2137438	0.45
33	1	132.89	31.7	2245877	2202410	0.23
50	1	223.26	37.42	2330992	2304361	0.16
100	1	985.8	39.6	2305088	2296194	0.04
100	2	93.1	32.3	2189628	2158208	0.34
50	2	109.4	31.8	2192574	2168433	0.29
33	2	85.07	32.7	2189648	2163338	0.38
33	5	81.9	32.9	2172392	2124361	0.40

improvements and so wastes energy in transmission attempts; and, a conservative strategy yields only minor energy savings thereby unnecessarily degrading throughput with no compensatory benefit.

4.1.1. The impact of error control on energy and throughput. In Table 1 and Figure 2 below we present results for the 5-MByte data set with On/Off phase duration of 10 seconds. In order to represent the energy expenditure overhead required to complete reliable transmission under different conditions, we use the Byte Overhead as a metric. This is the total *extra* number of bytes the protocol transmits, over and above the 5 MBytes delivered to the application at the receiver, from connection initiation through to connection termination. The Byte Overhead is thus given by the formula: Byte Overhead = Total − Base, where,

- Base is the number of bytes delivered to the high-level protocol at the receiver, and is given in the column **Orig. Data** of the table. It is a fixed 5 MBytes.
- Total is the total of all bytes transmitted by the transport layers, and is given in the column **Total Bytes**. This includes protocol control overhead, data segment retransmission, as well as the delivered data.

Results are reported in the column **Tr. En. Wastage** (Transmission Energy Wastage).

The time overhead required to complete reliable transmission is given in column **Time Overhead** using the formula: Time Overhead = Connection Time − Base, where,

- Base is the number of seconds required to deliver the 5 MBytes to the high level protocol at the receiver under error-free conditions, from connection initiation through to connection termination (see column **Time Ran** for the tests 1.1 and 1.2).
- Connection Time is the corresponding amount of time required for completion of data delivery under error-prone conditions, and is given in column **Time Ran**.

Table 5. Aggressive and conservative versions of WWP

		Aggressive						Conservative					
Rate (%)	Phase (sec)	Data		Time		Throughput		Data		Time		Throughput	
0	0	5268495		40.5		129453		5268495		40.5		129453	
5	2	50876	+1%	68.9	+56.8%	76465	−41%	46995	0.8%	75	+85%	70246	−45.7%
10	2	86941	+1.6%	72.5	+79%	72668	−43.9%	65662	+1.2%	79	+95%	66689	−48.5%
20	2	175435	+3.3%	73	+80%	72171	−44.25%	85333	+1.6%	80	+97%	65856	−49.1%
33	2	272276	+5.1%	80.7	+99%	65284	−49.57%	114195	+2.1%	81.3	+100%	64803	−50%
50	2	415587	+7.9%	95.3	+135%	55283	−57.3%	159731	+3%	86.2	+112%	61119	−52.8%
100	2	207544	+3.9%	80.5	+98.7%	65447	−49.4%	245288	+5%	94.2	+132%	55928	−56.8%

