

EXPERIMENTATIONS TOWARDS TCP WESTWOOD APPLICATION IN AD-HOC MOBILE NETWORKS

Ilhem Lengliz⁺

Haifa Touati
CRISTAL ENSI

Farouk Kamoun⁺⁺

Medy Y. Sanadidi
UCLA Computer
Science Department

Campus universitaire 2010 Manouba

{⁺Ilhem.Lengliz, ⁺⁺Farouk.Kamoun}@ensi.rnu.tn

medy@cs.ucla.edu

ABSTRACT

We study the performance of TCP Westwood (TCPW), a TCP protocol with a sender-side modification of the window congestion control scheme. The basic principle in this protocol is to continuously estimate, at the TCP sender, the packet rate of the connection based on the ACK reception rate. This estimation is then used to compute congestion window and slow start threshold settings after a congestion episode.

Given that it has been yet established through experimental studies that TCPW exhibits significant improvements in throughput performance over Reno in various environments, we are performing some preliminary measurements on TCPW performance, in comparison to that of TCP New Reno in wired/wireless networks. This is the first step in the study of TCPW when used in ad-hoc mobile networks. In this paper we present the results of some experimentations carried out in the CRISTAL Laboratory with a FreeBSD TCPW ABSE protocol implementation.

Keywords : TCP Westwood ABSE, Wireless, ad-hoc, congestion control, experimentation.

1 INTRODUCTION

TCP Westwood has been initially designed in [1]. This new protocol relies on a simple modification of the TCP source protocol behavior to achieve a faster recovery. This mechanism of faster recovery consists in choosing both a slow start threshold and a congestion window that result from the effective connection rate at the time congestion is experienced. Hence, TCPW attempts to make a more “informed” decision, in contrast with TCP Reno, which automatically halves the congestion window after three duplicate ACKs.

Like TCP Reno, TCPW cannot distinguish between buffer overflow loss and random loss. However, in presence of random loss, TCP Reno overreacts and reduces the window by half. TCPW, on the other hand, after packet loss and retransmission timeout, resumes with the previous

window as long as the bottleneck is not yet saturated (i.e., there is no buffer overflow).

To prevent the unnecessary window reduction of TCP Reno in case of random packet loss, and more precisely the loss caused when there is wireless links, several schemes have been proposed [2]. All of these schemes require the cooperation of intermediate routers/proxies. TCPW does not require any specific intervention/support from intermediate route, thus preserving the original “end to end design” principle [3].

One of the refinement of TCPW is proposed in [4] and named TCPW RE (Rate Estimation) to address TCP Reno Friendliness. The other is TCPW+, described and studied in [5], it is intended to improve TCPW performance in the case of Internet transmissions. The TCPW ABSE protocol used in this work to perform the experimentations has been proposed in [7]. It palliate TCP efficiency degradation in packet loss environment.

In this paper we present the results of preliminary experimental measurements on TCPW performances in wireless environment.

The remaining of this paper is organized as follows. In Section 2, we present the TCPW ABSE protocol. In Section 3, the experimental configuration and related parameters are described, as well as the measurement results. Section 4 concludes the paper and draws some perspectives of this work.

2 TCPW ABSE PROTOCOL

TCPW ABSE (Adaptive Bandwidth Share Estimation) [6] is a sender-only modification of TCP NewReno. The TCP sender Adaptively determines a Bandwidth Share Estimate (TCP-ABSE). The estimate is based on information in the ACKs, and the rate at which the ACKs are received. After a packet loss indication, which could be due to either congestion or link errors, the sender uses the estimated bandwidth to properly set the congestion window and the slow start threshold. Further details regarding bandwidth estimation are provided in following Sections. For now, let us assume that a sender has determined the connection bandwidth estimate as mentioned above, and let us

describe how the estimate is used to properly set $cwin$ and $ssthresh$ after a packet loss indication.

In TCPW, congestion window dynamics during slow start and congestion avoidance are unchanged; that is, they increase exponentially and linearly, respectively, as in current TCP NewReno.

A packet loss is indicated by :

- (a) the reception of 3 DUPACKs,
or
- (b) a coarse timeout expiration.

