Open Issues on TCP for Mobile Computing*

VASSILIS TSAOUSSIDIS
Computer Science
Northeastern University
Boston, MA 02115, USA
vassilis@ccs.neu.edu

IBRAHIM MATTA
Computer Science
Boston University
Boston, MA 02215, USA
matta@cs.bu.edu

Abstract

We discuss the design principles of TCP within the context of heterogeneous wired/wireless networks and mobile networking. We identify three shortcomings in TCP's behavior: (i) the protocol's error detection mechanism, which does not distinguish different types of errors and thus does not suffice for heterogeneous wired/wireless environments, (ii) the error recovery, which is not responsive to the distinctive characteristics of wireless networks such as transient or burst errors due to handoffs and fading channels, and (iii) the protocol strategy, which does not control the tradeoff between performance measures such as goodput and energy consumption, and often entails a wasteful effort of retransmission and energy expenditure. We discuss a solution-framework based on selected research proposals and the associated evaluation criteria for the suggested modifications. We highlight an important angle that did not attract the required attention so far: the need for new performance metrics, appropriate for evaluating the impact of protocol strategies on battery-powered devices.

Keywords: TCP, congestion control, wireless links, mobile computing, energy efficiency.

1 Introduction

Mobile Networking is facing a growing need for efficient communication mechanisms that functionally integrate wireless and wired Internet components across their varying and distinctive transmission characteristics. Traditionally, the design of the core protocols of the wired Internet did not account for wireless architectures. For example, the end-to-end services of the wired Internet were built based largely on the functionality of the Transmission Control Protocol (TCP). TCP was designed and carefully calibrated to overcome the problems of stability, heterogeneity (receiver buffers, network bandwidth and delay), fairness in bandwidth consumption of competing flows, efficiency in utilization, and congestion control that effectively avoids situations of congestive collapse. The wireless Internet raised a number of issues that call for new protocols and architectures and for re-evaluation of well-known standards. Key issues of the wireless Internet that call for attention are the error characteristics of wireless links and the specific operations of mobile computing as well as the performance metrics to evaluate efficiency.

Standard TCP was evaluated against several proposals recently in the context of wireless networks. It became clear that the protocol needs improvements mainly because of its insufficient error control. That is, TCP error control is centered on congestion losses and ignores the possibility of transient random errors or temporary "blackouts" due to hand-offs and extended burst errors that are typical in wireless networks. Although the reader might consider that bursty drops, for example, are typical also in wired networks due to congestion, the distinctive treatment of graduated window adjustments in order to avoid a congestive collapse in situations of limited bandwidth is not necessarily the optimal strategy to recover from link errors.

More precisely, it was realized that an efficient TCP for wired/wireless networks needs to be equiped with additional mechanisms that deal with:

*This work was supported in part by NSF grant CAREER ANI-0096045.

- The detection of the nature (frequency, duration, etc.) of the errors that caused packets to be dropped. This knowledge can be used to determine the appropriate error-recovery strategy. For example, hand-offs, fading channels, congestion and transient random errors call for distinct recovery strategies.
- The adjustments of the sender's level of data transmission, within the confines of fair behavior and depending on the nature of the errors detected. This enables recovery strategies that can adjust to the underlying networks' error characteristics, device constraints and performance tradeoffs.
- The protocol's energy- and time-saving capabilities. This allows for efficient applications of the protocol with mobile, battery-powered devices.
- The reliable and precise detection of congestion in both the forward and the reverse path (hence, not based on RTT measurements), and possibly not by means of experiencing packet drops as in standard TCP. Such additional mechanisms could bypass the known problems of asymmetric links and reduce the impact of the timeout, which is based on measurements of the Round Trip Time (RTT), on protocol performance.

Notably, some of the protocol's inefficiencies might not become apparent when the protocol is evaluated with traditional performance metrics. For example, goodput has traditionally been used to measure the rate at which data is delivered to higher layers. On the other hand, the transmission effort expended by the protocol in order to achieve its goodput is indeed a significant metric. The appropriate evaluation for today's Internet, mainly due to the increasing presence of battery-powered devices, requires additional metrics such as energy efficiency, overhead or transmission effort. Traditional results and methods that today constitute the beacon of both fairness and efficiency are not necessarily optimal for wired/wireless networks and the Internet.

In conjunction with the above considerations a design issue has also been raised: "Where is the right place to add the integrative functionality of the wired and wireless Internet?". We outline some of the key mechanisms of TCP that require modifications in order to enhance the protocol performance in wired/wireless networks. We discuss recent results from the perspective of the protocol functionality and specific mechanisms and we classify recent proposals accordingly. We organize the rest of the paper as follows: In Section 2 we review the operations of standard TCP congestion control. In Section 3 we discuss the problems of TCP in the context of wired and wireless networks. In Section 4 we outline the requirements of error detection and we present the design of a "probing" device that satisfies these requirements. We discuss techniques for error recovery and related proposals for improving TCP performance from the perspective of heterogeneous wired/wireless networks in Section 5. In Section 6 we agitate TCP's strategy from the perspective of adaptive error control and the associated energy/throughput tradeoffs. In Section 7 we present concluding remarks.

2 Transmission Control Protocol (TCP)

Today, the majority of application protocols use the Internet's reliable Transmission Control Protocol (TCP). The functionality of TCP [31] is designed to be adequate not only for Internet applications but also for the variety of underlying networks. The protocol aims at providing a reliable service with the following features:

- Fairness to other flows that potentially share a channel's bandwidth
- Dynamic discovery of current availability of bandwidth
- Mechanisms for congestion avoidance and control and for optimization of the error recovery process

Error control mechanisms are the central component of reliable protocols. They affect a protocol's performance with respect to goodput, energy expenditure, and overhead. Error control is usually a two-step process: error detection, followed by error recovery. TCP detects errors by monitoring the sequence of data segments acknowledged (received). When timeouts are correctly configured, a missing segment is taken to indicate an error, namely that the segment is lost due to congestion (i.e. buffer overflow). Reliable protocols usually implement an error recovery strategy based on two techniques: retransmission of lost segments; and downward adjustment of the sender's window size and readjustment of the timeout period.

In the standard TCP versions the receiver can accept segments out of sequence, but delivers them in order to higher protocols. 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 [1], which includes a sequence number identifying the next in-sequence byte expected. The transmitter implements a congestion window that defines the

[1] This is not exactly true with Delayed Acknowledgements. Delayed Acknowledgements are discussed in Section 5.2

maximum number of transmitted-but-unacknowledged bytes permitted. This adaptive window can increase and decrease, but the actual "sending window" never exceeds the minimum of the receiver advertised and congestion window. Standard TCP applies graduated multiplicative and additive adjustments to the sender's congestion window in order to guarantee stability and fairness to other flows that possibly attempt to simultaneously utilize the bandwidth of the channel. The Additive Increase Multiplicative Decrease (AIMD) algorithm is used to implement TCP window adjustments; based on the analysis of Chiu and Jain the algorithm achieves stability and converges to fairness in situations where the demand (of competing flows) exceeds the channel's bandwidth [15].

2.1 TCP Tahoe, Reno, Vegas

The error control mechanism of TCP is primarily oriented toward congestion control. Congestion control can be beneficial to the flow that experiences congestion, since avoiding unnecessary retransmission can lead to better goodput [22]. The basic idea is for each source to determine how much bandwidth 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 to congestion, the TCP sender reduces the transmission rate by shrinking its window.

Tahoe and Reno are the two most common reference implementations of TCP. They rely on inducing lost segments to detect congestion so that a flow can adjust its rate to its available share. Hence, their approach implements congestion control. Rather, TCP Vegas [12] implements congestion avoidance as it attempts to detect congestion before it happens (i.e. before segments are lost). We discuss each version in turn.

TCP Tahoe: TCP Tahoe congestion control algorithm includes Slow Start, Congestion Avoidance, and Fast Retransmit [3]. It also implements an RTT-based estimation of the retransmission timeout. In the Fast Retransmit mechanism, a number of successive (the threshold is usually set at three), duplicate acknowledgments (dacks) carrying the same sequence number triggers a retransmission without waiting for the associated timeout event to occur. The window adjustment strategy for this "early timeout" is the same as for the 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.

