# On protocol engineering: detect, confirm and adjust

I. Psaras*, L. Mamatas and V. Tsaoussidis

Department of Electrical and Computer Engineering,
Demokritos University of Thrace,
Xanthi, Greece
E-mail: ipsaras@ee.duth.gr E-mail: emamatas@ee.duth.gr
E-mail: vtsaousi@ee.duth.gr
*Corresponding author

**Abstract:** In this paper, we depart from TCP-Probing (Tsaoussidis and Badr, 2000) and propose an experimental transport protocol that achieves energy and throughput performance gains in mixed wired and wireless environments. Our approach decouples error recovery from contention estimation and focuses on the way these two mechanisms can (i) feed the probing decision process and (ii) implement the protocol strategy by shaping traffic, according to detected conditions. We use a validation mechanism that uncovers previous possibly wrong estimations. Our analysis matches well our simulation results that are very promising.

**Keywords:** TCP; probing; energy; wireless; heterogeneous; error recovery; error classification; error estimation.

**Reference** to this paper should be made as follows: Psaras, I., Mamatas, L. and Tsaoussidis, V. (xxxx) 'On protocol engineering: detect, confirm and adjust', *Int. J. Internet Protocol Technology*, Vol. x, No. x, pp.xxx–xxx.

**Biographical notes:** Ioannis Psaras graduated in 2004 from the Department of Electrical and Computer Engineering, Demokritos University of Thrace. Currently, he is a PhD student at Demokritos University under the supervision of professor Vassilis Tsaoussidis. His research interests are focused on congestion control in packet networks, transmission over wireless links and traffic shaping in heterogeneous networks. He received the Ericsson Award of Excellence in Telecommunications for the best diploma dissertation. He is a reviewer in the *Journal of Computer Networks (Elsevier), Transactions on Mobile Computing* (*IEEE*) *and in WWIC 2005* (also chair of the organising committee), WLN 2005.

Lefteris Mamatas is a PhD student in the Demokritos University of Thrace (supervisor: Vassilis Tsaoussidis). He graduated in 2003 from the Department of Electrical and Computer Engineering at Demokritos University of Thrace. He has published eight conference papers. He has been awarded three times in computer science competitions. His research interests lie in the area of transport protocols for heterogeneous wired/wireless environments.

Vassilis Tsaoussidis received a BSc in Applied Mathematics from Aristotle University, Greece; a Diploma in Statistics and Computer Science from the Hellenic Institute of Statistics and a PhD in Computer Networks from Humboldt University, Berlin, Germany (1995). Vassilis held faculty positions in Rutgers University, University of New Brunswick, SUNY Stony Brook and Northeastern University, Boston. In May 2003, Vassilis joined the Department of Electrical and Computer Engineering of Demokritos University, Greece. His research interests lie in the area of transport/network protocols, i.e., their design aspects and performance evaluation.

## 1 Introduction

TCP is the most widely used protocol for reliable data transmission over the internet. The Transmission Control Protocol was designed in days when the existing network infrastructure was based solely on wired components. Under these circumstances, TCP (Allman et al., 1999) was required to deal with problems such as fairness in bandwidth consumption of the competing flows and congestion control. However, the internet started growing in size and, in addition, much of its infrastructure became wireless. Today, the internet can be described as a fully heterogeneous internetwork.

Heterogeneity spans across a variety of components: from the network's type (wireless, satellite, etc.) and the network's speed (e.g., high-speed) to protocol, device and infrastructure heterogeneity (e.g., TCP variants, battery-powered devices and ad hoc networks), respectively.

One major task of end-to-end transport protocols is to hide this heterogeneity from end users. However, TCP is facing serious performance problems, regarding its capability to deal with the unique characteristics of a wireless network. In a wired network, a packet-drop is mainly due to congestion (i.e., buffer overflow), and this is where TCP's congestion control algorithms are focused on, while packet losses over wireless links are primarily due to fading channels, or handoff events. In the latter situation, TCP wrongly continues to behave under the rules of congestion control (Tsaoussidis and Matta, 2002). In this paper, we demonstrate the behaviour of a transport protocol, in response to different conditions, which discretely describes network dynamics.

We focus on some particular aspects of heterogeneity, and we confine the aforementioned broad perspective to a tractable simulation set of scenarios. More precisely, we focus:

- on last-mile wireless (Although a backbone can be wireless or satellite too, our present work focuses on wired backbone with wireless receiving-ends.)
- on the energy performance of devices
- on mobility patterns.

Initially, we make use of four simple scenarios that represent very common network conditions. This way, we monitor the behaviour of the protocols, for each one of these conditions, separately from the others. These four scenarios are focused on *congestion, contention decrease, handoff events* and *fading channels.* However, in the real world, a combination of all the above is possible. That said, we will present some additional, more complicated scenarios that simulate a more realistic network environment.

We are interested in end-to-end solutions that do not require any modification of the network's infrastructure. We propose an improvement to the core mechanism of the experimental protocol TCP-probing (Tsaoussidis and Badr, 2000; Tsaoussidis and Lahanas, 2003), and we present some results to show the potential of our work. We also use two new metrics introduced in Mamatas and Tsaoussidis (2004), in order to capture the protocol's behaviour in terms of energy expenditure and unexploited available resources.

In our perspective, TCP is trying to achieve three basic goals:

- *fairness*: every flow should be fair to all others in order to share the same channel
- *performance*: good performance is closely related to continuous discovery of available bandwidth
- *congestion avoidance*: overflowing the network and hence losing data segments due to congestion is a situation that should be avoided.

Reliable protocols use error control mechanisms, in order to achieve these goals. These mechanisms can be divided into error detection and error recovery mechanisms. In the widely deployed TCP versions (TCP-Tahoe (Stevens, 1997), TCP-Reno (Allman et al., 1999), TCP-NewReno (Floyd and Henderson, 1999) and TCP-SACK (Mathis et al., 1996)), timeouts and duplicate acknowledgments are interpreted as packet losses from the error detection mechanisms. Upon a packet loss, TCP adjusts downwards the sender's window size and extends the timeout period. In this way, it does not seem to differentiate the congestion window from the timeout period, an action that would be useful in many cases, especially in heterogeneous environments.

Below we discuss the distinctive characteristics of each, in turn. We mainly target on determining the appropriate responses of TCP, had it been able to detect them.

### *1.1 Congestion*

Congestion is a very common situation in packet networks. As noted earlier, in this situation and after a packet gets lost, TCP reduces the sender's congestion window and extends the timeout period. This situation appears mostly in wired networks, for which standard TCP has been optimised. However, even a wireless network, with high contention, may require similar response.

### *1.2 Contention decrease*

As already noted, in order to achieve good performance, a protocol should dynamically discover and immediately exploit the available bandwidth. Although congestion decrease is also common in heterogeneous networks, where bandwidth becomes available rapidly, here we study the protocol's performance, over a wired topology. In such a situation and after a packet loss, the appropriate behaviour of a protocol is to inspect the network's conditions and become aware of the available bandwidth. Backing off under such circumstances (as standard TCP would) might not be the right action.

### *1.3 Handoffs*

Another situation that appears quite often in wireless/mobile environments is the handoff event. During a handoff, no data can be transmitted through the link, and hence the appropriate action would be to suspend data transmission for the duration of the handoff. Furthermore, the timeout period has no reason to be extended, since there is no data exchange between the sender and the receiver, which means that there is no extra buffering, and as a result no additional queuing delay. Finally, the sender's congestion window adjustment depends largely on the level of contention after the handoff period. If no contention is indicated, there is no reason to shrink the window. Otherwise, a more conservative strategy may be appropriate. The absence of such an adaptive mechanism in TCP's error recovery impacts performance. For example, during a handoff period that lasts 5 RTTs, TCP would attempt to transmit

1 KB of data for every RTT, which means that a sum of 5 KB would be lost. In addition to the overhead (retransmitted packets plus TCP header bytes), extra energy consumption calls for more adaptive recovery strategies.

### *1.4 Fading channels*

Random transient errors due to fading channels form another situation that should be taken into consideration in a protocol's behaviour. Similar to the handoff event, in the presence of errors, there is no need to extend the timeout period. Random transient errors do not affect the router's buffering, leaving it at the same state prior to the error. The adjustment of the sender's window is a matter of strategy, since the rate of the errors varies. In high error rates, the shrinkage of the window seems to be the appropriate action, in order not to crash with a large congestion window, which means that a heavy payload will need to be retransmitted. However, in a situation of low error rate, where random transient errors do not occur very often, it seems that there is no need to back off (and reduce the window), since the probability of losing the next data segment is rather low.

Departing from the above four scenarios, we propose that TCP traffic should be shaped according to the nature of the error, in order for the transport protocol to achieve performance gains in heterogeneous networks.

Generally, we observe that in heterogeneous networks.

- Timeout should be growing, only in association with contention (Psaras et al., 2005). More precisely, a transient, random error does not call for either timeout or congestion window adjustment.
- An 'early' attempt to estimate contention (before the 3-DACKS or elapsed timeout) can lead to a more effective error classification strategy. When a packet is lost in a low-contention environment, a wireless error is clearly indicated.
- A false 'early' contention estimation can be filtered by a second-level contention estimation mechanism, which is deployed between packet-drop detection (3-DACKS or elapsed timeout) and error recovery.
- During a dense error or a handoff, a probing mechanism can reduce unnecessary overhead.