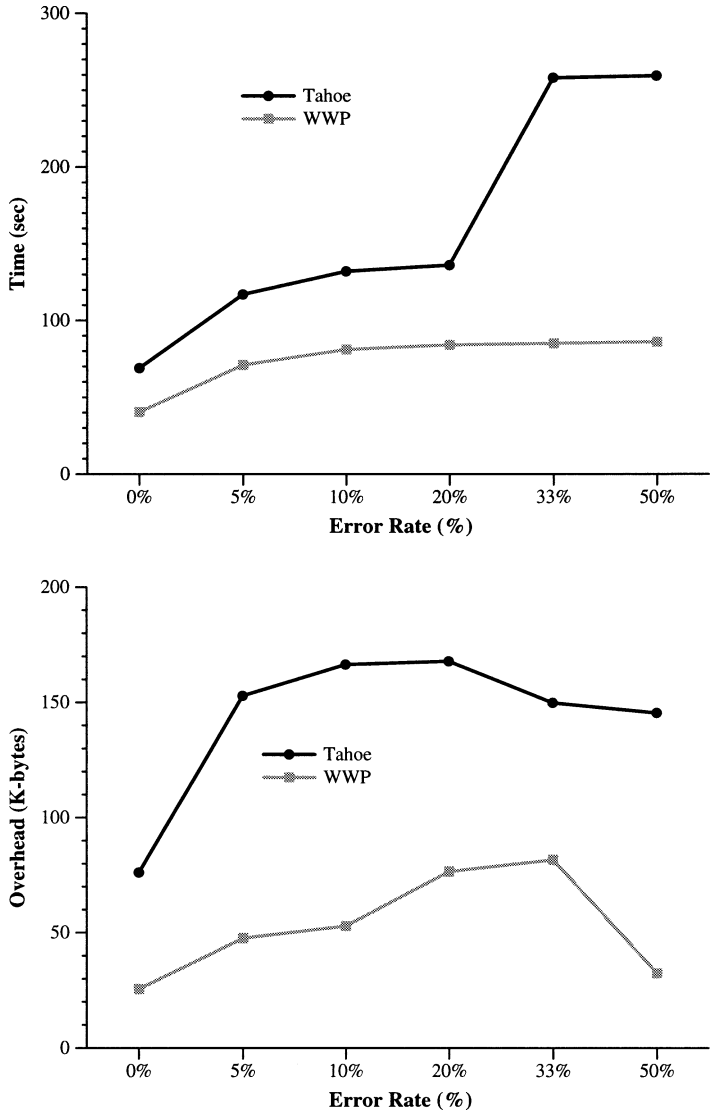

Figure 2. Connection time and overhead with 10-second On/Off error phases.

The effective throughput of the protocols is given in column **Effective Throughput** using the formula: Effective Throughput = Original Data/Connection Time, where,

- Original Data is the 5-MBytes data set (column **Orig. Data**).
- Connection Time is the amount of time required for completion of data delivery, from connection initiation to connection termination (column **Time Ran**).

On and Off phases are of equal duration, given in the column **P_D Phase**. The **P_D Rate** reported is the dropping rate for segments during the On phases, not

the averaged overall drop rate across On/Off phases. Entries of **0** in both the **P_D Phase** and **P_D Rate** columns signify error-free conditions.

As demonstrated by the results for test pairs 2, 3, 4, 5, and 6, the behavior of WWP is very constant with respect to time (and hence throughput) and energy expenditure. The throughput achieved is far better than TCP's and the energy expenditure is far less. WWP adapts quickly to error phases. It does not automatically decrease its sender window size (*i.e.*, wave level) in response to a drop as does TCP. Instead, immediately upon experiencing a drop, it pauses in its data transmission and probes. It then checks the measured RTTs from the probe cycle to assess whether conditions allow continued transmission and, if so, at what level. It can adjust back immediately to a high wave level where appropriate, unlike TCP which applies graduated multiplicative/additive increases to its window size.

This mechanism of WWP's has two significant results: (i) data transmission is not wasted during sustained periods of degraded network capacity, and thus retransmissions are reduced to a minimum; and (ii) throughput is maximized since we can adjust to high wave levels immediately under appropriate conditions, thereby not wasting opportunities for successful data transmission by attempting less than the network would easily be able to accommodate. This is clearly demonstrated by the results for test pairs 5 and 6. There, TCP reduces its window size significantly since the error rate is high. It then takes considerable amounts of time to readjust the size back up to an appropriate level, thereby "missing" the "good" phase. This results in unnecessarily "trading off" prolongation of the connection time in order to avoid retransmissions. The protocol is clearly inefficient under such conditions.

These observations gain further strength from Table 2 and Figure 3 below. There, we present results for tests that were run under the same experiment environment as for Table 1. Only the On/Off phase duration of PHASE_DROP differs: 4 seconds instead of 10. Consequently, TCP's window size is further reduced and, not only is the connection time increased (columns **Time Ran** and **Time Overhead**), but so are the total bytes transmitted (columns **Total Bytes** and **Tr. En. Wastage**), and hence the energy expended. This can be easily explained. There are now more phases during which TCP "collapses" in response to errors, and its attempts to readjust its window size causes the following dynamic to occur. Segment drops that take place during the early part of the error-prone phase occur under conditions where the window size has been growing in response to the preceding error-free phase, and these phase transitions are now more frequent. Furthermore, connection time is increased since the *average* window size during the connection is now much smaller than before. WWP, in contrast, exhibits exemplary behavior: it stops and restarts data transmissions immediately upon entering and leaving the error phases, respectively; it is constantly readjusting its wave levels, commensurate with the error environment it is detecting. Test pair 4 clearly demonstrates the vast difference between the two protocols' effectiveness under heavy error conditions.