TCP Reno: TCP Reno introduces Fast Recovery in conjunction with Fast Retransmit. The idea behind Fast Recovery is that a dack is an indication of available channel bandwidth since a segment has been successfully delivered. This, in turn, implies that the congestion window (cwnd) should actually be incremented upon a dack delivery. In particular, receiving the threshold number of dacks triggers Fast Recovery: the sender retransmits one segment, and sets the congestion threshold to half the current cwnd. Then, instead of entering Slow Start as in Tahoe, the sender increases its current 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" the window. The Fast Recovery stage is completed when an acknowledgment (ack) for new data is received. The sender then sets cwnd to the current congestion threshold value ("deflating" the window), and resets the dack counter. In Fast Recovery, cwnd is thus effectively set to half its previous value in the presence of dacks, and grows with Additive Increase rather than Slow Start.

TCP Reno's Fast Recovery can be effective when there is only one segment drop from a window of data, given the fact that Reno retransmits at most one dropped segment per RTT. The problem with the mechanism is that it is not optimized for multiple packet drops from a single window, and this could negatively impact performance.

TCP Vegas: TCP Vegas approaches the problem of congestion from another perspective. Based on sample RTT measurements, and the size of the sending window, the sender calculates the throughput rate every RTT. This rate is compared to an expected rate, which is calculated based on what is up-to-now known as best RTT (called base RTT). Two thresholds α and β trigger an additive increase or decrease, depending of whether the channel is under- or over-utilized, respectively. Although not obvious, the protocol is compliant to the rules of fairness: it applies multiplicative adjustments when congestion is detected. Indeed, it applies Slow Start in the event of a time-out. Vegas's approach is interesting: it attempts to estimate the level of congestion before it happens, and consequently avoid it, thereby avoiding unnecessary packet drops. One of the problems that it does not seem to overcome is the path asymmetry. The sender makes decisions based on the RTT measurements, which, however, might not accurately indicate the congestion level of the forward path. Furthermore, packet drops caused by transmission deficiencies or fading channels may trigger a Slow Start. However, this problem is common to all the above versions and is detailed below.

3 Research Issues

TCP displays some undesirable patterns of behavior in the context of wired/wireless networks, and lacks strategies for efficient energy expenditure that aspire to high goodput. A prime missing module from TCP that (its absence) negatively impacts its operation in heterogeneous environments with both wired and wireless components is error detection. TCP is not capable of detecting the nature of the error but only the result of the error; namely, that a packet is dropped. Hence, the error recovery mechanism is not always efficient, especially when the error pattern changes, since packet loss is invariably interpreted by the protocol as resulting from congestion. For example, when relatively infrequent random or short burst errors occur, the sender backs off and then applies a conservatively graduated increase to its reduced window size. During this phase of slow window expansion, opportunities for error-free transmissions are wasted and communication time is extended. In other words, in the presence of infrequent and transient errors, TCP's back-off strategy avoids only minor retransmission at the cost of significantly degraded goodput, which increases overall connection time. Yet, when an error occurs and TCP does back off, it continues to forcefully attempt transmissions within the confines of the reduced window size. In the presence of errors of a relatively persistent nature (fading channel, prolonged and frequent burst errors, congestion), this behavior does not favor energy saving, since it might yield only minor goodput improvement at high cost in transmission energy. In summary, from the perspective of energy expenditure in the context of heterogeneous wired/wireless networks, TCP seems to possess an inherent tendency to back off too much when it should not, and too little when it should [38].

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 [16, 37, 8, 24]. In addition, the protocol lacks the ability to efficiently monitor network conditions, rapidly readjust its window size in response to changes in these conditions[2], and detect congestion without inducing packet drops, thereby degrading overall performance through additional retransmission and wasted opportunities in maintaining the communication pipe full. The traditional schema of congestion control which uniformly uses backwards adjustment of the congestion window in the event of retransmission, and which is exemplified by the TCP paradigm, does not necessarily suffice for wired/wireless networks. Note that a complementary action to the congestion window adjustment is the timeout adjustment. Extension of this timeout causes a degraded ability to rapidly detect error-free conditions and recover immediately. In this context, an immediate full-window recovery after a transient link error does not violate the AIMD [15] principle, which applies only when congestion exists. That is, AIMD does not favor fairness, efficiency or stability when losses are due to errors other than the insufficient bandwidth for the current demand of competing flows. Along the same lines, hand-offs or burst errors due to fading do not necessarily justify an extension of the retransmission timeout or a graduated window adjustment (as opposed to full window recovery) upon successful retransmission.

Since TCP is not optimized for a specific network type or application requirement, its mechanisms permit for several application- and network-specific improvements. TCP's behavior over wired networks, where congestion is a regular cause for packet loss, was initially studied by Jacobson [22]. Recently, TCP behavior over wireless/wired and satellite networks has become a focus of attention. Recent research results [2, 23, 38, 46, 16, 8, 43, 30, 25] have shown that TCP throughput (i.e. sending rate) degrades, in the presence of the kind of random/burst errors and long propagation delays typical of wireless and satellite environments, respectively. As a result, some researchers have tended to focus on the development of architectures (e.g., wireless proxies) that assist the protocol's operation over specific networks in order to introduce minor changes to the protocol itself. Therefore, a large number of suggested proposals aiming at improving TCP performance and avoid or control network congestion deal with the functionality of network devices that can assist the protocol operations.

For example, the enhancements discussed in [8, 21, 32] require intervention at the router or base-station level, so as to either provide the sender with explicit information about the nature of the error or attempt to hide altogether the error from the sender. For example, Ramakrishnan and Floyd in [32] propose an Explicit Congestion Notification (ECN) to be added to the IP protocol in order to trigger TCP congestion control. A duality is served with ECN: TCP performance can be enhanced by means of avoiding losses of data windows due to limited buffer space at the bottleneck router, and congestive collapse can be avoided. Clearly, it could be beneficial for the sender to know with precision that congestion is about to happen. However, in the context of wired/wireless networks ECN contribution might be limited: by not receiving an explicit notification the TCP sender will not be able to safely assume that a detected drop was not caused due to congestion. The desired precision of ECN-capable TCP senders would be better approached if the level of congestion could also be indicated in order to allow TCP to implement more sophisticated recovery. Precision however comes at a cost: the functionality of the routers is more complex, modifications are required to both routers and TCP itself, and finally, the return might be small due to heterogeneity, i.e. some routers are not ECN-capable.

Unlike ECN, RED Gateways [19] drop, rather than mark, packets when congestion is about to happen. The goal is to trigger TCP congestion control, aiming at limiting the loss to one packet. RED can function without requiring any change to the current transport level infrastructure. Recent work [28] presents a critical discussion on the performance expectations

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

with RED. An interesting observation about RED and ECN is that they could, somehow, confine future evolution. Imagine a more sophisticated TCP which distinguishes between congestion and wireless losses. Since RED drops packets in proportion to sending rates, it is unclear how fair RED would be to the sophisticated TCP which just happens not to unnecessarily back off in case of transient wireless losses.

The TCP-related work discussed above raises an important question: *"where is the right place to add the required functionality?"*. This question does not have a clear answer; error control is not exclusively a management property of the router or the base station, nor is it exclusively assigned to the transport layer. A widely-accepted approach, presented by the end-to-end argument [35], states that we can only implement a function at a lower layer, if that layer can perform the complete function. Since the lower level cannot have enough information about the applications' requirements, protocol parameters, and device constraints, it cannot implement the whole function of error control; it can only be used to optimize the function of the higher layer.

There are three salient points to make: First, TCP performs error control that is specifically tailored toward congestion-induced errors; hence it does not have the entire error control functionality required for heterogeneous networks in place. Second, the proposed optimizations frequently exhibit a conflicting behavior with TCP's mechanisms; that is, a retransmission attempt at a lower level might result in extending the RTT estimates of TCP and hence its timeout. This extended timeout might adversely affect the ability of TCP to effectively detect error-free conditions. Furthermore, we cannot assume that all network devices or link layer related modules would have the required functionality to support a defective TCP. Third, low-level modifications might constrain future TCP evolution. That is, the transport layer seems to be the natural place for implementing the core functionality for error control. This does not exclude potential optimizations based on the more precise or accurate information that could be gathered from lower layers. For example, having a TCP-aware gateway at the border of the wired and wireless networks enables a more precise determination (i.e., localization) of the source of the error. By and large, it appears that there is no much supportive theory behind the above design issue; cost, applicability and impact appear to be the major criteria for evaluating different approaches.