The rest of the paper is organised as follows. In Section 2, we describe the way we attempt to decouple the error recovery strategy from the contention estimation. In Section 3, we review some of the most recent proposals that focus on error recovery strategies and contention estimation mechanisms. Section 4 details the implementation of the probing mechanism together with the contention estimation predictor onto the standard TCP and the algorithm used to decide about the recovery strategy. In Section 5, we describe the experimental methodology used to carry out simulations. In this section, we also analyse in depth the new proposed metrics, that provide a better perspective of the protocol's behaviour especially when heterogeneity is present. Our simulation results are discussed in Section 6. We conclude the paper in Section 7.

## 2 Decoupling error recovery from estimation

All the existing, widely deployed TCP versions are basically trying to avoid congestion events and/or recover after congestion losses. The congestion control algorithm, used in TCP-Tahoe (Stevens, 1997), includes Slow-Start, Congestion Avoidance and Fast Retransmit. TCP-Reno (Allman et al., 1999) introduces Fast Recovery in conjunction with Fast Retransmit. TCP-NewReno (Floyd and Henderson, 1999) addresses the problem of multiple segment drops within a single window of data. In effect, it can avoid many of the retransmit timeouts of Reno. The TCP-SACK (Mathis et al., 1996) modification introduces a selective acknowledgment strategy.

Standard TCP is not able to distinguish between errors due to congestion and wireless errors. It assumes that every time a packet loss occurs, it is always due to congestion (Tsaoussidis and Matta, 2002).

By adding a probing mechanism onto standard TCP, we can achieve two more goals. First of all, a probing mechanism inspects the network load whenever an error is detected and in this way it enables error classification (i.e., due to congestion or wireless error). Furthermore, it can monitor the network, in order to measure the level of contention. Secondly, it suspends data transmission for as long as the error persists, implementing in that way an energy-efficient mechanism.

The general idea of the probing mechanism is as follows. Whenever a packet is lost, TCP-Probing (Tsaoussidis and Badr, 2000; Tsaoussidis and Lahanas, 2003) instead of retransmitting the whole data segment, as would standard TCP, suspends data transmission and enters a probe cycle. Probe segments carry no payload; they consist of only segment headers, being in that way energy efficient even if the error is persistent and the probing segment is lost. For example, in the event of a burst error or a handoff, little overhead will be added when losing a single probe segment, than when losing the full data segment, as would standard TCP.

Our approach decouples error recovery from contention estimation, since it estimates the contention level before the packet-drop. This 'early' congestion detection has the following three goals: (i) to avoid a wrong aggressive error recovery that leads to unnecessary overhead, (ii) to avoid a false freezing of the timeout and (iii) to skip needless probing cycles.

Standard probing uses the RTT cycles to measure the level of contention. Therefore, a measurement is taken only after probing is triggered. Here, we introduce an enhancement to standard TCP-Probing, namely a congestion predictor that comes along the lines of TCP-Vegas estimation algorithm. We call this congestion predictor, Vegas Predictor or Estimator. Vegas Predictor may allow us to make a judgement on whether congestion exists prior to

entering probe cycles or not. That is, the decision on the transmission rate and the ability of probing to skip error-prone phases of transmission are decoupled. This also allows for better performance since the predictor is more accurate than the two RTT measurements and saves communication (i.e., probing) overhead in case of congestion.

We use, additionally, the contention estimation mechanism, which is bounded in TCP-Probing as a validation to our previous estimation. A conservative error recovery strategy is followed when either of the two contention estimators show high contention.

### *2.1 Error recovery mechanism*

The probe cycle will not terminate until the network conditions are such that the sender can make two successive round-trip-time (RTT) measurements from the network. These measurements are helpful information that will be taken into account by the recovery strategy. If these measurements show that there is high level of contention, the recovery strategy will back off, as in Reno. Otherwise, if the measurements show that there is available bandwidth (either due to transient random error or after a handoff), the recovery strategy will immediately try to exploit it.

The sender enters a probe cycle when either of two situations apply: a timeout event occurs or three dacks are received. When the probe cycle completes, we gather information about the network conditions from the measured probe RTTs (this scheme is explained in detail later), and if there is available throughput capacity, TCP-Probing assumes that there is no need to adjust downwards neither the congestion window nor the Slow-Start threshold. So it picks up from the point where it was before. This is called 'Immediate Recovery'. Otherwise, the protocol enters the Slow-Start phase.

### *2.2 Contention estimation*

In a packet-switched network, a situation where no congestion happens ever is rather non-realistic. On the contrary, especially in a wired scenario, the primary problem is the high level of contention, which finally leads to congestion. Under those circumstances, since TCP-Probing is an experimental protocol that focuses on heterogeneous wired/wireless networks, we added one additional feature. On every single RTT, TCP-Probing calculates a 'congestion predictor', along the lines of TCP-Vegas algorithm (Brakmo et al., 1996; Brakmo and Peterson, 1995). We call this vegas_predictor. If this predictor is higher than an adjustable threshold, then TCP-Probing does not enter a probe cycle in the event of three dacks, even when the packet is finally lost. Instead, it enters Slow-Start as in Reno.

## 3 Related work

Many proposals tried to classify the losses through different estimation techniques. Barman and Matta (2002) proposed an improvement on TCP-NewReno, New Reno-FF, a technique that is based on the average and variance of the round-trip-time using a Flip-Flop filter, a filter that is augmented with history information. Another classification technique proposed by Liu et al. (2003) is based on the loss pairs measurement technique and Hidden Markov Models (HMMs). This technique is based on the fact that the delay distribution around wireless losses is different from the one around congestion losses. Another enhancement to TCP is introduced by Chandra et al. (2003) and is referred to as E-TCP. This protocol uses a new acknowledgement packet format and an agent to assist E-TCP. This technique makes TCP aware of the existence of wireless losses. Many researchers are working on reliable congestion predictors, in order to avoid packet losses due to congestion. Biaz and Vaidya (1999) implemented a receiver-oriented technique to distinguish congestion losses from corruption losses.

Bimodal congestion avoidance and control mechanism (Attie et al., 2003) computes the fair-share of the total bandwidth that should be allocated for each flow, at any point, during the system's execution. TCP-Jersey (Xu et al., 2004, 2005) operates based on an 'available bandwidth' estimator to optimise the window size when network congestion is detected. The Packet-Pair technique (Keshav, 2001) estimates the end-to-end capacity of a path, using the difference in arrival times of two packets of the same size travelling from the same source to the same destination. Recently, several investigations have also been carried out regarding the accuracy of the proposed estimators, like Jain and Dovrolis (2004).

TCP-Westwood (Casetti et al., 2002) introduces a new modification to TCP's congestion window adjustment algorithm, which improves the performance of TCP-Reno in wired as well as wireless networks. The idea behind TCP-Westwood is that it continuously counts, at the sender side, the bandwidth utilisation during the data exchange, by monitoring the rate of the incoming acknowledgements. This estimation about the bandwidth is used in order to calculate the congestion window and the Slow-Start threshold after a congestion episode, after three dacks or timeout expiration. In contrast to Reno that, after a packet loss, blindly decreases the congestion window, TCP-Westwood tries to pick a value for the congestion window and the Slow-Start threshold consistent to the bandwidth utilisation at the time when congestion occurs. This mechanism is called *faster recovery*.

Freeze-TCP (Goff et al., 2000) makes the receiver responsible for detecting an impending handoff event based on the power of the incoming signals when concerning a wireless link. In such a situation, the receiver advertises a

window equal to zero, and in this way, it forces the sender to enter the *Zero Window Probe* (*ZWP*), during which the sender does not transmit data but only the TCP header. In case of a lost ZWP, the sender does not reduce the congestion window, and hence, Freeze-TCP does not require any modification at the sender or the base station.

TCP-Real (Tsaoussidis and Zhang, 2002) uses wave patterns: a wave consists of a number of fixed-sized data segments sent back-to-back. More precisely, the receiver monitors the rate by which the waves move into/go around the network. When high contention is present, this rate will fluctuate a lot, since the circulation of a packet or a wave depends highly on its position inside the router's buffer. Opposing to that, when a wireless error occurs, the circulation rate of the wave will not be affected.

## 4 Implementation

### *4.1 The core probing mechanism*

A probe cycle uses two probing segments (PROBE1 and PROBE2) and their corresponding acknowledgments (PR1ACK and PR2ACK), implemented as option extensions to the TCP header. As noted earlier, the segments carry no payload. The option header extension consists of three fields: (i) *type*, in order to distinguish between the four probe segments (this is effectively the *option code field*); (ii) (*options*) *length* and (iii) *id number*, used to identify the exchange of probe segments.

The sender initiates a probe cycle by transmitting a PROBE1 segment, to which the receiver immediately responds with a PR1ACK, upon receipt of which the sender transmits a PROBE2. The receiver acknowledges this second probing with a PR2ACK and returns to the ESTAB state (Figure 4.2). The sender makes an RTT measurement, based on the time delay between sending the PROBE1 and receiving the PR1ACK, and another based on the exchange of PROBE2 and PR2ACK.

The sender makes use of two timers during the probing cycle. The first is a *probe timer*, used to determine if a PROBE1 or its corresponding PR1ACK segment is missing, and the same again for the PROBE2/PR2ACK segments. The second is a *measurement timer*, used to measure each of the two RTTs from the probe cycles, in turn. The probe timer is set to the estimated RTT value current at the time the probe cycle is triggered.

The value in the option extension id number identifies the full exchange of PROBE1, PR1ACK, PROBE2 and RP2ACK segments rather than individual segment within that exchange. Thus, in the event that the PROBE1 or its PR1ACK is lost (i.e., the probe timer expires), the sender reinitialises the probe and measurement timers and retransmits PROBE1 with a new id number. Similarly, if a PROBE2 or its PR2ACK is lost, the protocol reinitiates the exchange of probe segments from the beginning by retransmitting PROBE1 with a new id number. A PR1ACK carries the same id number as the corresponding PROBE1 that it is acknowledging; this is also the id number used by the subsequent PROBE2 and PR2ACK segments. The receiver moves to the ESTAB state after sending the PR2ACK, which should terminate the probe cycle. In this state, and should the PR2ACK be lost, the receiver would receive–instead of data segments–a retransmitted PROBE1 that is reinitiating the exchange of probe segments since the sender's probe timer, in the meantime, will have expired.