In case (a), TCPW sets $cwin$ and $ssthresh$ as follows :

```

if (3 DUPACKs are received)
ssthresh = (ABSE * RTTmin) / seg_size;
if (cwin > ssthresh) /* congestion avoid. */
cwin = ssthresh;
endif
endif

```

In case a packet loss is indicated by a timeout expiration, $cwin$ and $ssthresh$ are set as follows:

```

if (coarse timeout expires)
cwin = 1;
ssthresh = (ABSE * RTTmin) / seg_size;
if (ssthresh < 2)
ssthresh = 2;
endif;
endif

```

The rationale of the algorithm above is that after a timeout, $cwin$ and the $ssthresh$ are set equal to 1 and ABSE, respectively. Thus, the basic Reno behavior is still captured, while a reasonably speedy recovery is ensured by setting $ssthresh$ to the value of ABSE.

2.1 Adaptive Bandwidth Share Estimation

The ABSE algorithm adapts to the congestion level in performing its bandwidth sampling, and employs a filter that adapts to the round trip time and to the rate of change of network conditions. The bandwidth share estimation is computed using a time varying coefficient, exponentially-weighted moving average (EWMA) filter, which has both adaptive gain and adaptive sampling. Let t_k be the time instant at which the k_{th} ACK is received at the sender. Let s_k be the bandwidth share sample, and \hat{s}_k the filtered estimate of the bandwidth share at time t_k .

Let α_k be the time-varying coefficient at t_k . The ABSE filter is then given by :

$$\hat{s}_k = \alpha_k \cdot \hat{s}_{k-1} + (1 - \alpha_k) s_k \quad (1)$$

where

$$\alpha_k = \frac{2 \tau_k - \Delta t_k}{2 \tau_k + \Delta t_k}$$

and τ_k a filter parameter which determines the filter gain, and varies over time adapting to path conditions. In the filter formula above, the bandwidth sample at time k is :

$$s_k = \frac{\sum_{t_j > t_k - T_k} d_j}{T_k} \quad (2)$$

where d_j is the number of bytes that have been reported delivered by the j_{th} ACK, and T_k is an interval over which the bandwidth sample is calculated.

2.1.1 ABSE adaptive sampling

ABSE provides an adaptive sampling scheme, in which the time interval T_k associated with the k_{th} received ACK is appropriately chosen, depending on the network congestion level. To determine the network congestion level, a simple throughput filter is proposed to estimate the recent throughput achieved. By comparing this estimate with the instantaneous sending rate obtained from $cwin$, the path congestion level is determined.

When the k_{th} ACK arrives, a sample of throughput during the previous RTT is calculated as : RTT , where d_j is the amount of data reported by ACK j . In [6], the value ($\varepsilon = 0.6$) has been employed for the constant-gain filter to calculate the recent throughput as :

$$T\hat{h}_k = \varepsilon T\hat{h}_{k-1} + (1 - \varepsilon) T h_k \quad (3)$$

When $T\hat{h}_k * RTT$ is larger than the current $cwin$ value, indicating a path without congestion, T_k is set to T_{min} . Otherwise, T_k is set to :

$$T_k = RTT * \frac{cwin - (T\hat{h}_k * RTT_{min})}{cwin} \quad (4)$$

2.1.2 Filter Gain Adaptation

When τ_k is larger, α_k will be larger and the filter tends to be more stable and less agile. After a certain point, α_k basically stays unchanged as the value of τ_k increases.

The parameter τ_k adapts to network conditions to dampen estimates when the network exhibits very unstable behavior, and react quickly to persistent changes. A stability detection filter can be used to dynamically change the value of τ_k . The network instability U is measured with a time-constant EWMA filter [6,7] :

$$U_k = \beta U_{k-1} + (1 - \beta) |s_k - s_{k-1}| \quad (5)$$

In (5), s_k is the k_{th} sample, and β is the gain of this filter, which is set to be 0.6 in [6]. When the network exhibits high instability, the consecutive observations diverge from each other, as a result,

U_k increases. Under this condition, increasing the value of τ_k makes the ABSE filter (1) more stable.