In summary, key problems of TCP that require attention are: **Error Detection**, that determines the locus of the drop and possibly the nature of the error, **Error Recovery** that is responsive to the error pattern and, **Protocol Strategy** that favors one service characteristic over another (e.g., energy over goodput).

4 Error Detection

TCP detects losses by means of timeouts and duplicate acknowledgments. Congestion control is trigerred upon a packet loss. TCP does not distinguish losses due to congestion or link errors resulting in degraded performance with wireless networks. This problem does not become apparent when TCP is split at the border of the wired and wireless networks since the intermediate device (i.e., base station) can distinguish the two networks. However, the above consideration comes with an oversimplified assumption: that the key issue is to distinguish in which portion of the network (i.e., the wired or the wireless) the drop has occurred. This information is expected to be used simply to determine whether the window should be adjusted downwards and the timeout should be extended (wired), or otherwise a more aggressive retransmission strategy should be implemented.

In fact, determining that the error has occurred in the wireless portion does not really provide sufficient information for the sender's recovery tactic. For example, further adjustments in response to a transient error could be practically unnecessary; suspending data transmission during burst errors could result in energy savings; and a conservative strategy during highly-erroneous transmission could possibly conserve significant amounts of energy for a minor degradation in goodput. Therefore, the "nature" of the error even within the wireless network seems to call for distinctive recovery tactics by the TCP sender. For example: to adjust the timeout and apply graduated adjustments after retransmission (e.g., during congestion); to suspend data transmission for periods of "blackout" and retransmit aggressively when conditions clear (e.g., during hand-offs); to reduce transmission effort for some period of time (e.g., during fading channels); to do nothing (e.g., after a transient random loss). In summary, error detection needs to go beyond localization and practically detect (classify) the "nature" of the error in order to provide feedback to the error recovery strategy. Notably, most of recent proposals do not tackle the problem of fine-grained error classification.

In order to enhance TCP goodput and energy efficiency, Tsaoussidis and Badr [37] proposed **TCP-Probing**, grafting a probing mechanism into standard TCP. In this scheme, a "Probe Cycle" consists of a structured exchange of "probe" segments between the sender and receiver that monitor network conditions. The sender enters a probe cycle when a segment is detected lost either by a time-out event, or by three dacks. When a data segment is detected lost, 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 (header without payload) are sent. A lost probe or acknowledgment re-initiates the cycle, hence suspending data transmission for the duration of the error. When the probe cycle is completed, the sender compares the measured probe RTTs and determines the level of congestion. Had congestion been the possible cause of the drop, the

sender would have applied backward adjustments as in Tahoe or Reno. The enhanced error detection mechanism allows for "Immediate Recovery" (full-window recovery) when the error is detected to be transient.

Hence, a "probing device" models two properties: (i) it inspects the network load whenever an error is detected and rules on the cause of that error and, (ii) it suspends data transmission for as long as the error persists, thereby forcing the sender to adapt its data transmission rate to the actual conditions of the channel.

A base station at the border of a wired and wireless network can also detect the nature of the error, and inform the sender or take actions appropriate to the error type. In **WTCP** [34, 33], the base station buffers data segments received from the sender. WTCP in the base station can then locally recover segments lost on the wireless link. WTCP independently performs flow control for the wireless connection (between the base station and mobile host). WTCP transmits from its buffer all the segments that fall within the wireless link transmission window.

Each time a segment is sent to the mobile host (including a retransmission), the timestamp of the segment is incremented by the amount of time that segment spent in the WTCP buffer (residence time). The reason for doing this is discussed shortly. The base station acknowledges a segment to the fixed host *only after* the mobile host actually receives and acknowledges that segment. Hence, TCP end-to-end semantics is maintained throughout the lifetime of the connection. Also, the round trip time seen by the source is the actual round trip time taken by a segment to reach and return from the mobile host, *i.e.* it does not include residence time at the base station needed for local recovery. Thus WTCP attempts to exclude the effect of the wireless link on the round trip time estimate and hence timeout maintained by the sender. Not doing so may unnecessarily increase the timeout value, which in turn may hinder the ability of the TCP sender to quickly detect congestion over the wired portion of the network.

Based on duplicate acknowledgment or timeout, the base station locally retransmits lost segments. In case of timeout, WTCP assumes a typical burst loss on the wireless link is going to follow. Unlike TCP Reno, WTCP interprets the reception of a duplicate acknowledgment as an indication that the wireless link is in good state. Section 5 discusses how the error recovery of WTCP exploits this wireless error classification.

4.1 Discussion

Probing mechanisms at the transport level were first presented in [39] in the context of an experimental protocol. Although the desired properties (i.e., self-adjusting capability, throughput/delay monitoring) of probing devices have been clearly identified in [39, 37], the optimal design of such devices for TCP and in the context of wired/wireless networks remains an open issue. For example, a probe cycle could consist of one extensible probe/ack exchange in the form of a packet-pair, where the level of congestion could be measured in only one RTT. This is expected to have a positive impact on throughput under infrequent errors of a transient nature. However, it may have a negative impact on the protocol's self-adjusting capability under scenarios of errors with certain density since the possibility for a probe or a probe acknowledgment to be lost would be confined. Looking at the same idea from another perspective, we could design a cycle to consist of more than one structured exchanges of probes and their acks, hence enhancing further the capability of detecting error patterns of moderated density at the cost of additional RTTs. Such a design would enforce a more conservative recovery strategy. The efficiency of different probing techniques for TCP has not been investigated.

Along the same lines, the efficiency of detecting the level of congestion calls for further attention. In [39, 44] Tsaoussidis *et al.* propose the use of the wave pattern for detecting errors at the receiver. A wave is a fixed [3] pattern of data exchange between sender and receiver that enables the receiver to measure the perceived level of congestion based on the time required for a wave to be delivered. More precisely, in TCP-Real [44], the receiver observes the network dynamics (congestion level) based on the ratio of the wave size to the wave delivery time. The wave delivery time is the time difference between the reception of the first and the last segment of the wave and it could be much smaller than the RTT.

The efficiency of such measurements on the congestion level has two dimensions: how to measure congestion, and where to measure it (i.e., at the sender or the receiver). In the next section we extend the discussion on the latter, since it crucial for the error recovery tactics of TCP. As for the former, whether an estimation of congestion should be based on RTT measurements, on the packet-pair approach, on the wave pattern, or on single probes is an open issue of further research and experimentation.

Finally, although wireless proxies like WTCP may be effective when the data and their acks follow the same path through the proxy, they become useless in situations where, for example, data segments follow a satellite link through the proxy and acks follow a separate terrestrial path. Furthermore, some wireless proxy schemes assume specific versions of TCP to be effective. For example, WTCP assume TCP versions which implement timestamps for their RTT estimation. Snoop [9], on the other hand, improves the performance of TCP Reno by exploiting its dupack mechanism to identify the loss of buffered segments over wireless links.

[3] Yet flexible: flexibility is achieved by using different wave-levels.

5 Error Recovery

Once the protocol is able to distinguish the nature of the error the recovery strategy might need to be more aggressive when the errors are not due to congestion. Error recovery in TCP is not managed by a single component. That is, the congestion window adjustment, the acknowledgement strategy, the timeout mechanism as well as other factors (e.g., receiver advertized window, slow start threshold) all contribute to the efficiency of the recovery process. Along these lines, research efforts were made to optimize specifically some of these components under predetermined conditions and to enhance functionality at different layers. We next discuss some of these enhancements.

5.1 The TCP sending window and timeout adjustments

TCP itself needs improvements that deal with various types of error, not only lost segments due to buffer overflow. Because TCP assumes only congestion-induced losses, Fast Recovery and Fast Retransmit are *not* fast enough when the detected losses are not because of network congestion. For example, the sender needs to respond to the error detected using the congestion window and the timeout mechanism appropriately. The regular course of action of most approaches is to shrink the congestion window and extend the timeout period due to congestion, or freeze the window and timeout upon drops due to non-congestion errors. Along these lines, Goff *et al.* [20] discuss a modification to avoid degrading performance due to handoffs. **Freeze-TCP** avoids timeouts at the sender during handoffs since a timeout shrinks the sending window to a minimum in all TCP versions. To this end, Freeze-TCP exploits the ability of the receiver to advertise a window of zero. Based on the signal's strength the receiver detects the handoff and advertizes a Zero Window Adjustment (ZWA). The sender then freezes its transmission and its timeout value. Once the handoff is completed, the receiver advertizes a non-zero window so that the sender resumes with its window and timeout values unaffected due to the handoff.