Table 3 below summarizes the effectiveness of the protocols with respect to energy savings and throughput, as well as their relative behavior under varying conditions, and is based on the results presented in Tables 1 and 2. Column **Energy Saving** gives the number of additional bytes transmitted by TCP to complete the 5-MByte

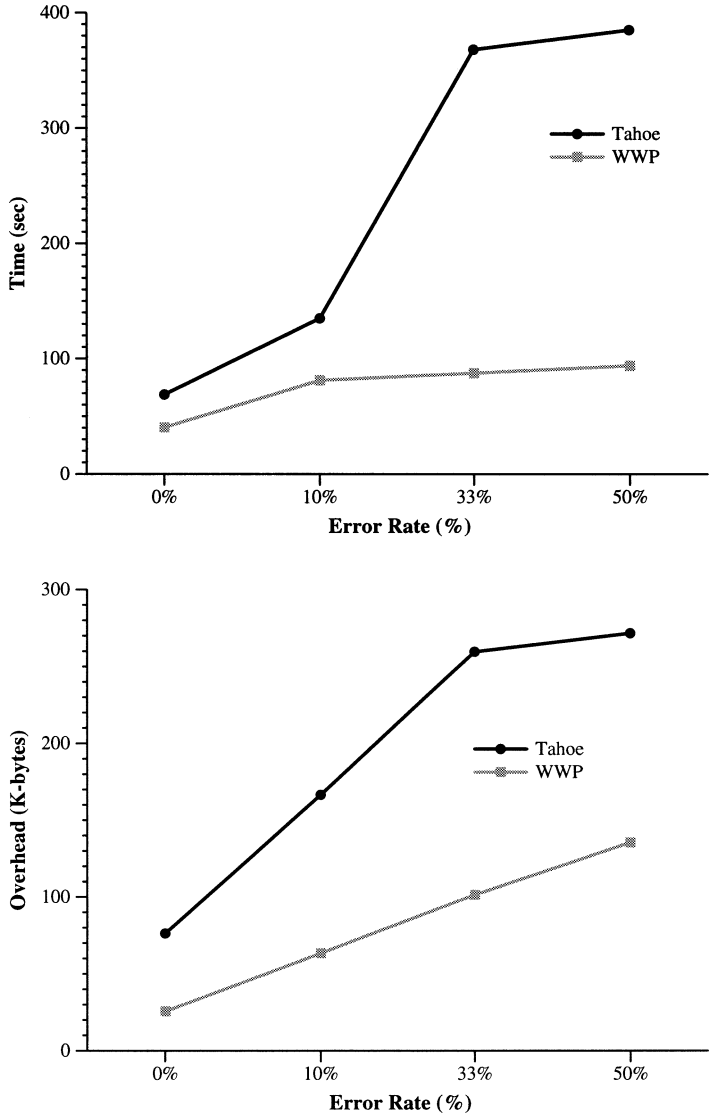

Figure 3. Connection time and overhead with 4-second On/Off error phases.

data delivery, over and above the total number of bytes transmitted by WWP under the same test conditions. Similarly, **Time Saving** gives the additional time taken by TCP compared to WWP. **Energy Expend. Ratio** is a measure of the relative energy expenditures of the two protocols, and is calculated as

$$\text{TCP (Total Bytes Transmitted} - 5\text{ MBytes)/WWP (Total Bytes Transmitted} - 5\text{ MBytes)}.$$

Throughput Ratio gives the relative throughput rates:

TCP Effective Throughput/WWP Effective Throughput.

Finally, **Time Ratio** shows the relative connection times for the two protocols (TCP/WWP).