When a TCP connection is operating normally, the interval between the consecutive acknowledgements are likely to vary between the smallest the bottleneck capacity allows, and one RTT. Therefore, τ_k should be larger than one RTT, thus $\tau_{min} = RTT$. In [6] τ_k is set to be :

$$\tau_k = RTT + N * RTT \frac{U_k}{U_{max}} \quad (6)$$

The value of RTT can be obtained from the smoothed RTT estimated in TCP [6].

2.2 Accounting for delayed and cumulative ACKs on bandwidth measurement

This issue has been addressed in [6]. DUPACKs should count toward the bandwidth estimation since their arrival indicates a successfully received segment, albeit in the wrong order. As a consequence, a cumulative ACK should only count as one segment's worth of data since duplicate ACKs ought to have been already taken into account. Further complications result from *delayed ACKs*. The standard TCP implementation provides for an ACK being sent back once every other in-sequence segment received, or if a 200 ms timeout expires after the reception of a single segment [7]. The combination of delayed and cumulative ACKs can potentially disrupt the bandwidth estimation process. Therefore, in [7] two important aspects of the bandwidth estimation process are stressed :

- (a) the source must keep track of the number of DUPACKs it has received before new data is acknowledged;
- (b) the source should be able to detect delayed ACKs and act accordingly.

The approach chosen to take care of these two issues can be found in the AckedCount procedure, detailed below, showing the set of actions to be undertaken upon the reception of an ACK, for a correct determination of the number of packets that should be accounted for by the bandwidth estimation procedure, indicated by the variable `acked` in the pseudocode. The key variable is `accounted_for`, which keeps track of the received DUPACKs.

When an ACK is received, the number of segments it acknowledges is first determined (`cumul_ack`). If `cumul_ack` is equal to 0, then the received ACK is clearly a DUPACK and counts as 1 segment towards the BWE; the DUPACK count is also updated. If `cumul_ack` is larger than 1, the received ACK is either a delayed ACK or a cumulative ACK following a retransmission event; in that case, the number of ACKed segments is to

be checked against the number of segments already accounted for (`accounted_for`). If the received ACK acknowledges fewer or the same number of segments than expected, it means that the "missing" segments were already accounted for when DUPACKs were received, and they should not be counted twice. If the received ACK acknowledges more segments than expected, it means that although part of them were already accounted for by way of DUPACKs, the rest are cumulatively acknowledged by the currentACK; therefore, the currentACK should only count as the cumulatively acknowledged segments. It should be noted that the last condition correctly estimates the delayed ACKs

```
(cumul_ack = 2 and accounted_for = 0):
PROCEDURE AckedCount
cumul_ack = current_ack_seqno -
last_ack_seqno;
if (cumul_ack = 0)
accounted_for = accounted_for + 1;
cumul_ack = 1;
endif
if (cumul_ack > 1)
if (accounted_for >= cumul_ack)
accounted_for = accounted_for -
cumul_ack;
cumul_ack = 1;
else if (accounted_for < cumul_ack)
cumul_ack = cumul_ack - accounted_for;
accounted_for = 0;
endif
endif
last_ack_seqno = current_ack_seqno;
acked = cumul_ack;
return(acked);
END PROCEDURE
```

3 MEASUREMENTS

3.1 Test bed configuration

In all the experiments we used the topology depicted in figure1. The test bed consists of two wired hosts, one laptop equipped with a 802.11 pcmcia card and an Access point that connect the wired portion to the wireless one. Wired stations are connected on a 10 Mbps Ethernet LAN and the wireless station portion is a 802.11 b WLAN (11 Mbps). The RTT from wired nodes to the Access point averages around 1,093 ms, whereas the RTT on the wireless portion is about 2ms. The wireless card provides real time indication on the transmission quality by displaying a transmission state as indicated in table1. Besides, wired nodes run FreeBSD 4.5 OS and the laptop runs on Windows2000 OS. On the first Wired station, we installed TCPW ABSE, while the second wired station uses its default TCP packages, namely TCP NewReno. TCPSack option is disabled on both stations.

