Shaping TCP Traffic in Heterogeneous Networks

I. Psarras, L. Mamatas and V.Tsaoussidis
Dept. of Electrical and Computer Engineering
Demokritos University of Thrace, Greece
{ipsaras, emamatas, vtsaousi}@ee.duth.gr

***Abstract* — In this paper, we depart from TCP-Probing [13, 14] and propose an experimental transport protocol that has energy and throughput performance gains in both wired and wireless environments. Our approach decouples error recovery from contention estimation and focuses on how these two mechanisms can (i) feed the decision process and (ii) implement the protocol strategy by shaping traffic, accordingly. We use a validation mechanism, which possibly uncovers previous wrong estimations. Our analysis matches well our simulation results, which are very promising.**

***Keywords* — TCP, Probing, Energy, Wireless**

I. 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 [1] 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 network's speed (i.e. high-speed) to the 'end-users' battery lifetime of mobile devices (i.e. in case of ad-hoc networks). Although a backbone can be also wireless or satellite, our present work focuses on wired backbone with wireless receiving-ends. Further experimentation can easily be used for other types of backbone.

As a result of this heterogeneity, TCP is facing serious performance problems, concerning the fact that it is not able to handle, for example, a wireless network's unique characteristics. 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 focusing on, while packet losses over wireless links are primarily due to fading channels, or handoffs. In this second situation TCP wrongly continues to behave under the rules of congestion [15].

In this paper, we are trying to demonstrate the behavior of an ideal protocol, in response to the different situations, which discretely describe a network state.

We are interested in end-to-end solutions, which do not require any modification to the network's infrastructure. We propose an improvement to the core mechanism of the experimental protocol TCP-Probing [13, 14] and we present some results showing the potential of our work. We also use two new metrics introduced in [10], in order to capture the protocol's behavior in terms of energy expenditure and discovery of unexploited available bandwidth.

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

- Fairness. Every flow should be fair to all the 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 loss of 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 [12], TCP-Reno [1], TCP-NewReno [8], TCP-SACK [11]) 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.

We will make use of four different, but very realistic scenarios, to show that an adaptive recovery strategy would approach better the aforementioned goal.

A. Congestion

The most common scenario in a packet network is that of the congested channel. As noted before, in this situation and after a packet is lost, TCP reduces the sender's congestion window and extends the timeout period. This situation appears mostly in wired networks, where standard TCP has a fairly good behavior. However, even in a wireless network, in case of high-level of contention, this might be the appropriate action.

B. 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 common in heterogeneous networks, where bandwidth becomes available rapidly, in this scenario we study the protocol’s performance, over a wired topology. In such a situation and after a packet loss, the appropriate behavior of a protocol is to inspect the network’s condition and become aware of the available bandwidth. Backing off under these circumstances (as standard TCP would) might not be the right action.

C. 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 buffering, and as a result there is no 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 is appropriate. The absence of such an adaptive mechanism in TCP's error recovery costs in performance. For example, during a handoff period that lasts 5 RTTs, TCP would try to transmit 1KB 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.

D. Fading Channels

Random transient errors due to fading channels declare another situation that should be taken into consideration in a protocol's behavior. Similarly 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 differently, 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.
- 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, then 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.
- A transient error does not call for either timeout or congestion window adjustment.
- During a dense error or a handoff, a probing mechanism can reduce unnecessary overhead.

II. DECOUPLING ERROR RECOVERY FROM ESTIMATION

All of the existing widely deployed TCP versions are basically trying to avoid, detect and recover from congestion. The congestion-control algorithm, used in TCP-Tahoe [12] includes Slow-Start, Congestion Avoidance, and Fast Retransmit. TCP-Reno [1] introduces Fast Recovery in conjunction with Fast Retransmit. TCP-NewReno [8] 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 [11] 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 because of congestion [13].