### *4.2 Contention estimation in TCP-probing*

As noted before, a new feature is added to TCP-Probing, namely the vegas_predictor. The calculation of this predictor takes place after each RTT measurement. If this predictor shows high contention, the probe cycles are skipped and TCP-Probing behaves like the traditional TCP-Reno. In such a case, the protocol's performance equals that of TCP-Reno in a wired network, something that did not happen in the previous versions of the protocol (Tsaoussidis and Badr, 2000; Tsaoussidis and Lahanas, 2003). The vegas_predictor is calculated as follows:

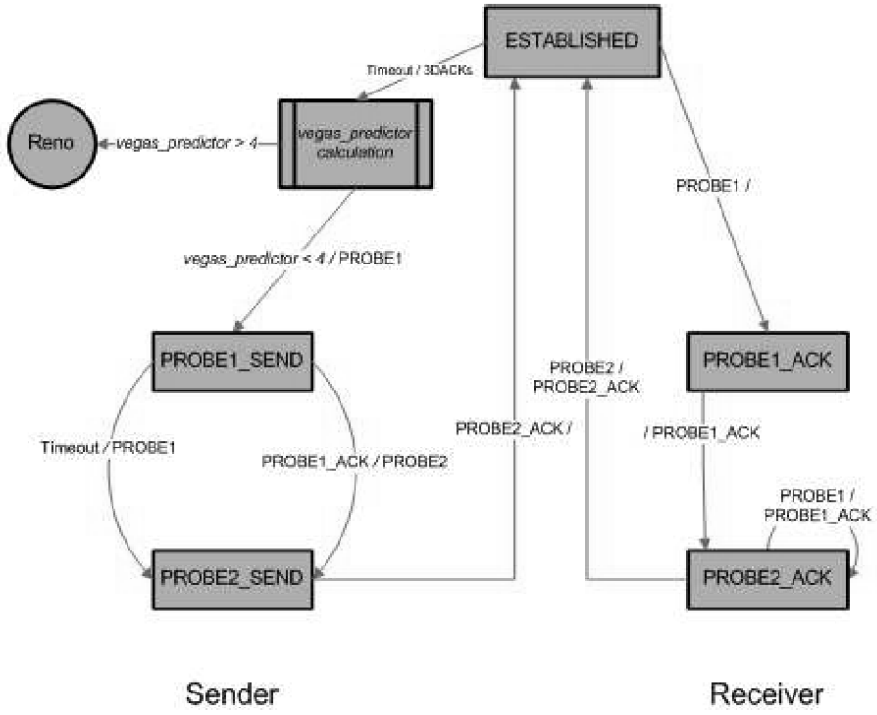
```
vegas_expected_throughput =
cwnd_ / best_rtt_
vegas_actual_throughput =
cwnd_ / last_rtt_
vegas_difference =
vegas_expected_throughput –
vegas_actual_throughput
```

The above equations are interpreted as follows. Similar to the TCP-Vegas algorithm, introduced in Brakmo et al. (1994) and Brakmo and Peterson (1995), we define the connection's *best_rtt* (called *Base_rtt* in Vegas algorithm) to be the smallest RTT measured during the connection. To be more precise, the *best_rtt* is usually the first RTT measured in the connection establishment phase, when the network is not congested. In this way (using *vegas_expected_throughput*), the Vegas algorithm is trying to identify the available capacity of the network. Along the previous lines, if we divide the current congestion window with the last measured RTT (*last_rtt*), we get the actual capacity of the network at that point (*vegas_actual_throughput*). Last, we compute the difference between the Expected Throughput and the Actual Throughput, again along the lines of Vegas algorithm (Brakmo et al., 1994; Brakmo and Peterson, 1995). The *vegas_difference* is always either zero or a positive number, since, if *vegas_actual_throughput_* < *vegas_expected_throughput*, we have just measured a smaller RTT than the *best_rtt*, and we need to update the *best_rtt* value. At that point, the Vegas algorithm introduces two additional thresholds $a$ and $b$, where $a < b$. Using these thresholds, Vegas decides about its transmission policy. If the difference between the Expected Throughput and the Actual Throughput is smaller than $a$, then no congestion is indicated, and Vegas increases its transmission rate in order to utilise all the available throughput capacity. Otherwise, if the above difference is greater than $b$, then this is interpreted as a congestion indicator and the transmission rate decreases. Generally, Vegas is trying to oscillate between

*a* and *b*. According to Brakmo et al. (1994), *a* and *b* are adjusted to 2 and 4, respectively.

In our algorithm, we need a congestion indicator, in order not to loose two RTTs by probing the network, since the probing cycles will also indicate that congestion is present. For this purpose, we adopt the Vegas algorithm and its upper threshold, *b*. We rename *b* as *vegas_difference* and base our decision about entering the probing cycles or not on that. That is, if *vegas_difference* is greater than 4, then the probe cycles are skipped (see Figure 1). Otherwise, if a packet goes lost, the probing mechanism is triggered.

**Figure 1** TCP-probing state diagram




### *4.3 How TCP-probing detects the network conditions*

A primary part of the probing mechanism, which directly impacts the protocol's performance, is the one responsible for the recovery strategy. The recovery strategy will be triggered after the end of the probe cycle. In case of a packet loss due to congestion, the protocol backs off as in Tahoe or in Reno depending on the level of contention (the higher the level of contention, the more conservative it will recover). If the loss was caused by a random transient error, the available bandwidth of the channel will, normally, no longer be affected. In such a situation, there is no need to back off and Immediate Recovery is applied. A handoff will cause the loss of the probe segments, and hence the probe cycle will not terminate until the communication channel is 'up' again. The fact that only probe segments (40 bytes) are lost during the handoff makes the protocol energy efficient compared to Reno and New Reno, since the overhead that the protocol imposes on the network is significantly decreased. Depending on the measurements of the probe cycles, the appropriate recovery will be applied.

Owing to its Immediate Recovery, TCP-Probing exploits the available bandwidth of a wireless channel better than TCP-Reno/TCP-NewReno would. TCP-Reno/TCP-NewReno would back off, considering that congestion is present.

This means that both the congestion window and the Slow-Start threshold are adjusted downwards. In this way, many RTTs are needed in order to reach the previous full window size.

It is possible to experience a random drop during a phase of moderated congestion. At this point, TCP-Probing calculates the *vegas_predictor*. If the predictor shows that there is high level of contention, the probe cycles are skipped and the protocol acts like TCP-Reno. Otherwise, the probing cycle process is triggered. Upon completion of the probing cycles, TCP-Probing takes advantage of the measurements gathered during the probe cycles and responds like TCP-Reno or TCP-Tahoe (the decision between TCP-Reno and TCP-Tahoe is explained later). More specifically, immediately after the probing process, the protocol calculates a threshold that identifies the Current Probe RTT (cp_rtt). The threshold used here is built dynamically and relies on the recent probe RTT sample. We compare the cp_rtt with the best and worst RTT measurements 'seen'/gathered during the history of the probe cycles (best_probe_rtt_, worst_probe_rtt, that is,

the smallest and the largest probe RTTs during the communication, respectively). The more the distance from the best_probe_rtt grows, the more conservatively we will recover.

More precisely, the protocol stores two variables during the communication, namely the *best_probe_rtt* and the *worst_probe_rtt*. The *best_probe_rtt* defines the best RTT measured during the probing cycles and the *worst_probe_rtt* defines the worst RTT measured during the probing cycles. By calculating the semi-total of the above two values, we get a second validation (the first comes out of the *vegas_predictor*) about the current network conditions.

```
average_probe_rtt = (best_probe_rtt +
worst_probe_rtt)/2
```

At this point, the protocol calculates the Current Probe RTT (cp_rtt)

```
cp_rtt = ALFA × cp_rtt + BETA × sample_
probe_rtt
```

where, ALFA = 0.25 and BETA = 0.75. The *sample_ probe_rtt* is either the first probe *RTT p_rtt1*, which is the first probe measurement during the last probing process or the second probe *RTT p_rtt2*, which is the second probe measurement during the last probing process. The Current Probe RTT also depends on the previous Current Probe RTT, taking in this way into consideration a brief history of the network conditions.

By comparing the Current Probe RTT with the Average Probe RTT, we know if the cp_rtt is above or below the average_probe_rtt. In both cases (above or below), we compare the probe RTTs with the smallest (best) RTT during the communication. After the above comparisons, the protocol has all the required information regarding the network conditions, in order to recover accordingly.

More precisely, if the Current Probe RTT is below the Average Probe RTT, then we conclude that the network is in a good state at the moment and there is no reason to back off adjusting the congestion window to half of its current value (as Reno would). Following, we compare both RTT measurements of the last probe cycles (p_rttl and p_rtt2) with the best probe RTT measured during the communication. If both samples are smaller than the best probe RTT ever measured, then the network is definitely not congested and the protocol recovers using Immediate Recovery. Otherwise, if one or both of the probe measurements are bigger than the best probe RTT, then the Slow-Start threshold is set to 3/4 of its previous value. At that point, we explicitly state that this recovery (3/4_(IR)) is a measurement-based defined value and is a subject of further investigation.