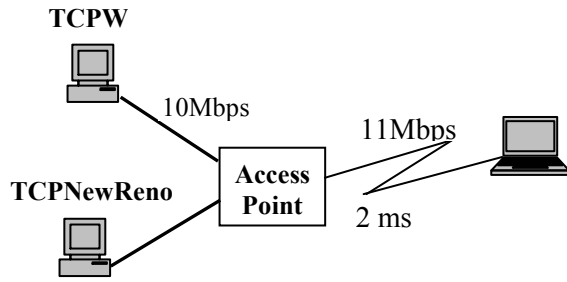


Figure1 : Laboratory test bed configuration

Table 1 : Wireless card transmission states

Transmission state	Explanation
Excellent	Transmission is at its maximum rate
Fair	Transmission is good
Poor	Minimum rate
Not connected	Can detect the receiver but can't reach it
Not Applicable	Blackout situation

3.2 Experiments description

To evaluate TCPW ABSE throughput versus TCP New Reno throughput in a wireless environment, we investigated three scenarios :

- Scenario 1 : it is an indoor environment with a static wireless station placed about 7 meters far from the Access point, on the wireless side, the transmission quality oscillates between “Excellent” and “fair” states.
- Scenario 2 : in this case the wireless station moves, but usually in an indoor environment, to introduce more disturb we, manually, hide the wireless card. In this case, the wireless transmission quality goes down to “poor” and “not connected” states.
- Scenario 3 : it is a mixture of indoor and outdoor environments. In fact during data transfer, we move away the laptop from the access point. The wireless transmission quality switches from “fair” to “poor” then “Not connected” and finally “Not applicable”, which reflects a blackout situation, and then we move back the laptop, the transmission quality moves back to its “Excellent” state.

The wireless TCP receiver implements delayed ACK. Notice that this introduces a complication in bandwidth estimation algorithm as delayed ACKs represent noise to be filtered. The experiments were conducted using repeated single file transfers. In scenario 1 and 2 we sent 10 Mbytes data files, whereas in the third scenario larger data files were used (40 Mbytes). In addition, transmissions on the connections were traced using tcpdump [8], the traces acquired from tcpdump were then analyzed

using the tcptrace tool [9]. In all the experiments, flows grow from wired to wireless portion.

3.3 Results

3.3.1 Scenario1

Figure 2 shows that when the wireless station doesn't move from it's initial location, TCPW ABSE doesn't perform better than TCP New Reno, moreover, the average on the measured rates are as follows : TCPW ABSE : 467.45 kbps; TCP Reno = 472.61 kbps. This is because TCPW ABSE is likely to be used in packet loss environment, unless TCP New Reno can be still applied since it induces less computational overhead.

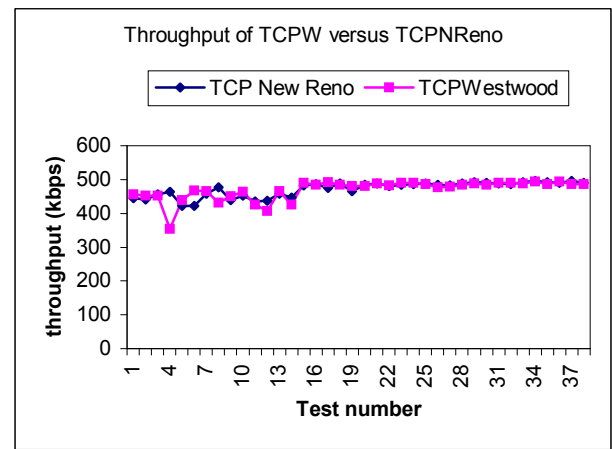


Figure 2 : The wireless station is in static location

3.3.2 Scenario 2

The behavior of TCPW ABSE when the wireless station moves continuously is completely different, in fact it achieves a better rate than that of TCP New Reno, even when it decreases. In this case the average on the measured rates become :TCPW ABSE = 346.82 kbps; TCP New Reno = 335.03 kbps. Besides the final decrease in the two throughputs is due to that the hiding of the wireless card has been performed manually.

3.3.3 Scenario 3

As plotted in figure 4, it is worth noting that TCPW ABSE achieves better average throughput than TCP New Reno. Hence, the measured throughput averages are : TCPNew Reno = 227.47 Kbps; TCPW ABSE = 232.76 Kbps.