By adding a probing mechanism onto standard TCP we can achieve two more goals. Firstly, a probing mechanism is trying to inspect the network load whenever an error is detected and in this way it enables error classification (i.e. due to congestion or to 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 [13, 14] instead of retransmitting the whole data segment, as would standard TCP, it 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 will be added to the overall overhead by losing the probe segment, than by losing the full data segment, as would standard TCP.

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

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

A. 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 some 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 the occurred event (timeout or 3 dacks). This is called “Immediate Recovery”. Otherwise, the protocol enters the Slow-Start phase.

B. Contention Estimation

In a packet-switched network, a situation where no congestion is indicated 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 [4, 5]. 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 3 dacks, even when the packet is finally lost. Instead, it enters Slow-Start as in Reno.

Many proposals tried to classify the losses through different estimation techniques. Barman and Matta [2], proposed an improvement on TCP New Reno, New Reno-FF, a technique that is based on average and variance of the round trip time using a filter called Flip-Flop, a filter which is augmented with history information. Another classification technique proposed by Liu, Matta and Crovella [9] 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, Harris and Shenoy [6] and is referred to as E-TCP. E-TCP uses a new acknowledgement packet format and an agent to assist E-TCP in implementation and in this way 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 [3] implemented a receiver-oriented technique to distinguish congestion losses from corruption losses.

III. IMPLEMENTATION

A. The Core Probing Mechanism

A probe cycle uses two probing segments (PROBE1, PROBE2) and their corresponding acknowledgments (PR1_ACK and PR2_ACK), implemented as option extensions to the TCP header. As noted before the segments carry no payload.

The sender initiates a probe cycle by transmitting a PROBE1 segment to which the receiver immediately responds with a PR1_ACK, upon receipt of which the sender transmits a PROBE2. The receiver acknowledges this second probing with a PR2_ACK and returns to the ESTAB state. The sender makes an RTT measurement based on the time delay between sending the PROBE1 and receiving the PR1_ACK, and another based on the exchange of PROBE2 and PR2_ACK.

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 PR1_ACK segment are missing, and the same again for the PROBE2/PR2_ACK segments. The second is measurement timer, used to measure each of the two RTTs from the probe cycle, in turn. The probe timer is set to the estimated RTT value current at the time the probe cycle is triggered. For a full analysis of the probe mechanism along with a state diagram, see [13, 14].

B. Freezing Through Probing

A primary part of the probing mechanism, which directly impacts the protocol's performance, is the part that is responsible for the recovery strategy. The recovery strategy will be followed 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. Depending on the measurements of the probe cycles the appropriate recovery will be applied.

TCP-Probing exploits due to Immediate Recovery the available bandwidth of a wireless channel more effectively than TCP-Reno / TCP-NewReno would. TCP-Reno / TCP-NewReno would back-off, considering that congestion is indicated. 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. TCP-Probing takes advantage of the recent measurements gathered by the probe cycle and acts like TCP-Reno or TCP-Tahoe (the decision between TCP-Reno and TCP-Tahoe is explained later). As noted before, a new feature is added to TCP-Probing, the vegas_predictor. The calculation of this predictor takes place on every RTT. If this predictor shows high contention, none of the above happens. The probe cycle is skipped and TCP-Probing acts like TCP-Reno. In such case, the protocol's performance equals TCP-Reno's in a wired network.

We implement the above plan as follows. Firstly, the vegas_predictor is calculated:

*vegas_expected_throughput = cwnd_ / best_rtt_*

*vegas_actual_throughput = cwnd_ / last_rtt_*

vegas_difference=vegas_expected_throughput - vegas_actual_throughput

If vegas_difference is greater than 4, then the probe cycles are skipped. Otherwise, if a packet goes lost, the probing mechanism is triggered. Upon exiting the probe cycle, we calculate 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 measurement of probe RTT (best_probe_rtt_, worst_probe_rtt_ - that is, the smallest and largest probe RTT during the communication, respectively). The more the distance from the best_probe_rtt grows the more conservatively we will recover.