If nothing of the above happens, which means that the Current Probe RTT is above the Average Probe RTT, then we recover more conservatively. Again, we compare both of the last probes measured with the best probe RTT taken during the communication. If both probe measurements are smaller than the best probe RTT, then we recover just as Reno would. Otherwise, we conclude that there is high contention in the network, so we back off to the Slow-Start phase.

```
if (cp_rtt_ < average_probe_rtt)
/*we are below the average: more aggressive behavior is needed*/
  if (both_sample_probe_rtts < best_RTT)
       FULL_IR
  else
   3/4_IR
/*we set ssthresh=3/4*ssthresh & cwnd=1*/
  end if
 end if

 if (cp_rtt_ > average_probe_rtt)
 /*we are above the average: more conservative behavior is needed*/
  if (both_sample_probe_rtts < best_RTT)
        RENO_RECOVERY
  else
        SLOW_ START_RECOVERY
 end if
end if
```

Other issues need to be considered for the implementation of the probing mechanism. Although the congestion predictor (*vegas_predictor*) seems to work well, it may need to be replaced by a more sophisticated mechanism in order to make a better estimation of network conditions. Another issue that calls for further investigation is the number of the probe segments sent. It may be better for the probing mechanism to monitor the network by one probe measurement only, avoiding the wastage of a second RTT. Such a change will also affect the probing decision, since there will be no *sample_probe_rtt* but only 'one_probe_rtt'. Finally, the version of Immediate Recovery that reduces the Slow-Start threshold to 3/4 of its previous value may need to become more adjustable, in order to exploit the available bandwidth in a more sophisticated way.

## 5 Experimental methodology

### 5.1 Testing plan

We have implemented our testing plan using ns-2. Two network topologies are used in our experiments. The first is the typical single-bottleneck *dumbbell*, as shown in Figure 2. The second is a more complex double-bottleneck network topology, that we call *split-dumbbell*. In this second topology, three routers take place; half of the receivers are connected to the second router, while the rest of them are connected to the third router, as shown in Figure 3. The results that came out of these simulations are very promising and call for further research. The link's capacity (*bw bottleneck*) is 100 Mbps. The *bwsrc* and the *bw_dst* are either 1 Mbps or 10 Mbps, while the delay injected into the links is set to 10 ms (*delay_src = delay_bottleneck = delay_dst = 10 ms*). We used equal number of source and sink nodes. We simulated a heterogeneous (wired and wireless) network with ns-2 error models that were inserted into the access links at the sink nodes. The Bernoulli model was used to simulate link-level errors with configurable bit error rate (BER). The number of flows occasionally changes, according to the needs of the several different scenarios. The simulation time was fixed at 60 seconds, a time period that seemed appropriate to allow all protocols to demonstrate their potential.

**Figure 2** The *dumbbell* network topology

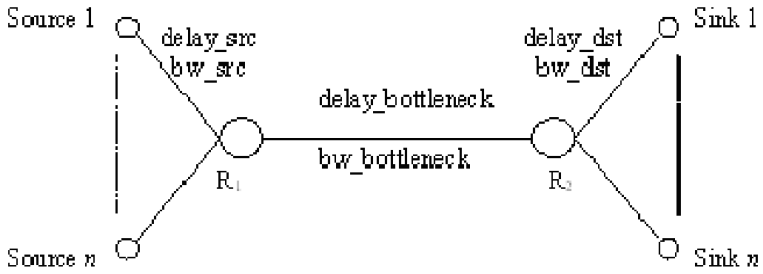
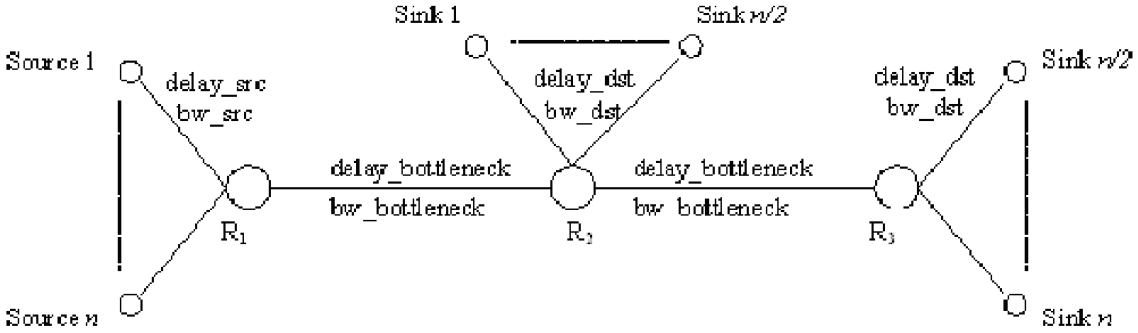



**Figure 3** The *split-dumbbell* network topology




In most of the scenarios, ftp flows are entering the system within the first two seconds. All flows are fixed during the rest 58 seconds. In order to evaluate how efficiently and fairly the protocols can exploit the available bandwidth, we also simulated some additional scenarios with graduated contention decrease and contention increase.

### 5.2 Performance metrics

Our evaluation plan calls for common as well as non-traditional metrics. We used traditional metrics for protocol efficiency and fairness.

The system Goodput is used to measure the overall system efficiency in bandwidth utilisation. The system Goodput is defined as:

```
Goodput = Original_Data/Connection_time
```

where, *Original_Data* is the number of bytes delivered to the high-level protocol at the receiver (i.e., excluding retransmitted packets and overhead) and *Connection_time* is the amount of time required for the data delivery. Fairness is measured by the Fairness Index, derived from the formula given in Chiu and Jain (1989) and defined as:

$$\text{Fairness} = \frac{\sum(\text{Throughput}_i)^2}{n\sum(\text{Throughput}_i^2)}$$

where, Throughput is the Throughput of the *i*th flow and *n* is the number of participating flows.

Energy expenditure or energy efficiency is a rather important factor that has a major impact on wireless connected, battery-powered devices. However, apart from the overhead metric, there is no other metric in the literature that monitors the behaviour of a protocol towards energy expenditure. Departing from that point and in order to capture the amount of *extra* energy expended, we introduce a new metric that was first presented in Mamatas and

Tsaoussidis (2004). We call this new metric, *Extra Energy Expenditure* (*3E*). Three variables take place in this new metric. These are $\text{Throughput}_{max}$, Throughput and Goodput. The idea behind $\text{Throughput}_{max}$ is that it captures the best possible data transmission that can be achieved under the given network conditions. The other two variables are the Throughput and the Goodput metrics that monitor the protocol's performance. We define $\text{Throughput}_{max}$ as follows: first of all, we define the network conditions for each different scenario (wired or wireless, handoff events, bit error rate, link capacity, delay, etc.). We simulate a large number of flows that run a CBR (Constant Bit Rate) application under UDP (User Datagram Protocol). In this way, we virtually form a very aggressive protocol that transmits the greatest possible amount of data for each given scenario or, in other words, it completely exploits the available bandwidth. At this point, we are not interested in successful data transmission. Hence, we use the Throughput metric, instead of the original data that reach the application level (Goodput). The 3E metric is given by the following formula:

$$\text{EEE} = a\frac{\text{Throughput} - \text{Goodput}}{\text{Throughput}_{max}} + b\frac{\text{Throughput}_{max} - \text{Throughput}}{\text{Throughput}_{max}}$$

where, $a = 1$ and $b = 0.3$.

We have chosen the above values for $a$ and $b$, in order for the metric (EEE) to meet the needs of some prescribed network and device conditions. That is, when Goodput = Throughput = $\text{Throughput}_{max}$, a situation where all the expended energy has been invested into efficient transmissions, the extra energy expenditure is zero. On the opposite direction, when the protocol is trying to transmit data but all data get lost (e.g., handoff event), the *extra energy expenditure* has to reach a peak value. For example, when $\text{Throughput}_{max} = 100$, Throughput = 100 and Goodput = 0, the extra expenditure due to unsuccessful retransmission grows to a maximum value (1). Finally, there is one more scenario, the conditions of which have to be taken into consideration, that is, when the host is connected but the protocol is neither transmitting nor receiving data, probably waiting for data segments or acknowledgements to arrive. Under these conditions, when Goodput approaches Throughput, which approaches 0, the extra expenditure is only due to time wasted, probably waiting in an idle state.. We assume that the extra expenditure at this stage is 0.3 (the first term is 0). In summary, parameter $a$ must be linked with the device transmission power: $a = P_{Tx}(W)$, while $b$ must be linked with the device idle power: $b = Pidie(W)$. According to Jones et al. (2001), the device idle power is 70% of the transmission power. We settled on $b = 0.3$ based on that observation, since $a = 1$.

It is clear that in all cases, $\text{Throughput}_{max}$ > Throughput > Goodput. Extra Energy Expenditure (3E) takes into account the difference of achieved Throughput from maximum Throughput ($\text{Throughput}_{max}$) for the given channel conditions, as well as the difference of Goodput from Throughput, attempting to locate the Goodput as a point within a line that starts from 0 and ends at $\text{Throughput}_{max}$. We will give some examples in order to get a better aspect of this metric.