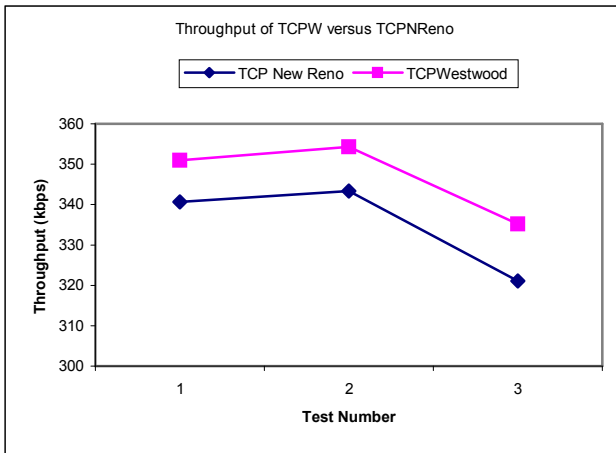


Figure 3 : The wireless station is moving indoor

This is due to the more efficient resetting of cwin and ssthresh, especially when packets losses are not due to congestion, but to inherent wireless environment characteristics. In this scenario, the measured packet loss average is 10^{-3} .

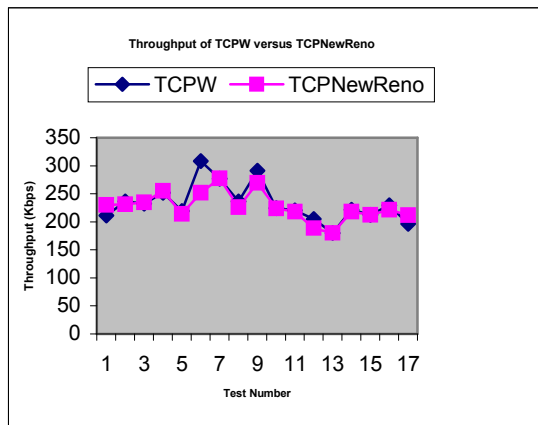


Figure 4 : The wireless station is moving indoor/outdoor

4 CONCLUSION AND FURTHER WORK

In this work we have presented the study of TCPW ABSE, in comparison to that of TCP New Reno, by carrying out laboratory experimentations. On the positive results of these tests, we saw the improvement brought by TCPW ABSE on the throughput, in a mobile environment. So we can now test the implementation of TCPW ABSE in ad-hoc mobile networks, regarding some improvement in the test configuration and other test out of our laboratory.

REFERENCES

- [1] C. Casetti, M. Gerla, S. Lee, S. Mascolo, and M. Sanadidi, "TCP with Faster Recovery," MILCOM 2000, Los Angeles, CA, October 2000.
- [2] H. Balakrishnan, V. N. Padmanabhan, S. Seshan, and R. H. Katz, "A Comparison of Mechanisms for Improving TCP Performance over Wireless Links," IEEE/ACM Transactions on Networking, December 1997.
- [3] M. Gerla, M. Y. Sanadidi, R. Wang, A. Zanella, C. Casetti, S. Mascolo, "TCP Westwood: Congestion Window Control Using Bandwidth Estimation", In Proceedings of IEEE Globecom 2001, Volume: 3, pp1698-1702, San Antonio, Texas, USA, November 25-29, 2001.
- [4] R. Wang, M Valla, M. Y. Sanadidi, B. K.F Ng, M. Gerla, "Efficiency/Friendliness Tradeoffs in TCP Westwood", Seventh IEEE Symposium on Computers and Communications, Taormina, Italy, 1-4 July, 2002.
- [5] R. Ferorelli, L. A. Grieco, S. Mascolo, G. Piscitelli, P. Camarda, "Live Internet measurements using Westwood+ TCP Congestion Control", IEEE Globecom 2002, Taipei, Taiwan, November 18-20, 2002.
- [6] R. Wang, M Valla, M. Y. Sanadidi, M. Gerla, "Adaptive Bandwidth Share Estimation in TCPWestwood", IEEE Globecom 2002, Taipei, Taiwan, November 18-20, 2002.
- [7] C. Casetti, M. Gerla, S. Mascolo, M. Y. Sanadidi, R. Wang "TCP Westwood : End-to-End Bandwidth Estimation for Enhanced Transport over Wireless Links", Journal of Wireless Networks, Volume 8, pp 467-479, 2002.
- [8] www.tcpdump.org
- [9] www.tcptrace.org