average_probe_rtt= (best_probe_rtt_+worst_probe_rtt_) / 2

Comparing those two values, we know if the cp_rtt is above or below the average_probe_rtt. In each of the two cases (above or below), we compare the probe RTTs with the smallest (best) RTT during the communication. After these two comparisons we have a good estimation of the network's state, in order to recover in the appropriate way.

```
cp_rtt_ = ALFA*sample_probe_rtt

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  (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. First of all, and although it seems to work well, the congestion predictor that indicates the skipping of the probe cycles (vegas_predictor), might need to be replaced by a more sophisticated mechanism that makes a better estimation of network conditions. Another issue that calls for further investigation is the number of the probe segments that are sent. It may be better for the probing mechanism to monitor the network by one probe measurement only, avoiding the loss 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 ¾ of its previous value may need to become more adjustable, in order to exploit the available bandwidth in a more sophisticated way.

IV. EXPERIMENTAL METHODOLOGY

A. Testing Plan

We have implemented our testing plan on the ns-2 network simulator. The network topology used as a test-bed is the typical single-bottleneck *dumbbell*, as shown in Figure 1. The link's capacity (bw_bottleneck) is 100Mbps. The bw_src is 10Mbps, and the bw_dst either 1Mbs or 10Mbps. We used equal number of source and sink nodes. We simulated an heterogeneous (wired and wireless) network with ns-2 error models which were inserted into the access links at the sink nodes. The Bernoulli model was used to simulate link-level errors with configurable packet error rate (PER). The number of flows occasionally changes for the different scenarios. The simulation time was fixed at 60 seconds, a time-period seemed appropriate to allow all protocols to demonstrate their potential.

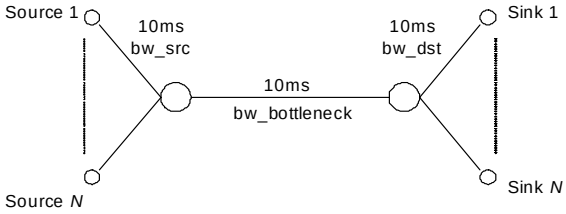

Figure 1. Network topology

In the first scenarios, ftp flows are entering the system within the first seconds. All flows are fixed, during the rest 58 seconds. In order to evaluate how efficiently and fairly the protocols can exploit available bandwidth, we used, additionally, scenarios with graduated contention decrease.

B. 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 utilization. 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 [7] and defined as:

$$Fairness = \frac{(\sum_{i=0}^{n} Throughput_i)^2}{n(\sum_{i=0}^{n} Throughput_i^2)}$$