We set $\text{Throughput}_{max}$ at a fixed maximum value: $T_{max} = 100$. Suppose we have two flows, where $T_1$, $T_2$, and $G_1$, $G_2$ are the Throughput and Goodput values for the two flows, respectively. If $T_{max} = T_1 = T_2 = 100$, $G_1 = 80$ and $G_2 = 60$, we can easily understand that the second flow has spent more energy on its effort to transmit data. Hence, $\text{EEE}_1 < \text{EEE}_2$. However, here the difference between Throughput and Goodput is different for the two flows, and the extra energy expenditure of the second flow is clear. Suppose a situation where $T_1 = 80$, $G_1 = 60$, $T_2 = 60$ and $G_2 = 40$. Here the difference between the Throughput and Goodput of the two flows is equal ($T_1 - G_1 = T_2 - G_2$), making it more difficult to understand which of the two flows has spent more energy. However, if we put this difference ($T_1 - G_1 = T_2 - G_2$) over $T_1$ and $T_2$, respectively, we conclude that $((T_1 - G_1)/T_1) < ((T_2 - G_2)/T_2)$ or $1/4 < 1/3$. This means that the first flow spent less transmission effort (lower Throughput) than the second, but it still imposes the same amount of overhead to the network. The EEE metric takes into consideration the difference between $\text{Throughput}_{max}$ and Throughput too, which in this case is $T_{max} - T_1 < T_{max} - T_2$, and so it concludes that $\text{EEE}_1 < \text{EEE}_2$. Finally, we will give another example, where $T_{max} - T_1 > T_{max} - T_2$ and $T_1 - G_1 < T_2 - G_2$. Suppose $T_1 = 40$, $G_1 = 39$, $T_2 = 60$ and $G_2 = 50$. In this case, the EEE metric concludes that $\text{EEE}_1 < \text{EEE}_2$. The important point here is that in the second example, the first flow transmits a greater amount of data and spends less energy than the second flow, while in the third example, although the second flow transmits more data than the first flow, it still has a greater energy expenditure than the first flow.

Another important factor that determines a protocol's performance, is the way in which a protocol exploits the available resources of the network. It is very common, especially in heterogeneous wired/wireless networks, a situation where bandwidth becomes available rapidly because of the presence of a handoff event that affects some–and not all – of the competing flows that share the same channel. In such a situation, and for as long as the handoff lasts, the remaining flows have opportunities to enlarge their windows and transmit a greater amount of data without any impact on their fairness, being in that way more efficient. This is the contention decrease scenario. Along these lines, we need to introduce another metric as well, in order to capture the level of *Unexploited Available Resources* (*UAR*). The UAR metric is given by the following formula:

$$\text{UAR} = 1 - \left[ a\frac{\text{Throughput}}{\text{Throughput}_{max}} + b\frac{\text{Goodput}}{\text{Throughput}} \right]$$

where, $a = 0.5$ and $b = 0.5$.

The UAR index ranges from 0 to 1 (like the EEE metric), expressing also a negative performance aspect. We will give two examples in order to justify the fixed values of *a* and *b*. We suppose again that $\text{Throughput}_{max}$ equals 100 in all cases. In contrast to the EEE metric, UAR should give a different result when Throughput = 1 and Goodput = 0. In this situation, the protocol does not exploit the available bandwidth at all and hence, this should give it a major penalty (UAR = 1). In the opposite situation, where the protocol fully exploits the available resources (bandwidth) of the network, which means that Throughput = Goodput = 100, the UAR metric equals 0. The UAR metric, however, should be taken into consideration in conjunction with the Fairness Index presented above. This is because full exploitation of the available bandwidth does not always conclude to a fair behaviour of the protocol, as it happens with TCP-Vegas (Brakmo et al., 1994; Brakmo and Peterson, 1995), for example.

We will present some examples in order to understand the potential of this new metric. Just like before, we suppose that $T_{max} = 100$ and that we have two flows, where $T_1$, $T_2$ and $G_1$, $G_2$ are the Throughput and Goodput values for the two flows, respectively. If $T_{max} = T_1 = T_2 = 100$, $G_1 = 80$ and $G_2 = 60$, we can easily understand that the first flow made better use of the available network resources. In this situation, it is clear that $UAR_1 < UAR_2$. In a different scenario, suppose that $T_1 = G_1 = 80$ and $T_2 = G_2 = 60$. Here, both flows performed fairly well energy-efficiency-wise and neither of them seems to have spent effort for retransmission of lost data. However, it is clear that the second flow did not exploit the available bandwidth as effectively as the first flow. Moreover, even the first flow might have wasted some opportunities for data transmission, since the network conditions seemed perfect. Again $UAR_1 < UAR_2$. The UAR metric takes into consideration not only the Goodput as for Throughput but also the Throughput as for $\text{Throughput}_{max}$, being in this way able to capture both the overhead (Goodput/Throughput) and the transmission effort ($\text{Throughput}/\text{Throughput}_{max}$) of the protocol. This can be made clear by the following scenario, where $T_1 = 40$, $G_1 = 39$, $T_2 = 60$ and $G_2 = 50$. Here, although the second flow spent more energy in order to transmit the data segments (as it was shown by the EEE metric) and hence has a greater overhead than the first flow, it exploited the available bandwidth in a better manner than the first flow. As a result, $UAR_1 > UAR_2$. These two new metrics seem to graph, in a rather good and realistic manner, the protocol's behaviour from a point of view that was never introduced in the related literature. The metrics seem to be very helpful especially for protocols that focus on heterogeneous wired/wireless networks.

At this point, we explicitly state the difference between the two metrics. The Extra Energy Expenditure (EEE) metric answers the question: 'How much energy did the protocol spend while it should not?', while the Unexploited Available Resources (UAR) metric answers the question: 'How much bandwidth did the protocol leave unexploited while it should not?'. In other words, the EEE metric captures the protocol's behaviour in terms of energy expenditure, while the UAR metric captures the protocol's behaviour in terms of bandwidth utilisation. As it is clear from the above analysis, the two metrics behave differently. For example, if the protocol does not transmit even a single data packet, which means that Throughput = Goodput = 0, the EEE metric will equal the minimum value of 0.3, while the UAR metric will give a major penalty (UAR = 1).

Two versions of TCP took place in the simulations: TCP-NewReno and TCP-Probing (Tsaoussidis and Badr, 2000; Tsaoussidis and Lahanas, 2003). Many experiments with different characteristics have been carried out, in order to understand each protocol's behaviour. We present four simple scenarios whose characteristics show the efficiency of our adaptive recovery strategy. Afterwards, we present another four more complicated scenarios in order to monitor the protocols' behaviour in more realistic environments. Each version of the protocol was tested by itself, separately from the others, so that the error control mechanism can demonstrate its capability, without being influenced by the presence of other type of flows in each channel.

## 6 Results and discussion

### *6.1 A full wired scenario*

In the first scenario, we set the bw_dst to 10 Mbps, in order for the flows to have enough fair-share to expand their windows. The topology is the *dumbbell* network topology, and the queueing algorithm is the simple Drop Tail. Our purpose here is to demonstrate the effectiveness of TCP-Probing in a wired environment. In fact, in some cases, TCP-Probing slightly outperforms New Reno. This is mostly happening because after a packet is lost owing to buffer overflow, and should vegas_predictor indicate no congestion (since there is enough throughput capacity), TCP-Probing immediately becomes aware of this capacity and recovers with Immediate Recovery. The Goodput and UAR charts below (Figures 4 and 5) validate our previous assumptions. In this scenario, there is no difference in the energy expenditure of the two protocols, and the Fairness Index approaches 1.

**Figure 4** Goodput on wired

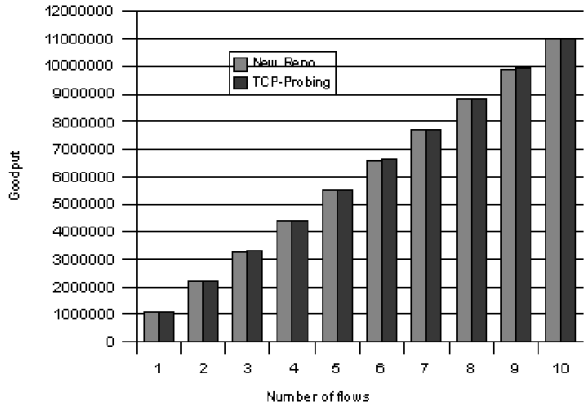
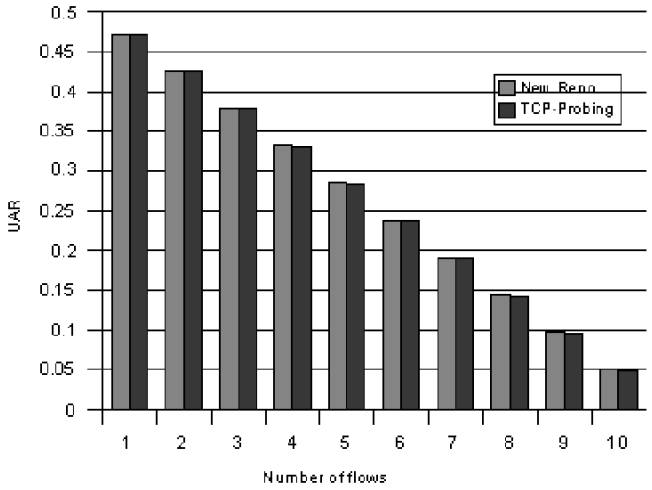


**Figure 5** UAR on wired




## *6.2 A wired scenario with contention decrease*

In this scenario, we present a simple wired environment in order to understand the basic behaviour of the protocols in conjunction with contention decrease. The topology used is the *split dumbbell* network topology, where bw_src = 10 Mbps and bw_dst = 1 Mbps. We measure Unexploited Available Resources (UAR) Index and Goodput for a range of flows from 10 to 100. All flows enter the system within the first two seconds. For the rest 58 seconds, we have a graduated decrease, starting from 10 flows and repeating the experiment for 20, 30 up to 100 flows. At each stage, we reduce the number of flows to half every Decrease_Step seconds, where Decrease_Step is the step needed in order for the last flow to exit at the 60th second. Here the bw_dst is 1 Mbps, and hence there is a high level of contention. We used a large number of flows, in order for the flows to 'step out' in groups, and as a result, for every Decrease_Step, a large amount of the bottleneck's throughput capacity is freed up.