In **TCP-Probing** [37] Tsaoussidis and Badr introduce a recovery strategy that is responsive to the nature of the error detected. Detection is based on the grafted *probing device* outlined in Section 4 and is coupled with an additional tactic, called *Immediate Recovery*. That is, neither the congestion window nor the slow start threshold are adjusted downwards. Timeout values during probing are also not adjusted. The protocol allows for three distinct tactics in response to the nature of the error detected: Slow Start (for congestion detected by timeout), Fast Recovery (for moderated congestion detected by three dacks), and Immediate Recovery (for congestion-free path). Note that a fourth tactic (i.e., conservative recovery due to frequent link errors) is not included in the recovery strategy. The reasoning behind this decision is the supportive property of the *probing cycle* to be extended when a probe or an acknowledgment is missing. This property complements the recovery strategy: when the error is dense, probing will be naturally extended and data transmission will be suspended, resulting in a more conservative transmission strategy. Hence, a major contribution of the "Probing Device" is its ability to naturally adapt to error conditions and extend the "probing cycle". The result of this operation is gracious during handoffs or persistent burst errors: the sender suspends transmission and the timeout is not extended, thereby maintaining the advantage of detecting the "good" phases over other versions. Suspending data transmission during such deteriorated conditions results in energy conservation, which makes the protocol particularly applicable to mobile, battery-powered devices.

Immediate Recovery is in marked distinction to Reno behavior at the end of Fast Retransmit. The logic here is that having sat out the error condition during the probe cycle and finding that network throughput is improved at the end of the cycle, an aggressive transmission is more clearly indicated. Although two RTTs are wasted even by a single packet loss, Probing can be more effective than Reno and Tahoe when the sending window is not too small. The Immediate Recovery mechanism of TCP-Probing avoids the Slow Start and/or the congestion avoidance phase of Tahoe and Reno. In this case Probing immediately adjusts the congestion window to the recorded value prior to the initiation of the probe cycle. When congestion is indicated, the protocol complies with the congestion control principles of standard TCP.

5.1.1 Decoupling the Sending Window from RTT

TCP window increases/decreases, in response to the receipt of acknowledgments and their delay or loss due to insufficient bandwidth availability at the reverse path, cause a conservative behavior at the forward path. Similarly, timeout extensions due to lost acknowledgments might degrade further the protocol's ability to rapidly detect error-free conditions and consequently degrade its capability to detect the windows of opportunity to exploit the available bandwidth. A study of TCP performance over asymmetric links is presented in [7].

In **TCP Santa Cruz** [29], Parsa and Aceves propose an interesting modification, which replaces the round trip delay measurements of TCP with estimations of delay along the forward path, and use an operating point for the number of packets in the bottleneck. In **TCP-Real** [44] Zhang and Tsaoussidis propose modifications to (i) enhance the real-time capabilities of the protocol over wired/wireless networks and (ii) tackle the problem of asymmetry by decoupling the size of congestion

window from the timeout. TCP-Real implements the wave mechanism presented in [41]. Congestion control is now receiver-oriented, and the congestion window is included in the TCP header in order for the receiver to communicate with the sender. As noted earlier, the receiver observes the network dynamics based on the ratio of the wave size to the wave delivery time.

Since the receiver is aware of the size of the current wave sent, it can further estimate the level of loss and the changes in current conditions by measuring the number of successfully delivered segments within a wave and the wave delivery time, respectively. Based on the observation of the forward-path dynamics, the receiver directs the sender's congestion control. The congestion window (at the sender) is not adjusted backwards when problems are identified in the reverse direction; the RTT however is taken into account for the calculation of the timeout in order to avoid duplicate packets that unnecessarily increase the overhead. More precisely, in TCP-Real [44] the timeout can be extended but the window size could remain the same or even increase. The reasoning behind this strategic modification is that the sender needs to extend the timeout based on the RTT measurements, in order to accommodate potential delays on the reverse path and avoid an early timeout. However, only the perceived congestion of the forward path will determine the sender's congestion window size. Congestion control in TCP-Real has therefore two additional properties: (i) it can avoid unnecessary congestion window adjustments due to path asymmetry, and (ii) it can determine the level of loss and jitter with better precision since the size of the sending window is known to the receiver (i.e., the wave pattern).

Determination of the most appropriate way to use the information about packet jitter and wave delivery time remains an open question; this issue is not tailored specifically to the protocols discussed above, but refers to error detection mechanisms in general. For example, the information about the delay variation and the pattern of the detected gaps could be potentially a useful feedback for the recovery strategy. Further issues of research in error detection have also been raised from the design of the above protocols. Both the Wave-and-Wait Protocol (WWP) [39] and TCP-Real measure the time between the delivery of the first and last segment of the wave; more sophisticated techniques could have been applied here as well. For example, what should be the corresponding window size based on the level of congestion detected, how many levels are appropriate to enable an efficient utilization of the bandwidth and feed a simple and efficient algorithm for recovery?

5.2 Acknowledgment Strategy

It is important to note that the acknowledgment strategy contributes to error detection. It is presented here mainly because it does not contribute much to error classification and, in addition, because it directs the behavior of the congestion window. That is, TCP is an ack-clocked protocol and the acknowledgment strategy determines, in effect, the transmission rate. The standard protocol's strategy is to acknowledge each packet received. It has been shown that, occasionally, it can be beneficial for the sender to communicate with a receiver that delays the generation of acknowledgments. The mechanism is widely known as "Delayed Acks". Delaying Acks reduces the processing overhead and also the communication overhead. Many TCP implementations acknowledge only every K^{th} segment out of a group of segments that arrive within a short time interval. The TCP sender must measure the effective RTT, including the additional time due to delayed ACKs, or else it will retransmit prematurely [42].

The delayed acknowledgments approach could be potentially a useful mechanism for controlling the sender's transmission rate at the receiver. Similarly, the pace of the TCP sender can also be adjusted by a network device in accordance with the current conditions detected. For example, the **fast-retransmission** approach [14] reduces the effect of mobile host hand-off. During a mobile host hand-off from one base station to another, TCP segments can be lost or delayed, and the source can timeout. Because of the typical coarse granularity of the TCP clock, the timeout period is much higher than hand-off time, and the mobile host has to unnecessarily wait for a long duration to receive a retransmission of a lost TCP segment from the source. In the fast retransmit approach, immediately after completing the hand-off, the IP in the mobile host triggers TCP to generate a certain number of duplicate acknowledgments. Since most of TCP implementations now have fast-retransmit, these duplicate acknowledgments cause the source to retransmit the lost segment without waiting for the timeout period to expire.

TCP makes use of duplicate acknowledgments mainly to indicate the next in-sequence byte expected. However, indirectly, a DACK is an indication that packets are not stuck at some bottleneck device. Fast Recovery is designed based on this dynamic. A question arises for the sender on how to interpret acknowledgements. **New Reno** for example, perceives certain acknowledgments as *partial* [18]: A partial acknowledgment is defined as an acknowledgment 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 have been received. In Reno, the partial ack triggers exit from Fast Recovery. In New Reno, it is an indication that (at least) one segment is missing and needs to be retransmitted. When multiple packets are lost from a window of data, New Reno can recover without waiting for a retransmission timeout. Under rather specific conditions [38], this results in some performance improvement. Although New Reno addresses the problem of multiple segment drops within the same window, and can thus avoid many of the retransmit timeouts of Reno, the algorithm still retransmits at most one segment per RTT. Furthermore, the retransmission triggered by a partial ack might be for a delayed

rather than lost segment. Thus, the strategy risks making multiple transmissions for the segment, which can seriously impact its energy efficiency with no compensatory gain in goodput. Results in [38] demonstrate this inefficiency of New Reno.

Enhancements of the TCP acknowledgment strategy are also discussed in [27, 1, 5, 17]. **TCP-SACK** [27] uses selective acknowledgments to enable the receiver to inform the sender about the blocks of segments that have been successfully received. Selective acknowledgments are used in conjunction with a Selective Repeat strategy: the missing segments can be retransmitted in one RTT. Practically, the size limitation of the options field of the TCP header constitutes a limitation on the number of blocks that can be reported and hence, for the number of segments that can be retransmitted. It is interesting to note that Selective Acknowledgments provide more information to the sender but do not enforce a more aggressive retransmission. Although it is recommended [17] that the Selective Repeat mechanism be implemented, it would be interesting to investigate when exactly this is a good practice; when error conditions do not necessitate an aggressive behavior, this strategy might have negative impact on protocol and/or network performance [38]. However, this issue has not been investigated for TCP-SACK and there are no results that the authors are aware of to support this hypothesis.