where $Throughput_i$ is the Throughput of the i_{th} flow and n the flow number.

In order to capture the amount of *extra* energy expended, we introduce a new metric. Extra Energy Expenditure (3E) takes into account the difference of achieved Throughput from maximum Throughput ($Throughput_{max}$) for the given channel conditions, the difference of Goodput from Throughput, attempting to locate the Goodput as a point within a line that starts from 0 and ends at $Throughput_{max}$. The metric 3E takes values from 0 to 1, attempting to capture both distances.

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

where a=1 and b=0.3

When Goodput approaches Throughput which approaches 0, the extra expenditure is only due to time waiting (probably in an idle state). We assume that the extra expenditure at this stage is 0.3 (the first term is 0). Instead, when Goodput=Throughput=$Throughput_{max}$ the extra expenditure is 0, since all the expended energy has been invested into efficient transmissions. Also, when $Throughput_{max}$= 100, Throughput=99, Goodput=1, the extra expenditure due to unsuccessful retransmission grows to an almost maximum value (0.993)

We need to introduce another metric as well, in order for us to capture the level of *Unexploited Available Resources* (UAR). That is, how well did we exploit the windows of opportunities for successful transmissions. Reasonably, the case of Goodput=Throughput=0 should not give us at this point a minor (as with the 3E metric) but a major penalty.

$$UAR = 1 - [a\frac{Throughput}{Throughput_{max}} + b\frac{Goodput}{Throughput}]$$

where a=0.5 and b=0.5. The UAR index ranges also from 0 to 1, expressing also a negative performance aspect.

V. RESULTS AND DISCUSSION

Two versions of TCP took place in the simulations. The TCP-NewReno and TCP-Probing [13, 14]. Many measurements with different characteristics have been carried out, in order to understand each protocol's behavior. Here we will present 4 simple scenarios with characteristics that can show the efficiency of our adaptive recovery strategy. Each version of the protocol was tested by itself, separately from the others. So, the error control mechanism can demonstrate its capability, without being influenced by the presence of other type of flows in each channel.

A. A Full Wired Scenario

