Novel delayed ACK techniques for improving TCP performance in multihop wireless networks

Eitan Altman*
INRIA, BP93
2004 Route des Lucioles,
06902 Sophia Antipolis Cedex, France
email: altman@sophia.inria.fr
Tel: (33) 4 92 38 77 86, Fax: (33) 4 92 38 77 65.

Tania Jiménez
C.E.S.I.M.O., Facultad de Ingeniería,
Universidad de Los Andes,
Mérida, Venezuela
email: tjimenez@ula.ve

Abstract

We study in this paper TCP performance over a static multihop network that uses IEEE 802.11 protocol for access. For such networks it has been shown in [6] that TCP performance is mainly determined by the hidden terminal effects (and not by drop probabilities at buffers) which limits the number of packets that can be transmitted simultaneously in the network. Two techniques have been proposed in [6] to improve the performance, one based on RED type technique and the other on adaptive spacing at the link layer. In this paper we propose new approaches for improving the performance based on thinning the ACK streams that competes over the same radio resources as the TCP packets. In particular, we propose a new delayed ACK scheme in which the delay coefficient varies with the sequence number of the TCP packet. Through simulations in ns, we show that the ACK thinning allows to increase TCP throughput by around 50%, which is substantially more than the previous improvement methods.

*Eitan Altman is the correspondence author

1 Introduction

Various mobility-induced aspects of TCP performance over ad hoc networks have been studied in the past, such as link breakage and routing failures and have motivated proposals for improvements of TCP:
(1) ELFN (Explicit Link Failure Notification) [7, 11]: This allows TCP to interact with routing protocols and to freeze its timers when a route failure is detected
(2) Fixed RTO: An important source of degradation of TCP performance is the exponential backoff of TCP during timeouts due to disconnections, which may result in very long silence periods when connection is available again. In [5] it is proposed to disable the exponential backoff.
(3) ATCP proposed in [10] does not change TCP but implements another layer between the network and transport layers.
(4) TCP DOOR: TCP protocol is changed so as to be able to better react to out-of-order packets that may occur frequently in Ad-hoc networks [17].
(5) The ATRA framework [4]: consists of a set of MAC and routing layers mechanisms that reduce route failures, predict route failures before they occur and minimize the latency for route error propagation.

Most of the above work is summarized in [12]. Other work has been devoted to comparison of the performance of TCP under various Ad-Hoc protocols [3, 5].

In this paper we focus on a static multihop network that uses IEEE 802.11 protocol for access, see [6, 15]. For such scenarios it has been shown in [6] that TCP performance is mainly determined by the hidden terminal effects (and not by drop probabilities at buffers) which limits the number of packets that can be transmitted simultaneously in the network (this is called the "spatial reuse"). In particular, for the chain topology that we study in this paper, the spatial reuse factor of the network has been shown in [6] to have some limit which was around 1/4, which means that given h nodes, only around $h/4$ can transmit packets simultaneously. This allowed the authors to conclude that there is an optimal size for the maximum window of TCP which is close to the value of $h/4$. Any increase in the maximum window size results in decreasing TCP throughput, although the loss in performance is not large (around 20%).

Two techniques have been proposed in [6] to improve the performance, one based on RED type technique and the other on adaptive spacing at the link layer. The reported gains in TCP throughput were between 5% and 30%.

In practice the number of nodes cannot be assumed to be known and can also vary in time. Thus the value of the maximum window for TCP for optimal operation may not be known, and we may have to use a large value for reasonable performance in cases there is a possibility that the number of nodes is large. Our first goal in this paper is to obtain a performance of TCP which is good for a large range of number of nodes; this is not possible with standard TCP where we know that good performance can be obtained only for a given number of nodes n by fixing the maximum window size to around $4n$. Our second goal is to further obtain improvement in TCP performance with minimal changes in TCP.