It should be noted that SACK can be beneficial anyway: the additional information can be used to trigger a conservative behavior that avoids extra overhead not only because the sender can back-off during deteriorated conditions, but also because retransmission can be more precise and selective. Of course, the overhead that is added by SACK itself needs to be considered in the evaluation of the protocol performance. Finally, it is interesting to note that TCP-SACK exhausts all the available space of the header's options. That is, it does not allow space for future evolution of TCP or for combination of additional functions that might need to be simultaneously negotiated between the two TCP ends.

5.3 Wired-Wireless Gateway

Most proxy-based solutions, typically implemented at the base station at the wired-wireless boundary, attempt to hide the effect of wireless losses from TCP. This is achieved by either avoiding timeouts and fast retransmit (and hence window shrinkage) at the sender, or avoiding unnecessary increases in timeout (and hence delayed reaction to congestion). These solutions affect the recovery at the sender either implicitly or explicitly. For example, implicit approaches include dropping dacks resulting from wireless losses so that fast retransmit is not trigerred, or setting ZWA in acknowledgments to freeze TCP so that timeout is not trigerred. Explicit approaches include sending ECN-like notifications, which require code modification (at least) at the sender. We next discuss some proxy-based solutions.

The **Indirect-TCP** (I-TCP) [4] proposal splits the transport link at the wireline–wireless border so as to completely shield the sender from the effect of wireless losses. The base station maintains two TCP connections, one over the fixed network, and another over the wireless link. This way, the poor quality of the wireless link is hidden from the fixed network. By splitting the transport link, I-TCP does *not* maintain end-to-end TCP semantics, i.e. I-TCP relies on the application layer to ensure reliability. It is argued that many applications provide end-to-end reliability, and one can choose I-TCP if the application provides reliability, and choose TCP otherwise. Thus, the mobile host must be aware of an application's ability to provide reliability when choosing a proper protocol for the transport layer. Provided the mobile host can choose the appropriate transmission control protocol, the base station should be informed about the mobile host's selection as it is where I-TCP executes.[4] Since I-TCP uses TCP over the wireless link, it suffers from poor performance in dealing with wireless losses.

The **MTCP** proposal [13] is similar to I-TCP, except that the last TCP byte of the data is acknowledged to the source only after it is received by the mobile host. Although the source falsely believes that every data byte except the last byte is received by the receiver, it can take remedial action based on whether the last byte is received or not. In particular, if the acknowledgement for the last byte is not received by the source, it has to resend all the data including those that may have already been received by the mobile host. By holding on to the last byte, the base station can send ZWA to freeze the source during handoffs, so the window and timeout are not affected.

Like other approaches, **Explicit Bad State Notification** (EBSN) [6] uses local retransmission from the base station to shield wireless link errors and improve throughput. However, if the wireless link is in error state for an extended duration, the source may timeout causing unnecessary source retransmission. The EBSN approach avoids source timeout by using an explicit feedback mechanism. If the wireless link is in bad state, the base station sends an EBSN message to the source for every retransmission of a segment to the mobile host. The EBSN message causes the source to reinitialize the timer. If the base station sends the EBSN messages before the source timer expires, then there will be no timeouts at the source. However, the main disadvantage of this approach is that it requires TCP code modification at the source to be able to interpret EBSN messages.

Ratnam and Matta [34, 33] proposed **WTCP** to hide the time spent by a TCP segment in the base station buffer (cf. Section 4). Thus, RTT estimates and timeout maintained at the sender (and consequently the sender's ability to detect

[4] Relying on the base station to decide on the transport protocol is out of the question as this requires the base station to be aware of the type of application passing through it.

wired congestion losses) are not affected by wireless losses. This is achieved without explicit feedback messages, rather by modifying the timestamp field in acknowledgments. In WTCP, the TCP connection from the source is terminated at the base station, and another reliable connection is created from the base station to the mobile host. However, the base station acknowledges a TCP segment to the source only after that segment is acknowledged by the mobile host. This way, TCP's end-to-end semantics are completely preserved.

The reliable connection from the base station to the mobile host takes into account the unique characteristics of the local wireless link. In case of timeout, the transmission window for the wireless connection is reduced to just *one* segment assuming a typical burst loss on the wireless link is going to follow. Thus, by rapidly reducing the transmission window, potentially wasteful wireless transmission is avoided and the interference with other channels is reduced. Unlike regular TCP, each time an acknowledgment is received, WTCP opens the wireless transmission window completely assuming that an acknowledgement indicates that the wireless link is in good state. That is, when an acknowledgment is received, the transmission window size is set to the window size advertised by the receiver (i.e. mobile host).

TCP Reno assumes that a duplicate acknowledgment is caused by a TCP segment being lost due to congestion and the connection is put in the congestion avoidance phase. However, for duplicate acknowledgment, WTCP does *not* alter the wireless transmission window assuming that the reception of the duplicate acknowledgement is an indication that the wireless link is in good state, and immediately retransmits the lost segment. The base station continues to transmit the remaining segments that are within the transmission window if they have not already been transmitted. However, until the mobile host receives the lost segment, the reception of each out-of-order segment will generate a duplicate acknowledgment. The number of these additional duplicate acknowledgments can be determined by the base station, which ceases to retransmit during their reception. By avoiding more than one duplicate acknowledgment based retransmission for a segment, WTCP attempts to improve the utilization of the wireless channel.

Snoop [9] is similar to WTCP, except that it is implemented at the link layer of the base station. The base station sniffs the link interface for any TCP segments destined for the mobile host, and buffers them if buffer space is available. Source retransmitted segments that have already been acknowledged by the mobile host are not forwarded by the base station. The base station also sniffs into the acknowledgments from the mobile host. If the base station sees a duplicate acknowledgment, it detects a segment loss and if it is buffered then Snoop identifies a loss over the local wireless link, retransmits the lost segment and starts a timer. The duplicate acknowledgment is also dropped to avoid unnecessary fast retransmission at the sender.

5.3.1 Design Issues in Proxy-based Approaches

An important design decision in a proxy-based solution is whether it should be based on explicit or implicit feedback. An implicit approach attains three comparative assets: (i) induces less overhead at the end-hosts and network, (ii) requires no code modification to process any explicit messages, and (iii) maintains the end-to-end semantics of TCP. Then, a primary goal is to effectively shield wireless losses from end-hosts. This is typically done by buffering data segments at the proxy and retransmitting them over the local wireless link if they get lost due to transmission errors. In these cases, window shrinkage at the TCP sender should be avoided, for example by dropping associated duplicate acknowledgments. An ideal proxy-based solution should also shield the effect of wireless losses on the retransmission timeout maintained at the sender. The timeout should not expire too soon before the proxy has the chance to locally repair wireless losses, but also should not be extended too much that it hinders TCP's ability to quickly react to changes in congestion conditions. How to optimally adjust the timeout of TCP over wireless links is still an open research question.

Proxy-based solutions should be designed to satisfy these design goals. Figure 2 shows the throughput of I-TCP, Snoop, WTCP and TCP-tahoe[5]. Throughput is measured as the average number of bits successfully transmitted to the mobile host in one second. The simulated network topology is shown in Figure 1. We model the wireless link from the base station (proxy) to the mobile host (client) using a two-state Markov channel [11], where the channel is in either good state or bad state. In the good state the bit error rate (BER) of the channel is low, and in the bad state BER is high. We simulated the wireless link with bit error rates of 10^{-6} and 10^{-2} in good and bad states, respectively. The time spent in each state is modeled by an exponential distribution with mean good state duration fixed at 1 second. The mean bad state duration is varied from 10 ms to 100 ms. We assume no losses on the wired link.

For a good quality wireless link, the throughput of I-TCP is better. When the wireless link quality degrades WTCP yields better throughput mainly because of its aggressive retransmission policy over the wireless link. WTCP achieves throughput values 4-8 times higher than TCP-tahoe. However, the aforementioned policy degrades (slightly) the utilization of the wireless link [34, 33]. An open research question is whether throughput gain outweighs the small loss in link utilization (and associated wasted energy).