In this first scenario, the bw_dst is 10Mbps, in order for the flows to have enough fair-share to expand their windows. Our purpose here is to demonstrate that this version of TCP-Probing performs good enough in a wired environment too, compared to the traditional TCP-Probing (without the use of the vegas_predictor). In fact, in some cases TCP-Probing slightly outperforms New Reno. This is mostly happening because, after a packet is lost because of buffer overflow, and should vegas_predictor indicate no congestion (since there is enough throughput capacity), TCP-Probing immediately gets aware of this capacity and recovers with Immediate Recovery. The above can be understood from the Goodput and UAR Charts below. In this scenario there is no difference in the energy expenditure of the two protocols. The Fairness Index of the two protocols is very close to 1.

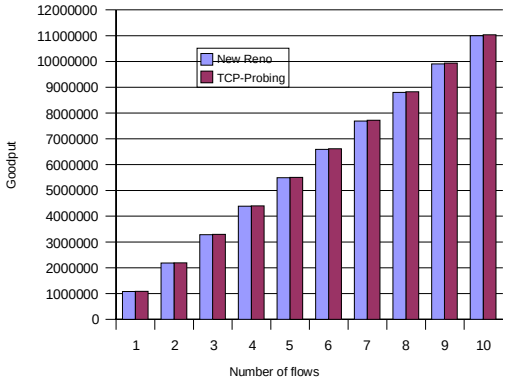

Figure 2. Goodput on Wired

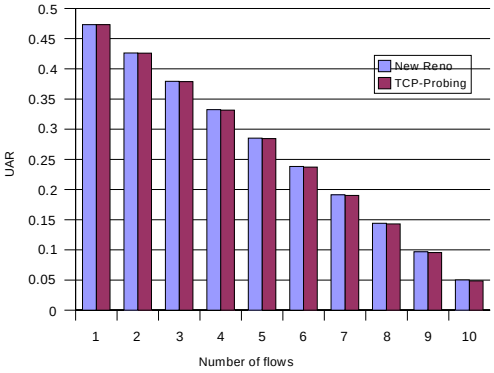

Figure 3. UAR on Wired

B. A Wired Scenario with Contention Decrease

In this scenario, we present a simple wired environment in order to understand the basic behavior of the protocols in conjunction with contention decrease. We measure Unexploited Available Resources Index (UAR) 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 1Mbps 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 every Decrease_Step a large amount of the bottleneck's throughput capacity is freed up.

As can be seen from Figures 4 and 5, 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.

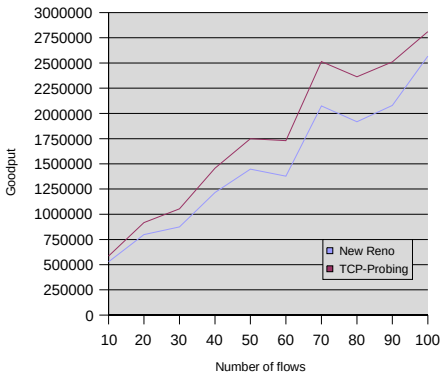

Figure 4. Goodput on Wired/Congestion Decrease

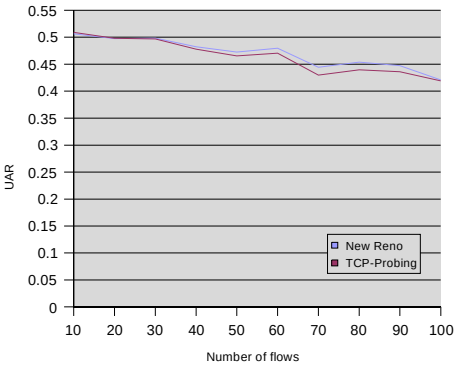

Figure 5. UAR on Wired/Congestion Decrease

C. Wireless Scenario with Handoffs

Handoffs are very common events in wireless networks. This scenario intent to present some results that monitor the behavior of the protocols in a situation of frequent handoffs. The bw_dst is set at 10Mbps and the protocols experience a situation where a handoff event occurs every 5 seconds and lasts for 0.5 seconds.

Our purpose here is to show the inefficient recovery strategy of standard TCP. This inefficiency causes TCP not only to perform poorly in such an environment, but also to consume a lot of energy. This makes it inappropriate for battery-powered devices. In Figures 6, 7 and 8 TCP-Probing clearly outperforms TCP-NewReno in matters of Goodput, Unexploited Available Bandwidth and Extra Energy Expenditure. 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 continues to send data segments. These segments are dropped, until the end of the "blackout". After the end of it, although a lot of bandwidth is available, TCP-NewReno transmits only 1KB and enters Slow-Start phase. On the contrary, TCP-Probing suspends data transmission for as long as the handoff is present (in fact it loses the probe segments which are 40 Bytes each). After the end of the handoff period TCP-Probing monitors the conditions of the network (by the successful probe cycles) and gets immediately aware of the available bandwidth, which immediately exploits. Hence, it acts with respect to energy consumption, while it is aggressive only when the network conditions allow it.