When error rates are low, TCP behaves well and its major weakness—inability to adequately readjust to rapidly changing conditions—does not catastrophically degrade its performance. WWP displays consistently good behavior, even when variability in the error environment is quite significant. Under such conditions of high variability, TCP wastes enormous amounts of energy and unexploited throughput capacity, and does not achieve good performance. Its throughput in all error-prone cases is well below what it is capable of achieving under error-free conditions, let alone the higher throughput displayed by WWP.

As demonstrated by test pairs 3 and 4, TCP is unable to take advantage of network conditions, expending up to fourfold more time than WWP. It is interesting to note (see 3rd test pair) that it transmits up to 158 KBytes more than does WWP under the same conditions, in order to deliver its 5 MByte data set. So it is not even trading off lesser transmission effort at the expense of longer connection times in an effective manner. Indeed, it is especially the longer connection times of TCP that seriously aggravate the energy consumption characteristics of the protocol, rather than wasted transmission effort as such. Throughput can reach as low as only 23% of that achieved by WWP, although under error-free conditions it can achieve up to 58.6% of WWP's corresponding throughput. Similarly, under error-free conditions, their relative time ratio was 1.7/1; this ultimately grows to about 4/1 as error conditions deteriorate.

Results presented in the table for the relative energy expenditure deteriorate to a value of 4.48/1, though we would argue that it really should reach even worse values were the total connection times to be taken fully into account. For example, if we were to estimate energy consumption per unit of idle time (*i.e.*, when no transmission is taking place) at 10% of the energy consumed during a unit of transmission time (though this is in fact, device dependent), the energy consumed by WWP in order to deliver 5 MBytes of data could easily be calculated to be up to 30%–50% less than that consumed by TCP.

4.1.2. Behavior under rapidly-changing conditions. The following experiments use shorter error-duration periods under similar conditions of error rates. The data size is 2 Mbytes. The more persistent the error, the better the relative performance for TCP, comparing it in its own right. WWP, however, clearly achieves higher throughput and conserves more energy than TCP. Furthermore, while both protocols' throughputs and energy-expenditures degrade as increasingly problematic network conditions are simulated, WWP's performance *relative* to TCP's improves, using their performance under error-free conditions as the reference point (compare columns **TCP Bytes** and **WWP Bytes**; and, especially, the entries in column **Time Ratio**). It is notable that WWP performance remains relatively stable under varying error-phase duration (1 sec, 2 sce, 5 sec) while TCP's inability to adjust its

transmission policy to current network conditions is more marked as phase changes become more frequent/rapid.

An interesting observation can be made about the 50% and 100% error-rate tests. Note that TCP overall performance (energy and time) *improves* when the error rate increases from 50% to 100% . Although not reported here, similar behavior occurs with On/Off periods of 3, 4, and 5 seconds. We conjecture the following explanation. At a 50% error rate, attempting transmissions that are likely to be lost causes upward adjustment of the sender's timeout as well as reduction in the size of the congestion window. For each acknowledgment that comes through, depending on the TCP version used, additional adjustments will be applied. Waiting for an extended timeout to expire results in wasting possibly error-free periods without attempting any transmission. In contrast to this scenario, with a 100% error rate the sender backs off virtually completely. No acknowledgment comes through and no further timeout adjustments are triggered off, resulting in a more rapid response by the sender to the changeover to an error-free phase, and hence in better performance. Note that this seemingly anomalous behavior is very environment-specific. In another testing environment, 2-, 3-, 4-, and 5-second periods might trigger different patterns. However, we would expect to see this anomalous behavior across some range of error-phase duration. A similar effect is observed and explained in the next subsection with respect to the aggressive version of WWP.

4.2. The energy/throughput tradeoff

Table 5 summarizes results from experiments undertaken to explore the nature of the energy/throughput tradeoff. WWP is configured to behave aggressively in one group of tests, backing off and probing when errors occur, but immediately resuming maximum transmission effort at the highest wave level when a clear period is detected. It behaves more conservatively in the second group of tests, during which its transmission varies between three different wave levels depending on the RTT measurements and the detected frequency/duration of the errors. Note that the first row of the table gives the base (error-free conditions) measurement for the two versions. The remaining rows report the overhead *relative* to this measurement.