[5]TCP-tahoe is the traditional end-to-end TCP implementation, and is shown here to illustrate the performance gains of employing proxy-based solutions. We do not show results for TCP-reno as it was found to be less robust than TCP-tahoe in wireless applications [26].

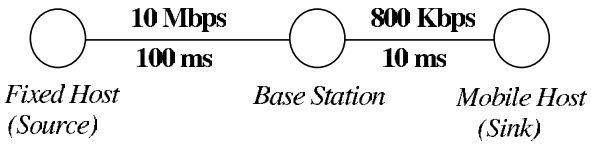

Figure 1. Simulated network.

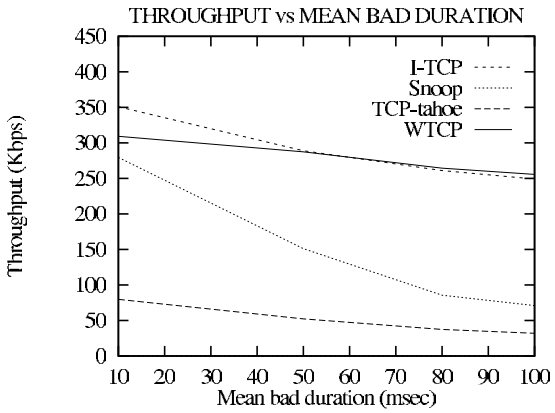

Figure 2. Throughput of different protocols.

Though Snoop achieves throughput values comparable to I-TCP and WTCP at low loss situations, the throughput of Snoop is poor when the wireless link is very bursty (in bad error state) for long durations. The reason is that Snoop triggers retransmission only after the base station receives a duplicate acknowledgment. When the wireless link is in a bad state, acknowledgments might be lost. In this scenario, the base station fails to trigger DACK-based retransmission, and the throughput degrades.

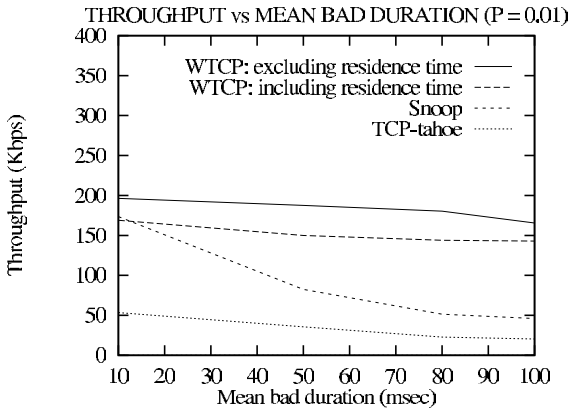

Figure 3. WTCP throughput with wired (congestion) link loss probability $P = 0.01$

Recall that WTCP hides the residence time in the base station buffer by modifying the timestamp in acknowledgments. Otherwise, the timeout value maintained at the source will be high. As pointed out earlier, this affects the TCP source's ability to effectively detect any congestion losses in the wired network portion of the connection. To demonstrate this effect a lossy wired link is modeled to randomly lose data packets with probability *P*. Figure 3 shows the throughput for WTCP with and without modifying the timestamp to exclude the residence time, as well as the throughput for Snoop and TCP-tahoe. The wireless link is modeled by a two-state error channel as before, and the mean good period is 1 second. The advantage of hiding the residence time at the base station becomes even more pronounced for higher wired (congestion) loss probabilities. However, when there are only few wired link losses (Figure 4), as expected, the number of data segments lost is low, and hence excluding the residence time does not provide significant improvement in throughput.

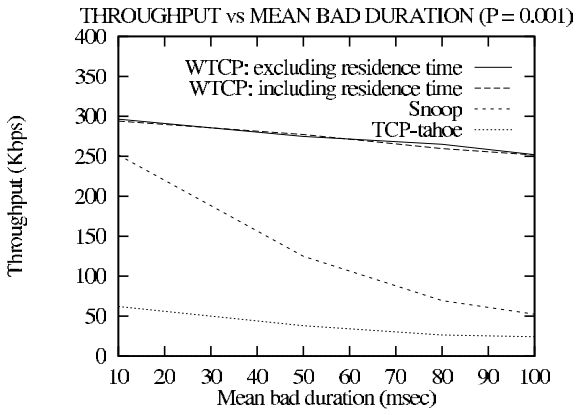

Figure 4. WTCP throughput with wired (congestion) link loss probability $P = 0.001$

All the above results indicate that proxy-based solutions can improve the performance of reliable transport over wireless links. However, further investigation is needed to balance the retransmission timeout at senders so as to satisfy two conflicting goals: (1) keep the timeout long enough to provide proxies with sufficient time to perform local recovery, and (2) keep the timeout short enough to not adversely affect the congestion detection capability of TCP. This research has to consider the effect of propagation delays between senders and proxies, and between proxies and their clients. In addition, the effect of congestion in contention-based wireless links needs further study. For example, a WTCP proxy may not exclude the local recovery time if the segment residence time was actually due to congestion on the wireless portion. The issue of energy consumption, by and large, has also not been studied (see Section 6.1.1).

6 Protocol Strategy

6.1 Toward Adaptive Error Control

Congestion Control encompasses the characteristics of adaptive behavior: the sender adjusts the transmission rate according to the current level of available bandwidth. Adaptive *error control* introduces another type of adjustment: the protocol's strategy. That is, the strategy of congestion control is always to back off. An error control that is responsive to various types of errors requires a strategy that allows for behavioral flexibility, from conservative *through* aggressive. Thus, an adaptive strategy encompasses adaptive behavior that is different for each error type. Tsaoussidis *et al.* [38] compare Tahoe, Reno and New Reno from the perspective of energy and goodput efficiency under varying error conditions. The aforementioned protocols exhibit a conservative, moderated and more aggressive behavior, respectively and are appropriate candidates for testing the impact of aggressiveness of a recovery strategy in response to the error characteristics. The results of [38] show that a conservative strategy yields better results when the error density over a wireless channel increases, while an aggressive strategy appears more appropriate at low error rates.

Hence, a simplification of the sort "Conservative in wired / Aggressive in wireless" could not conduct an efficient recovery. In this context, Reno's strategy, for example, to always back off in the presence of errors cannot display the rate-adaptive pattern required to match the conditions of the channel. Reno behaves occasionally conservatively when the channel permits more aggressive transmission, and relatively aggressively when it forcefully attempts to transmit half window in the presence of a "bad" channel. Reno follows this strategy in response to all errors. It is important to note here that TCP's strategy is not clearly geared toward a conservative or aggressive retransmission; the protocol occasionally exhibits a conflicting[6] behavior that may not conform to the strategic goals of a protocol for wireless networks. The conflict arises from the two distinct components of TCP's error control, namely the timeout and the congestion window. For example, while the back-off strategy during burst errors or hand-offs may be appropriate, the timeout extension does not contribute to efficient recovery. This contradictory behavior stands out from the comparison of Reno and Tahoe. In Fast Recovery, Reno halves the congestion window in the presence of three dacks. This behavior is more aggressive than Slow Start. If the channel conditions do not favor such aggressive strategy, the timeout of Reno might be extended further than the one of Tahoe. Thus the attempt to implement a more aggressive strategy than Tahoe may produce an adverse result.

[6]Conflicting only in the context of wired/wireless networks.

The "probing device" presented in Section 4 models exactly this desired property of adaptive error control and avoids the conflict between the timeout and the congestion window. 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 "sitting out" these error conditions awaiting successful completion of the probe cycle. In the case of transient random loss, however, the probe cycle will complete much more quickly, in proportion to the prevailing density of the occurrence of the random errors.

The results we present below indicate the advantage of TCP-Probing's adaptive strategy over the fixed strategy of Reno for a 5-Mbyte file transmission task. The results are based on tests that were carried out in two stages. Initially in a single session, with the client and the server running on two directly-connected dedicated hosts, so as to avoid unpredictable conditions with distorting effects on the protocol's performance. Each version of the protocol was tested by itself, separately from the others, so that its error-control mechanism can demonstrate its capability without being influenced by the presence of other flows in each channel. We simulated error conditions of different intensity and duration in order to evaluate the protocols' performance in response to changes in that environment. In order to have wireless error patterns incorporated into the experiments (i.e., path loss, slow fading, fast fading, noise/interference from other networks/devices and handoffs), we ran the tests across a range of error rates and phases that correspond to packet error rates (PER) using a two-state continuous-time Markov chain. The error model was configured between the TCP and IP layer and was integrated in the *xkernel* [36] protocol framework. Each state has a mean sojourn time m_i and a drop rate r_i ($i = 1, 2$). The drop rate r_i takes a value between 0 and 1, and determines the proportion of packets to be dropped during state i. Thus, when state i is visited, 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. One state representing the "good" state was configured with a zero drop probability.