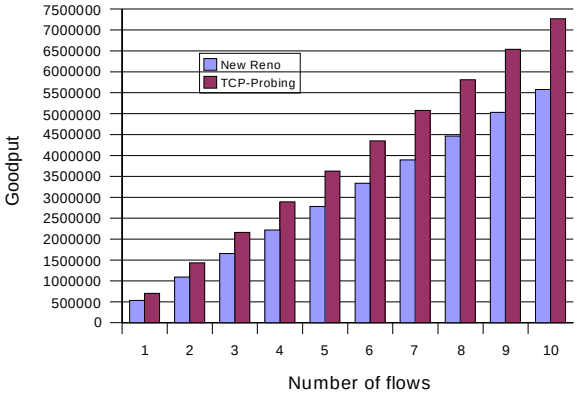

Figure 6. Goodput on Wireless/Handoffs

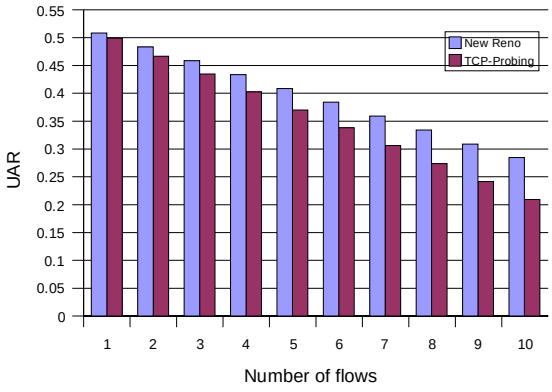

Figure 7. UAR on Wireless/Handoffs

D. A Wireless Scenario with Errors

In the last scenario, we simulated 60 flows, with a variable error-rate from 0.01 to 0.6 PER and bw_dst set at 1Mbps, which means that congestion is often indicated. 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.

As it can be seen from Figures 9, 10 and 11, the adaptive recovery strategy of TCP-Probing, where there is no timeout extension in the Immediate Recovery, performs better that TCP-NewReno. The impact on energy expenditure is also clear. In this scenario, TCP-Probing recovers with Immediate Recovery after a packet loss, which means that there is a small shrinkage of the window. From Figure 9, we can see that, in low error rate, the performance of TCP-NewReno is degraded, while in high error rate the performance increases very much. This might be an indication, as stated in [10] too, that in an environment with high error rate the appropriate action is to be more conservative. The last can be seen from Figures 10 and 11 where TCP-Probing, a more aggressive protocol, does not consume less energy, nor does it exploit more bandwidth than TCP-NewReno.

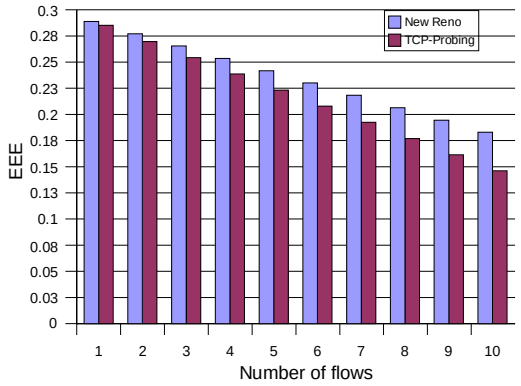

Figure 8. EEE on Wireless/Handoffs

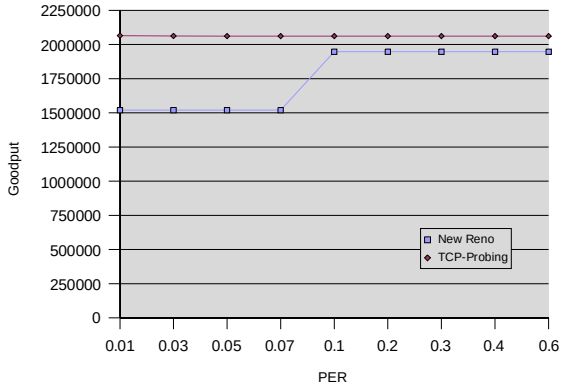

Figure 9. Goodput on Wireless/Errors

In a wireless network, often all of the above (high level of contention, contention decrease, handoffs, and random transient errors) happen altogether. Our purpose here was to start from the beginning and monitor the unique characteristics of each network condition in order to find a recovery strategy that tackles all of them and suites best to heterogeneous networks. Finally, we present the Goodput performance of the two protocols (Figures 12, 13) in a more realistic scenario than the above, where the bw_dst is 1Mbps, 5 handoffs occur during the communication with 1-second duration each, and there is a low error rate of 0.01 PER.

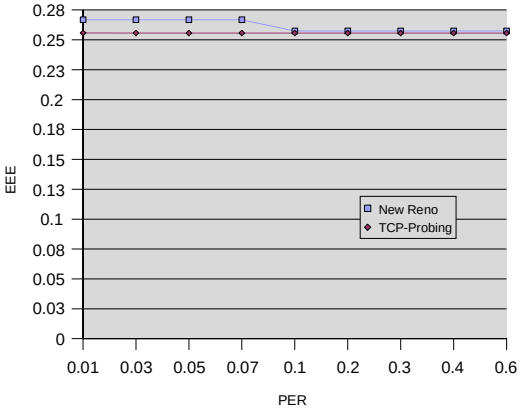

Figure 10. EEE on Wireless/Errors

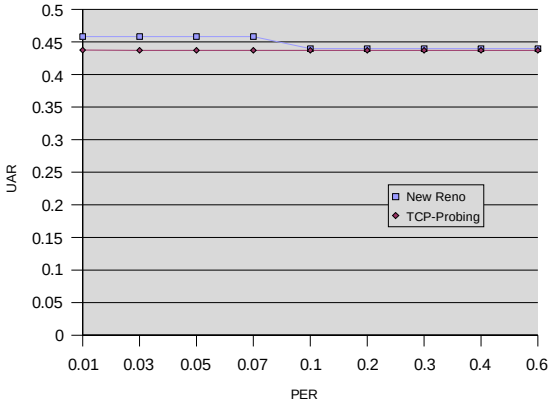

Figure 11. UAR on Wireless/Errors

VI. CONCLUSION