As can be seen from Figures 6 and 7, the adaptive recovery strategy of TCP-Probing, which inspects the network conditions (by the probe cycles), is able to detect the available bandwidth and immediately exploit it. On the contrary, TCP-NewReno reduces its window on every packet loss, and thus not only is it unable to exploit the extra available bandwidth but also leaves some of the existing bandwidth unexploited.

From Figure 8, we can see that New Reno does not spend much extra energy compared to Probing in its effort to transmit data. Instead, it appears to be unfair as can be seen from Figure 9. The performance degradation of New Reno is clear in this scenario.

**Figure 6** Goodput on wired/congestion decrease

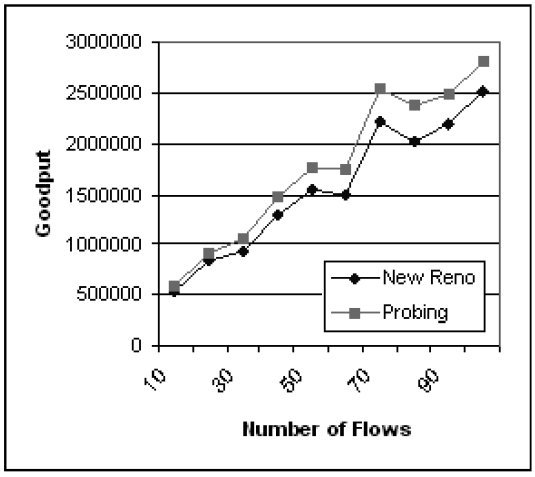



**Figure 7** UAR on wired/congestion decrease

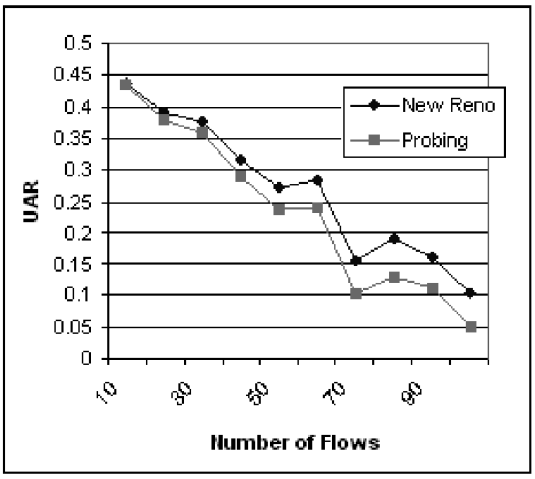



**Figure 8** EEE on wired/congestion decrease

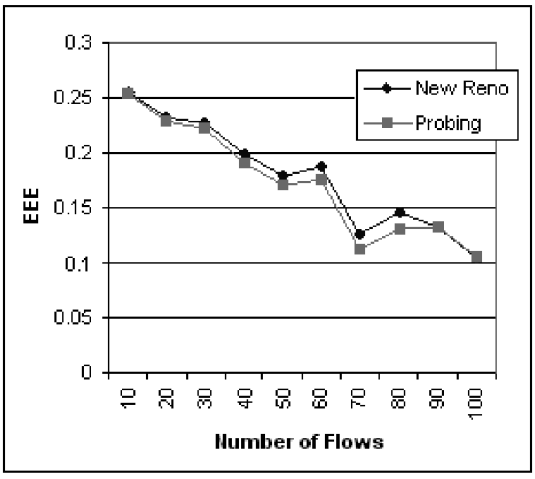
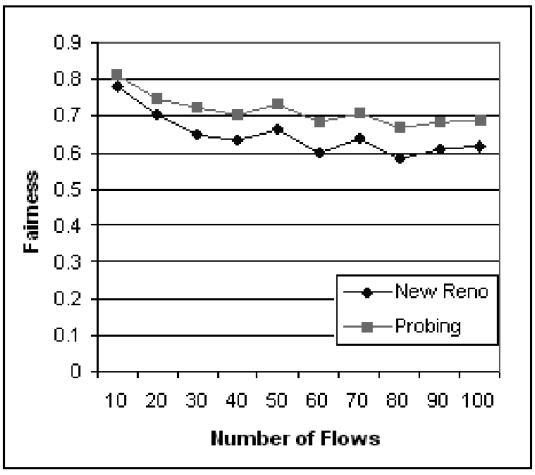


**Figure 9** Fairness on wired/congestion decrease




### *6.3 Wireless scenario with handoffs*

Handoff events are very common handoff are in events in mobile networks. This scenario intends to present some results that monitor the behaviour of the protocols in a situation of frequent handoffs. The bw_dst is set at 1 Mbps, in the *split dumbbell* network topology. The queueing algorithm is again the Drop Tail, and the protocols experience a situation where ten handoff events occur randomly and last for 0.5 seconds each. In addition, there is a small bit error rate (0.001%), since this is considered to be a wireless link. Apart from the above, in this scenario, the competing flows do not enter the communication channel 'simultaneously', but instead some time interval differentiates their entering times. The first flow starts the data transmission when the simulation begins. The last flow in each scenario enters the communication channel at the 20th second. That is, the flows coexist in the channel for 40 seconds, since the total simulation time is 60 seconds.

In Figures 10 and 11, TCP-Probing clearly outperforms TCP-NewReno in terms of Goodput and Fairness. This is easily explained if we take into consideration that every time a handoff event occurs, and hence data is lost, TCP-NewReno considers this as an indication of congestion. As a result, it backs off, by reducing its window and extending its timeout period, but still persists on transmitting data segments. These segments are dropped, until the communication link recovers. After the end of the handoff event, although a lot of bandwidth is available, TCP-NewReno transmits only 1 KB and enters Slow-Start phase.

On the contrary, TCP-Probing suspends data transmission for as long as the handoff is present (in fact, it losses the probe segments that are 40 bytes each). After the end of the handoff period, TCP-Probing detects the network conditions (by completing successfully two probe cycles) and gets immediately aware of the available bandwidth, which it immediately exploits. Hence, it acts with respect to energy consumption, while it is aggressive only when the network conditions so allow.

**Figure 10** Goodput on wireless/handoffs

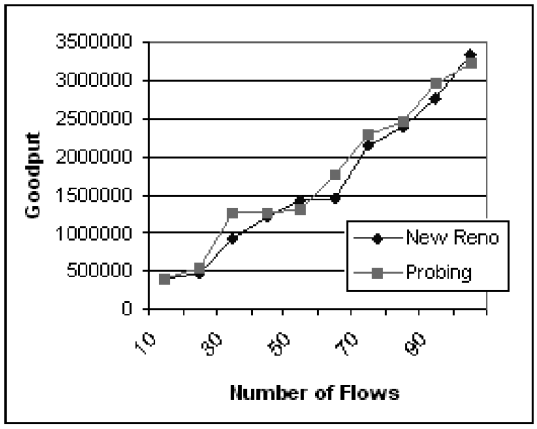
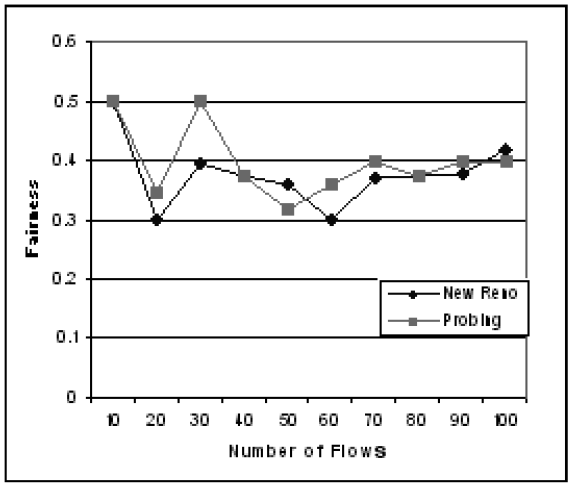



**Figure 11** Fairness on wireless/handoffs




### *6.4 A wireless scenario with errors*

In this scenario, we simulated a 0.2% packet error rate environment, with a variable number of flows from 10 to 100 where bw_dst is set at 1 Mbps, which means that congestion is often indicated. The topology used is the *dumbbell* network topology, and the queuing algorithm is the Random Early Drop (RED) (Floyd and Jacobson, 1993). As noted before, in the presence of random transient errors, there is no reason to extend the timeout period, while the sender's window might have to be reduced, especially in high error rate environments.

As can be seen from Figures 12–14, the adaptive recovery strategy of TCP-Probing, where no timeout extension takes place in the Immediate Recovery phase, performs better than TCP-NewReno. The impact on energy expenditure is also clear (see Figure 13). In this scenario, TCP-Probing recovers with Immediate Recovery after a packet loss, inducing only a small shrinkage of the congestion window. Although this seems to be an aggressive recovery, the fairness diagram (Figure 14) shows that Probing effectively responds with respect to the rest of the competing flows.