At the next stage we have used an external, cross-traffic, TCP-based application protocol to exploit the channel's bandwidth all by itself, in order to observe the protocols' behavior in an environment with varying error and delay patterns. During these experiments, drops were caused by congestion and/or by the simulated link deficiencies. This set of experiment allowed us to draw conclusions on the capabilities of probing in distinguishing the nature of the error and to respond appropriately. More precisely, the aim here was to test the capability of TCP-Probing to respond aggressively whenever the error rate permitted an aggressive behavior, and more conservatively whenever moderated congestion was present.

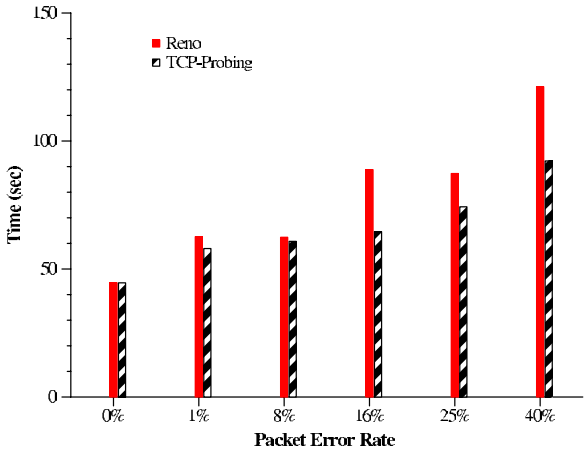

Figure 5. Task Completion Time of Reno and TCP-Probing.

It is instructive to consider how the probing and Immediate Recovery mechanisms provide TCP-Probing with the behavioral flexibility underlying its superior performance in Figures 5 and 6. At relatively low error rates (1% to 8%), TCP-Probing is able to expand its window in the Good phase. During the Bad phase, its probing mechanism allows it to explore windows of opportunity for error-free transmissions, which are then exploited by Immediate Recovery. TCP-Probing is effectively backing off for the duration of the probe cycle, but is also capable of rapid window adjustments with Immediate Recovery where appropriate. In contrast, Reno's mechanism is exclusively focused on congestion control. The idea is to alleviate congested routers and avoid flooding the network, so Reno's mechanism does not allow for rapid window adjustments as a recovery strategy after backing off.

It can be observed from Figure 6 that the impact of delay is even more significant on standard TCP. The cogitation that probing requires two RTTs (as opposed to several RTTs required by standard TCP's graduated adjustments) and then

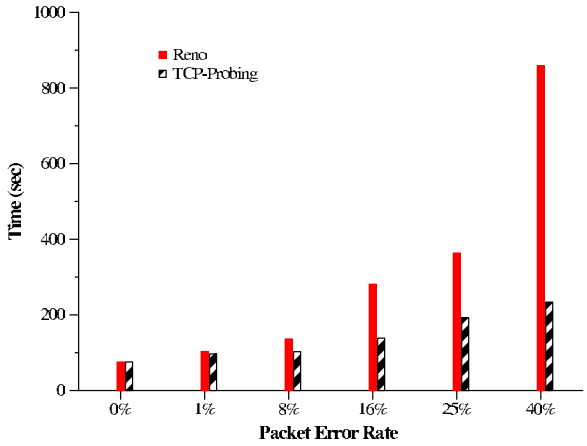

Figure 6. Task Completion Time of Reno and TCP-Probing with additional 100ms propagation delay.

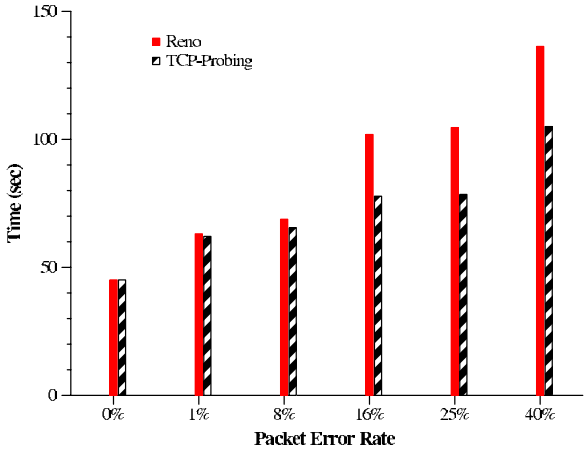

Figure 7. Task Completion Time performance with 50ms propagation delay and Congestion.

Immediate Recovery ensues, along with the consideration that longer delay denotes a large Delay×Bandwidth product and hence window size, justifies the results.

Probing in general is not a new concept and it has been proposed in the past as a mechanism to measure current network conditions (e.g. [45]). Probing here explores a fundamental property that is indeed novel: it is used as a device that implements the protocol's adaptive strategy by shaping data transmission to conform to the load *and* error characteristics of the network. More precisely, probing is self-adjusted (extended) in proportion to the density of the error and hence transmission rate is shaped accordingly. Clearly, adaptive error control does not have only one possible design. Likewise, the desired properties of the probing device allow for different designs. The results presented above call for further research. For example, there are situations where neither an aggressive nor conservative strategy could be clearly indicated or implemented by probing. For example, as the protocol exhibits a more conservative strategy (the error becomes more dense) the relative performance gain of TCP-Probing drops (see Figure 5 at the 25% error rate). A different design of the probing device could implement a different strategy. Or, the probing device itself could be designed to be adaptive, for example, to extend the required number of cycles as the error becomes more dense, resulting in a more conservative strategy earlier than the point indicated by the current design.

6.1.1 Combining Goodput and Energy Efficiency

One distinctive characteristic of adaptive strategies that span across a wide range of behaviors in the conservative/aggressive scale is how they manage the tradeoff between effort expended and goodput achieved, or otherwise stated, how to optimize the goodput/overhead ratio. The ratio exploits interesting dynamics: although one might expect that expending more transmission

effort should result in better goodput, [40, 46] show that this is not always true. Consider for example the overhead expended by TCP-Probing (see Figures 8 and 9) in order to achieve the performance gains depicted in Figures 5 and 6. We observe that the overhead of TCP-Probing never exceeds the overhead of Reno. Note that overhead includes probes, retransmitted packets as well as protocol header overhead. Notably, the overhead of TCP-Probing relative to Reno decreases when the propagation delay increases due to the fact that opportunities for clear retransmission are not wasted and communication time is not extended.

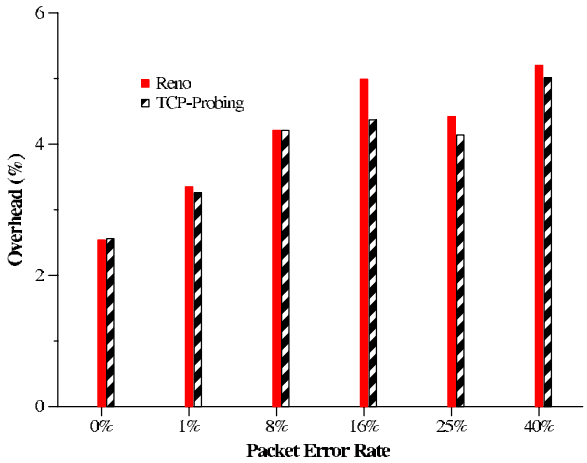

Figure 8. Overhead of Reno and Probing with wireless errors.

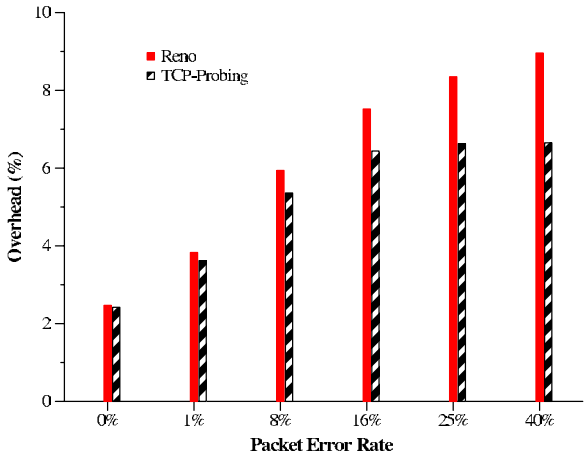

Figure 9. Overhead of Reno and Probing with additional 100ms propagation delay and wireless errors.