Although, we studied primary the way those decoupled parts work together (without focusing in either the contention estimation or error recovery mechanism), our results are very promising. In a future work, we plan to optimize those two parts, in order to achieve better performance, even in some situations (i.e. Figures 12, 13) where TCP-Probing is outperformed by standard TCP.

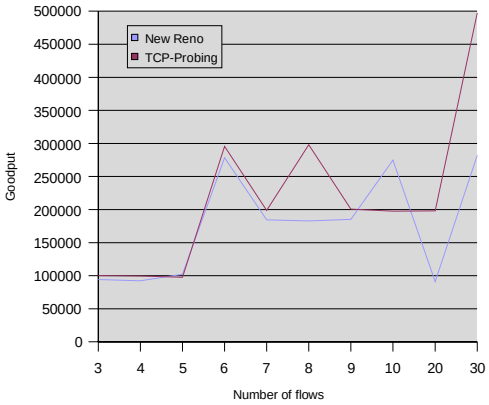

Figure 12. Goodput on Wireless

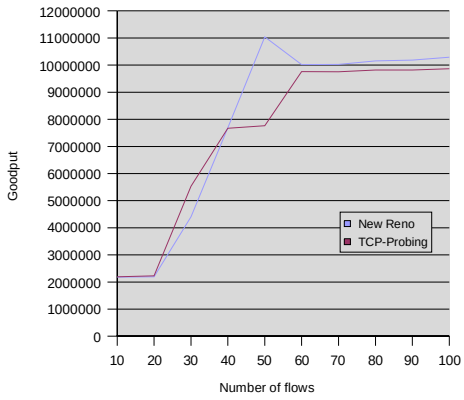

Figure 13. Goodput on Wireless

REFERENCES

[1] M. Allman, V. Paxson, and W. Stevens, “TCP Congestion Control”, RFC2581, April 1999.

[2] Dhiman Barman and Ibrahim Matta. “Effectiveness of Loss Labeling in Improving TCP Performance in Wired/Wireless Networks”, Proceedings of the 10th IEEE International Conference on Network Protocols.

[3] Saad Biaz and Nitin Vaidya. “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.

[4] Lawrence S. Brakmo, Sean W. O'Malley, Larry L. Peterson: TCP Vegas: New Techniques for Congestion Detection and Avoidance. SIGCOMM 1994: 24-35

[5] Lawrence S. Brakmo, Larry L. Peterson: TCP Vegas: End to End Congestion Avoidance on a Global Internet. IEEE Journal on Selected Areas in Communications 13(8): 1465-1480 (1995)

[6] Deddy Chandra, Richard Harris and Nirmala Shenoy. “Congestion and Corruption Loss Detection with Enhanced-TCP”, ATNAC 2003.

[7] D.-M. Chiu and R. Jain, “Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks”, Computer Networks and ISDN Systems, 17(1):1-14, 1989.

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

[9] Jun Liu, Ibrahim Matta and Mark Crovella. “End-to-End Inference of Loss Nature in a Hybrid Wired/Wireless Environment”, In Proceedings of WiOpt'03: Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks, 2003.

[10] L. Mamatas, V.Tsaoussidis: Protocol Behavior: More Effort, More Gains? The 15th IEEE International Symposium on personal, indoor and mobile radio communications, Barcelona, Spain.

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

[12] W. Stevens, "TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms", RFC 2001, January 1997

[13] V. Tsaoussidis, H. Badr, "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 2000.

[14] V. Tsaoussidis and A. Lahanas, Exploiting the Adaptive Properties of a Probing Device for TCP in Heterogeneous Networks" The Journal of Computer Communications COMCOM, Elsevier Science, pp 177-192, Volume 26, Issue 2, February 2003

[15] V. Tsaoussidis and I. Matta, “Open issues on TCP for Mobile Computing”, Journal of Wireless Communications and Mobile Computing, Wiley Academic Publishers, Issue 2, Vol. 2, February 2002.