As anticipated, the more aggressive the transmission effort, the more the effective throughput so long as the error rate is low (*e.g.* 5%). The gain in throughput is higher than in the conservative approach, while the extra energy expenditure is minor. As conditions deteriorate and reach error levels of 20%, the energy cost becomes significant but the gain in throughput is still important. At this stage, it is an application's choice whether energy or throughput would be the significant factor. This dilemma, however, resolves itself when error rates reach levels as high as 33%. At these rates aggressive behavior not only does not save energy, but instead degrades the protocol's overall performance.

The last row of the table represents a heavy burst error condition where the error rate for the On phases is 100%. Here, the aggressive version, using shorter time-outs and adjusting faster to maximum transmission effort, results in enhanced throughput. Note that when the error rate increases to 100%, performance of the

aggressive version of the protocol actually improves. In the 50% case the protocol continues to attempt transmissions during the On periods at some relatively aggressive level. The error rate is sufficiently dense, however, that the gain achieved by successfully-attempted segments is more than counterbalanced by the failures that require retransmission. Under the 100% case, on the other hand, the protocol effectively backs off completely during the On case and does not attempt any transmissions at all.

5. Conclusion

We have presented the "wave" and "probing" mechanisms, which are suitable for reliable transport protocols that aspire to high throughput and low energy expenditure in network environments with significantly variable error characteristics. In contrast to TCP congestion control, wave and probing suffices as a universal, end-to-end error-control mechanism: it effectively detects the network condition by measuring the receiving throughput instead of observing packet loss at the sender side. It has the ability to distinguish different types of errors and apply an appropriate strategy for error recovery. The protocols that use waves and probing do not achieve better performance because of their aggressive behavior; they can be aggressive or conservative based on the nature of the detected error. It is exactly this capability that permits such protocols to utilize better the available bandwidth within the confines of fairness to other flows that potentially share the bandwidth of the communication channel.

WWP itself owns a number of advantages over standard TCP for mobile computing. It expends less energy since retransmission overhead and duplications can be avoided. Beyond its energy saving feature, WWP also maintains high throughput. Finally, since the receiver, instead of the sender, decides about the size of the transmission window, problems that arise from asymmetric paths can be efficiently resolved. WWP, however, is not compatible with TCP semantics. It might be desirable in several cases to incorporate the mechanisms presented here into TCP and permit a wide range of existing applications to take advantage of potential improvements of energy and throughput performance. Our current work on TCP-Probing and TCP-Wave is towards that direction.

References

1. M. Allman, V. Paxson, and W. Stevens. TCP congestion control. RFC 2581, April 1999.
2. 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.
3. A. Chocalingam, M. Zorzi, and R. R. Rao. Performance of TCP on wireless fading links with memory. In *Proceedings of IEEE ICC'98*, June 1998.
4. S. Floyd and T. Henderson. The new Reno modification to TCP's fast recovery algorithm. RFC 2582, April 1999.
5. V. Jacobson. Congestion avoidance and control. In *Proceedings 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 and 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. V. Paxson, et al. Known TCP implementation problems. RFC 2525, March 1999.
9. J. Postel. Transmission control protocol. RFC 793, September 1981.
10. K. Ramakrishnan and S. Floyd. "A proposal to add explicit congestion notification (ECN) to IP", RFC 2481, January 1999.
11. V. Tsaoussidis, H. Badr, and R. Verma. Wave and wait: an energy-saving transport protocol for mobile IP-devices. In *Proceedings of IEEE ICNP '99*, Toronto, Ont., Canada, October 1999.
12. V. Tsaoussidis, A. Lahanas, and H. Badr. The Wave and wait protocol: high throughput and low energy expenditure for mobile-IP devices. In *Proceeidngs of IEEE ICON*, 2000.
13. 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.
14. The X-kernel. *www.cs.arizona.edu/xkernel*.
15. M. Zorzi and R. Rao. Energy efficiency of TCP. In *Proceedings of MoMUC '99*, Calif., San Diego, 1999.