This goodput/overhead ratio is a particularly useful performance metric for applications on mobile, battery-powered devices since it reflects the energy efficiency of the protocol. That is, the overhead and time metrics combined can provide sufficient knowledge of a protocol's energy-saving capabilities. 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, or are idle. Hence, an estimation of additional energy expenditure cannot be based solely on the byte overhead due to retransmission, since overall connection time might have been extended while waiting or while attempting only a moderate rate of transmission. On the other hand, the estimation cannot be based solely on the overall connection time either, since the distinct operations performed during that time (e.g. transmission vs. idle) consume different levels of energy. Nevertheless, the potential for energy saving can be gauged from the combination of time and byte overhead savings achieved. Indeed, if we consider that

several devices are not optimized to adjust the power of the transmitting/receiving module whenever the device does not actually transmit/receive data, and also that TCP's operations do not involve any "idle" state by default[7], communication time seems to be the most significant factor. Determination of the energy function which is appropriate to describe TCP's operation on specific devices is an ongoing research work of the authors.

7 Conclusion

We presented open issues and challenges in the design of reliable transport over heterogeneous wired/wireless networks. We discussed recent research results that expose these issues, provide preliminary solutions, and call for further investigation. A traditional approach to describe such reliable transport-over-wireless research is to classify the proposed solutions either in terms of where new functionality is placed or in terms of whether or not the solution is end-to-end. The first classification determines whether code modifications are done at the sender, receiver, intermediate node(s), or combinations thereof. The second classification determines the recovery actions that are taken either by the end-hosts only (i.e. sender and receiver) or by additional intermediate proxy/node(s).

In this paper, we take a non-traditional approach to discuss design issues as they relate to the different protocol functions, namely, error detection and error correction. We argued for an error detection that is capable of classifying different kinds of error (e.g. congestion, transient wireless errors, persistent wireless errors). Based on this error classification, an error correction involves appropriate recovery actions that might differ from congestion-oriented mechanisms employed by TCP. We also argued for a protocol recovery strategy that flexibly adapts its behavior to accommodate various tradeoffs. An important tradeoff for battery-powered devices involve energy consumption and goodput performance. We highlighted the importance of defining new metrics that capture such tradeoffs and that could be used to evaluate aspects of stability and fairness in heterogeneous wired/wireless networks.

References

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

[2] M. Allman, C. Hayes, H. Kruse, and S. Ostermann. TCP Performance over Satellite Links. In *Proceedings of the 5th International Conference on Telecommunication Systems*, March 1997.

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

[4] A. Bakre and B. Badrinath. I-TCP: Indirect TCP for Mobile Hosts. In *Proceedings of the IEEE ICDCS '95*, pages 136–143, 1995.

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

[6] Bikram S. Bakshi, P. Krishna, N. H. Vaidya, and D. K. Pradhan. Improving Performance of TCP over Wireless Networks. Technical Report 96-014, Texas A & M University, 1996.

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

[8] 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.

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

[10] I. Batsiolas and I. Nikolaidis. Selective Idling: An Experiment in Transport-Layer Power-Efficient Protocol Implementation. In *Proceedings of the International Conference on Internet Computing*, June 2000.

[11] Pravin Bhagwat, Partha Bhattacharya, Arvind Krishna, and Satish K. Tripathi. Enchancing Throughput over Wireless LANs using Channel State Dependent Packet Scheduling. In *IEEE INFOCOM'96*, pages 1133–40, March 1996.

[7] [10] shows that this is feasible and beneficial in some cases

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

[13] Kevin Brown and Suresh Singh. M-TCP: TCP for Mobile Cellular Networks. In *Proceedings of the ACM SIGCOMM Computer Communication Review*, pages 19–43, 1997.

[14] Ramon Caceres and Liviu Iftode. Improving the Performance of Reliable Transport Protocol in Mobile Computing Environment. *IEEE Journal of Selected Areas in Communications*, 13(5), June 1995.

[15] D. Chiu and R. Jain. Analysis of the Increase/Decrease Algorithms for Congestion Avoidance in Computer Networks. *Journal of Computer Networks*, 17(1), June 1989.

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

[17] K. Fall and S. Floyd. Simulation-based Comparisons of Tahoe, Reno, and SACK TCP. *Computer Communication Review*, 26(3):5–21, July 1996.

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

[19] S. Floyd and V. Jacobson. Random Early Detection Gateways for Congestion Avoidance. *IEEE/ACM Transactions on Networking*, August 1993.

[20] T. Goff, J. Moronski, and D. Phatak. Freeze-TCP: A True End-to-End Enhancement Mechanism for Mobile Environments. In *Proceedings of the INFOCOM*, 2000.

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

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

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

[24] 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*, pages 336–350, June 1997.

[25] T. Lakshman and U. Madhow. TCP/IP Performance with Random Loss and Bidirectional Congestion. *IEEE/ACM Transactions on Networking*, 8(5):541–555, October 2000.

[26] T.V. Lakshman and Upamanyu Madhow. The Performance of TCP/IP for Networks with High Bandwidth-Delay Product and Random Loss. *IEEE/ACM Transactions on Networking*, 5(3), June 1997.

[27] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. TCP Selective Acknowledgement Options. *RFC 2018*, April 1996.

[28] M. May, T. Bonald, and J. Bolot. Analytic Evaluation of RED Performance. In *INFOCOM 2000*, August 2000.

[29] C. Parsa and G. Aceves. Improving TCP Congestion Control over Internets with Heterogeneous Transmission Media. *IEEE International Conference on Network Protocols (ICNP '99)*, 1999.

[30] C. Partridge and T. Shepard. TCP Performance over Satellite Links. *IEEE Network*, 11(5):44–99, September 1997.

[31] J. Postel. Transmission Control Protocol. *RFC 793*, September 1981.

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

[33] Karu Ratnam and Ibrahim Matta. Effect of Local Retransmission at Wireless Access Points on the Round Trip Time Estimation of TCP. In *Proceedings of IEEE 31st Annual Simulation Symposium '98*, April 1998.

[34] Karu Ratnam and Ibrahim Matta. WTCP: An Efficient Mechanism for Improving TCP Performance over Wireless Links. In *Proceedings of the Third IEEE Symposium on Computer and Communications (ISCC '98)*, June 1998. Code available from http://www.cs.bu.edu/faculty/matta/software.html.

[35] J. H. Saltzer, D. Reed, and D. Clark. End-to-End Arguments in System Design. *ACM Transactions on Computer Systems*, November 1984.

[36] The x-kernel Protocol Framework. www.cs.princeton.edu/xkernel.

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

[38] V. Tsaoussidis, H. Badr, X. Ge, and K. Pentikousis. Energy / Throughput Tradeoffs of TCP Error Control Strategies. In *Proceedings of the 5th IEEE Symposium on Computers and Communications (ISCC)*, 2000.

[39] V. Tsaoussidis, H. Badr, and R. Verma. Wave & Wait Protocol (WWP) An Energy-Saving Transport Protocol for Mobile IP-Devices. In *Proceedings of the 7th International Conference on Network Protocols*, 1999.

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

[41] V. Tsaoussidis, A. Lahanas, and C. Zhang. The Wave and Probe Communication Mechanisms. *The Journal of Supercomputing*, 20(2), September 2001.

[42] D. Borman V. Jacobson, R. Braden. TCP Extensions for High Performance. *RFC 1323*, May 1992.

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

[44] C. Zhang and V. Tsaoussidis. TCP Real: Improving Real-time Capabilities of TCP over Heterogeneous Networks. In *Proceedings of the 11th IEEE/ACM NOSSDAV 2001, New York*, 2001.

[45] M. Zorzi and R. Rao. Error Control and Energy Consumption in Communications for Nomadic Computing. In *IEEE Transactions on Computers (Special Issue on Mobile Computing)*, March 1997.

[46] M. Zorzi and R. Rao. Is TCP Energy Efficient. In *Proceedings of the MoMUC '99, San Diego, California*, November 1999.