**Figure 12** Goodput on wireless/errors

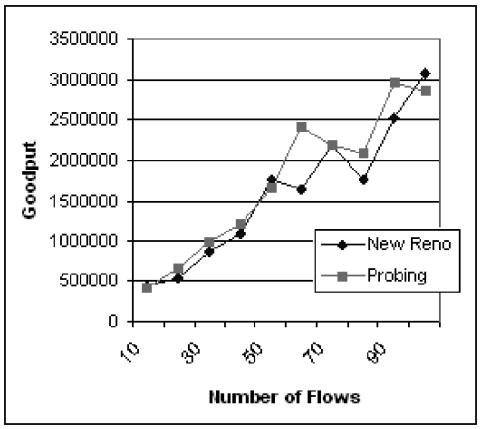



**Figure 13** EEE on wireless/errors

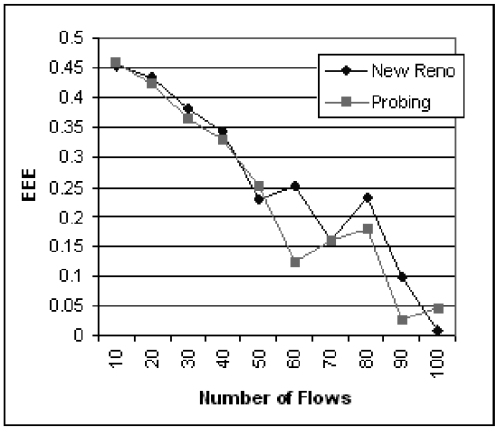
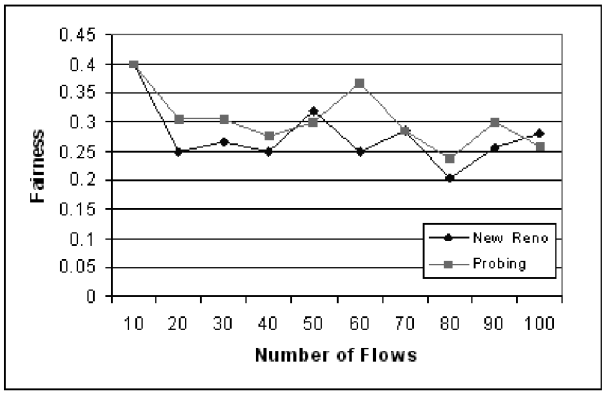



**Figure 14** Fairness on wireless/errors




## *6.5 Wireless – mobile environments*

In a mobile environment, it is very common that all the above (high level of contention, contention increase/decrease, handoffs and random transient errors) happen at the same time. The purpose of the scenarios presented so far was to start from the beginning and monitor the unique characteristics of each network condition, in order to find a recovery strategy that deals with each of them, apart from the others. In order to monitor the protocols' behaviour in a heterogeneous, mobile environment, we simulated several scenarios where all kinds of wireless errors coexist. Here, we will present four different and more complicated scenarios to show that a probing mechanism suites better to heterogeneous networks.

In the first, more complicated scenario, we simulate a wireless environment, where five handoff events occur at random times and each one lasts for 1 second. Furthermore, there is a small bit error rate (0.001%), and in addition, during the simulation time, some of the competing flows finish their work and leave the communication channel (*Contention Decrease*). We use the *dumbbell* network topology with bw_dst = 1 Mbps, and the routers use the Drop Tail queuing algorithm.

From the Goodput diagram (Figure 15), we see the improved performance of TCP-Probing compared to that of New Reno, even when there is a high level of contention. The duration of each handoff event is 1 second, during which New Reno continues its transmission effort. However, all the data segments get lost, as can be seen from the EEE diagram (Figure 16). On the contrary, TCP-Probing suspends data transmission for as long as the handoff is present, being in that way more energy efficient. The complicated conditions of the network seem to confuse the protocols (fairness diagram, Figure 17). Even in this situation, TCP-Probing appears to be fairer than New Reno.

In the next scenario, the simulated topology is the *split dumbbell*, where the bw_dst is 1 Mbps and the RED queue policy is implemented in the routers. During the 60 seconds of the simulation time, ten handoff events occur. Each one lasts for 0.5 seconds, and there is a bit error rate of 0.1%. Similar to the previous scenario, here we have a situation *of contention decrease.*

Figures 18 and 19 clearly show that TCP-Probing outperforms New Reno one more time, both in terms of Goodput and Fairness. The level of contention in this scenario is rather high, since the bw_dst = 1 Mbps. Under these conditions, the performance of New Reno was expected to be rather good compared to a more aggressive protocol, such as Probing. However, it seems that *vegas_predictor* gives an accurate estimation of the network load, forcing TCP-Probing to respond aggressively, only when the network conditions so allow.

**Figure 15** Goodput on wireless 1

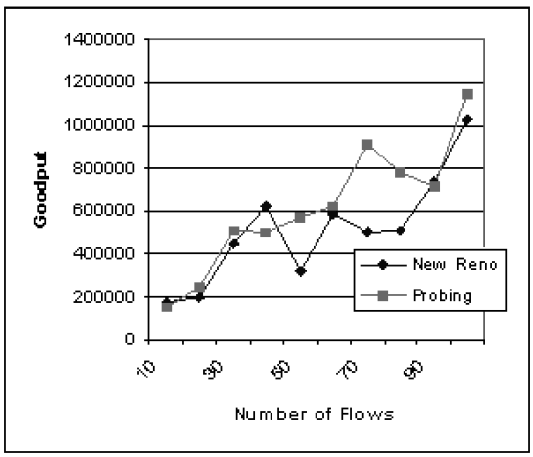


**Figure 16** EEE on wireless 1

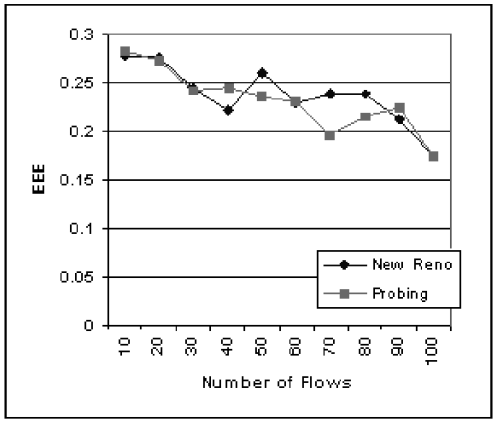



**Figure 17** Fairness on wireless 1

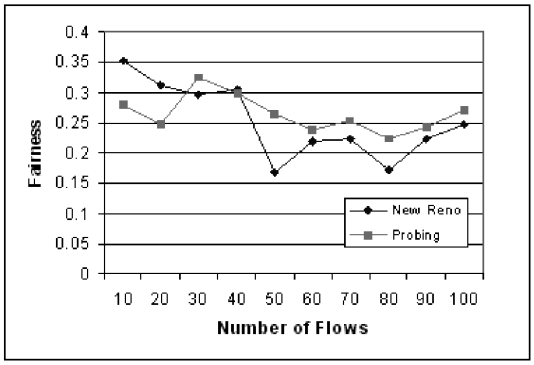



**Figure 18** Goodput on wireless 2

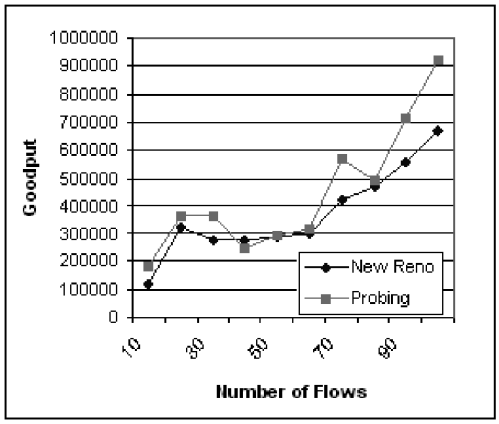
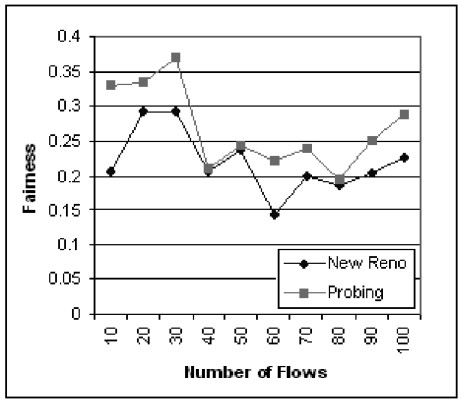



**Figure 19** Fairness on wireless 2




In the last two scenarios, we simulated a fixed number of flows that experience variable bit error rates ranging from 0.01% to 0.4%. We used the *split dumbbell* network topology, where bw_src = bw_dst = 1 Mbps. In the first scenario, there are 70 competing flows and the queue type used in the routers is RED. The flows experience three handoff events that take place at random time points, and each one of them lasts for 0.5 seconds.

As mentioned before, the capacities of the links in this scenario are bw_src = bw_dst = 1 Mbps and bwbottleneck = 100 Mbps, something that indicates a rather low level of contention. This means that data loss happens mostly owing to wireless random or burst errors and not owing to congestion or buffer overflow. Our purpose here is to show the inefficient recovery strategy of standard TCP over wireless links. This inefficiency causes TCP not only to perform poorly in such an environment, but also to consume a lot of energy. Figures 20 and 21 make it clear that standard TCP is inappropriate for battery-powered devices.

**Figure 20** Goodput on wireless 3

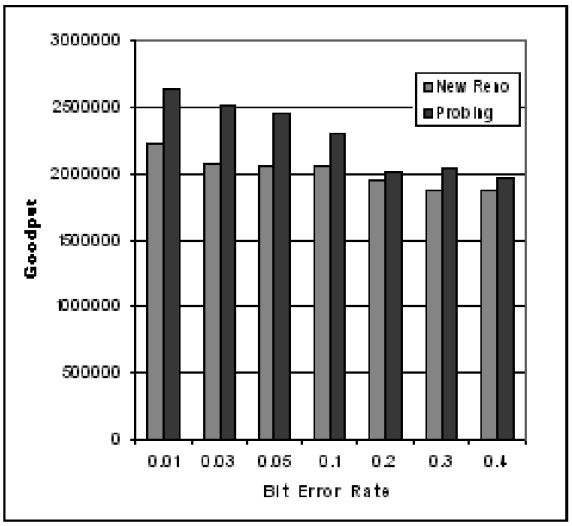
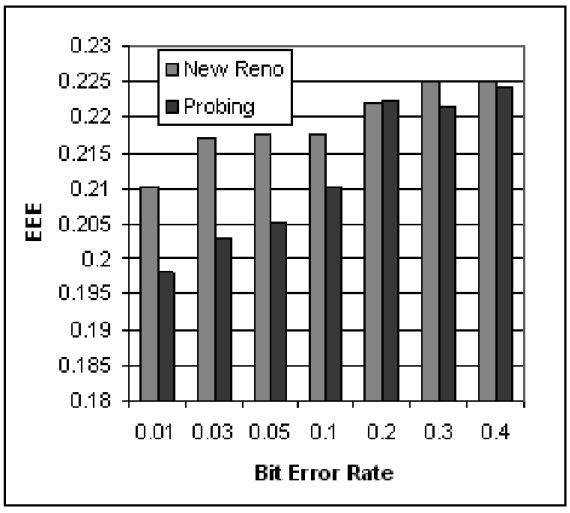



**Figure 21** EEE on wireless 3



The last scenario is similar to the previous one, but here there are no handoff events. Here, we simulate 60 flows in the *split dumbbell* topology (just like before), and the queue policy of the routers is the simple Drop Tail. The results presented in Figures 22 and 23 are also similar to that of the previous scenario. The purpose here is to simulate an environment where only random errors occur.

In the last two scenarios, TCP-Probing recovers using *Immediate Recovery*, since there is enough throughput capacity, being in this way more aggressive. From the figures of both the previous and the present scenario (Figures 20–23), we conclude that, the performance of Probing is high when the error rate is low and degrades as the error rate increases. This might be an indication, as stated in Mamatas and Tsaoussidis (2004) too, that in an environment with high error rate, the appropriate action is to be more conservative (New Reno). The last is validated from Figures 21 and 23 where TCP-Probing, a more aggressive protocol, does not consume less energy, nor does it exploit more bandwidth than TCP-NewReno, when the error rate is high.

**Figure 22** Goodput on wireless 4

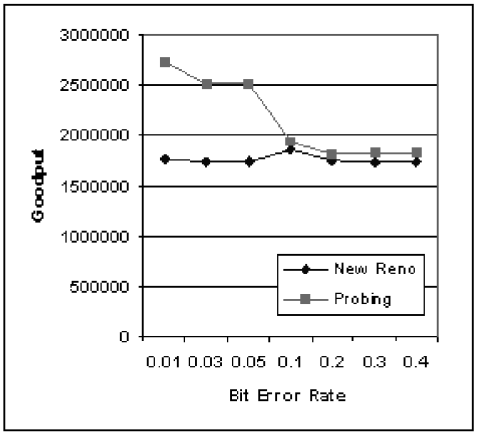
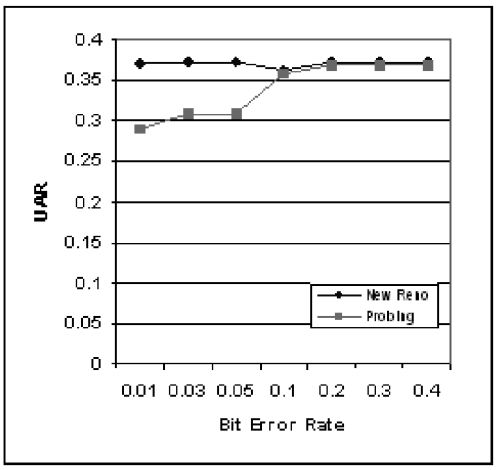



**Figure 23** UAR on wireless 4




## 7 Conclusions

We presented and analysed the most common, but still very important characteristics of heterogeneous wired/wireless networks and the way in which standard TCP handles data transmission under such conditions. We studied in depth the behaviour of an experimental protocol, TCP-Probing (Tsaoussidis and Badr, 2000; Tsaoussidis and Lahanas, 2003) that uses a clever technique in order to be aware of the network conditions and somehow classify the nature of the error (congestion or not).

We propose an additional mechanism that focuses on the continuous contention estimation of the network in order to forerun congestion losses. Hence, in case of congestion losses, the protocol backs off as standard TCP would. If no congestion is indicated, the level of contention is measured (using the probe cycles) and the appropriate recovery strategy is applied afterwards. In this way, using the *congestion* predictor and the *contention* or *wireless error* classifier, we can get a rather accurate picture of the network and respond accordingly.

Although we studied primarily the way the decoupled parts work together (without focusing on either the contention estimation or error recovery mechanism), our results are very promising. As mentioned before, the contention estimator (*vegas_predictor*) can be replaced by a more sophisticated mechanism. In addition, the decision followed after the completion of the probe cycles can become more adjustable according to the network conditions. The experimental protocol presented in this work focuses on heterogeneous wired/wireless networks that characterise today's inter-networking. In a future work, we plan to optimise those two parts, in order to cooperate in a more successful way and achieve better performance in a wider spectrum of computer communications.

## References

Allman, M., Paxson, V. and Stevens, W. (1999) *TCP Congestion Control, RFC2581*, April.

Attie, P.C., Lahanas, A. and Tsaoussidis, V. (2003) 'Beyond AIMD: explicit fair-share calculation', *Proceedings of ISCC2003*, June, Turkey, p.727.

Barman, D. and Matta, I. (2002) 'Effectiveness of loss labeling in improving TCP performance in wired/wireless networks', *Proceedings of the 10th IEEE International Conference on Network Protocols*, Paris, pp.2–11.

Biaz, S. and Vaidya, N. (1999) 'Discriminating congestion losses from wireless losses using inter-arrival times at the receiver', *Proceedings of the 1999 IEEE Symposium on Application – Specific Systems and Software Engineering and Technology*, Richardson, TX, USA, p.10.

Brakmo, L.S. and Peterson, L.L. (1995) 'TCP vegas: end to end congestion avoidance on a global internet', *IEEE Journal on Selected Areas in Communications*, Vol. 13, No. 8, pp.1465–1480.

Brakmo, L.S., O'Malley, S.W. and Peterson, L.L. (1994) *TCP Vegas: New Techniques for Congestion Detection and Avoidance, SIGCOMM*, pp.24–35.

Casetti, C., Gerla, M., Mascolo, S., Sansadidi, M.Y. and Wang, R. (2002) 'TCP westwood: end-to-end congestion control for wired/wireless networks', *Wireless Networks Journal*, Vol. 8, pp.467–479.

Chandra, D., Harris, R. and Shenoy, N. (2003) *Congestion and Corruption Loss Detection with Enhanced-TCP, ATNAC 2003*, Melbourne.

Chiu, D-M. and Jain, R. (1989) 'Analysis of the increase and decrease algorithms for congestion avoidance in computer networks', *Computer Networks and ISDN Systems*, Vol. 17, No. 1, pp.1–14.

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

Floyd, S. and Jacobson, V. (1993) 'Random early detection gateways for congestion avoidance', *IEEE/ACM Transactions on Networking*, Vol. 1, No. 4, August, pp.397–413.

Goff, T., Moronski, J., Phatak, D. and Gupta, V.V. (2000) 'Freeze-TCP: a true end-to-end enhancement mechanism for mobile environments', *Proceedings of IEEE INFOCOM 2000*, March, Israel, pp.1537–1545.

Jain, M. and Dovrolis, C. (2004) 'Ten fallacies and pitfalls in end-to-end available bandwidth estimation', *ACM Internet Measurements Conference* (*IMC*), Sicily Italy, October.

Jones, C.E., Sivalingam, K.M., Agrawal, P. and Chen, J-C. (2001) 'A survey of energy efficient transport protocols for wireless networks', *Journal of Wireless Networks*, Vol. 7, No. 4, pp.343–358.

Keshav, S. (2001) *Congestion Control in Computer Networks*, PhD Thesis, UC Berkeley, published as UCB TR 91/649, September, University of California at Berkeley.

Liu, J., Matta, I. and Crovella, M. (2003) 'End-to-end inference of loss nature in a hybrid wired/wireless environment', *Proceedings of WiOpt'03: Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks*, France.

Mamatas, L. and Tsaoussidis, V. (2004) 'Protocol behavior: more effort, more gains?', *The 15th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications*, Barcelona, Spain.

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

Psaras, I., Tsaoussidis, V. and Mamatas, L. (2005) 'CA-RTO: a contention adaptive retransmission timeout', *International Conference on Computer Communications and Networks, ICCCN 2005*, San Diego, California, USA.

Stevens, W. (1997) *TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms*, RFC 2001, January.

Tsaoussidis, V. and Badr, H. (2000) 'TCP-probing: towards an error control schema with energy and throughput performance gains', *The 8th IEEE Conference on Network Protocols, ICNP 2000*, Osaka, Japan, November.

Tsaoussidis, V. and Lahanas, A. (2003) 'Exploiting the adaptive properties of a probing device for TCP in heterogeneous networks', *The Journal of Computer Communications COMCOM, Elsevier Science*, Vol. 26, No. 2, February, pp.177–192.

Tsaoussidis, V. and Matta, I. (2002) 'Open issues on TCP for mobile computing', *Journal of Wireless Communications and Mobile Computing*, Wiley Academic Publishers, Vol. 2, No. 2, February, pp.477–497.

Tsaoussidis, V. and Zhang, C. (2002) 'TCP-real: receiver-oriented congestion control', *Computer Networks Journal* (*Elsevier*), Vol. 40, No. 4, November.

Xu, K., Tian, Y. and Ansari, N. (2004) 'TCP-jersey for wireless IP communications', *IEEE JSAC*, Vol. 22, No. 4, pp.747–756.