To that end, we observe that the bottleneck of the system is the spatial reuse: the number of packets that can be transmitted simultaneously. Our aim is thus to decrease the flow of ACKs so as to give more bandwidth to TCP data packets. We note that although TCP ACKs are much smaller (40bytes) than TCP data packets (typically between 500 bytes and 1kbyte), their transmission requires the same signaling overhead of the 802.11 MAC, i.e. the RTS and CTS packets that precede each transmission of a packet and an ACK packet (of the MAC layer) for acknowledging successful transmission.

To reduce the ACK flow, we use the delayed ACK option of TCP, in which an ACK is generated for every d TCP packets, with one exception: if the first packet (of an expected group of d packets) arrives at the destination, then after some time interval (typically 100ms) if d packets have not yet arrived, then an acknowledgement is generated without further waiting. The standard delayed ACK option has the value $d = 2$ (see RFC 1122). We observe significant improvement of standard delayed ACK with respect to the nondelayed option. This then motivates us to modify the standard delayed ACK option and to take $d > 2$. Using simulations in ns [1], we test this approach for both permanent as well as short TCP connections. We show that this allows to increase TCP throughput by around 50%, which is substantially more than the previous improvement methods. We then propose a modification in the delayed ACK approach which we call the "dynamic delayed ACK", where the parameter d gradually increases with the sequence number of the acknowledged packet. This allows us to have more ACKs when the window size is small, thus enabling to increase rapidly the throughput, and less ACKs when the window is large (and during which TCP is more vulnerable to losses).

The structure of the paper is as follows. We present in the next section the simulation scenario. Then we present in Section 3 our findings on performance of persistent TCP connections. In Section 4 we introduce our dynamic delayed ACK modification; through simulations of short TCP connections, we compare this proposal to standard TCP (without delayed ACK) and to TCP with delayed ACK with different fixed parameters d. We end with a concluding section.

2 The simulation Scenario

We use ns2 [1] simulate the chain scenario of [6] that consists of n nodes over a line separated by a distance of 200m, as seen in Figure 1. For each wireless node, the transmission range is 250m,

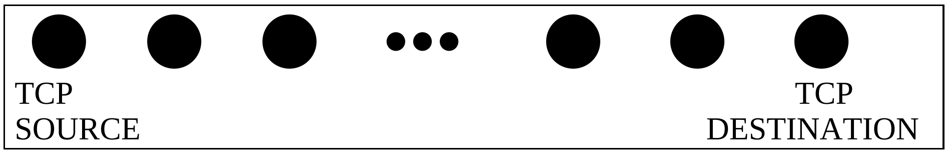

Figure 1: The chain topology

the carrier sensing range is 550m and the interference range is about 550m.

We use the standard two-ray ground propagation model, the IEEE802.11 MAC, and an omni-directional antenna model of ns [1]. an interface queue length of 50 at each node We tested the NewReno version of TCP, which is the most deployed one. We tested four scenarios: 3,9,20 and 30 nodes. The cases of 3 and 9 nodes required 150 sec per simulation (to obtain stationary behavior). The other cases required 1500 sec per simulation. A TCP data packet is taken to be of size 1040 bytes. We used the AODV (Ad hoc On-Demand Distance Vector) routing algorithm (see [14]) in our simulations.

3 Simulation results for persistent TCP connections

3.1 Effect of delayed Ack on TCP throughput

Our simulation results for $n = 9, 20$ and 30 nodes are summarized in Figures 2-4, respectively. In all these cases the hidden terminal effect causes losses. We shall later examine the case of $n = 3$ in which we do not have TCP losses.

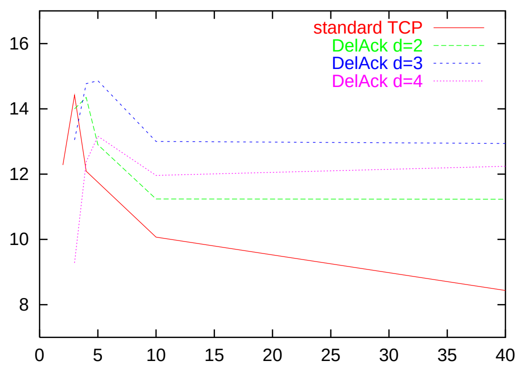

Figure 2: Throughput in pkt/sec for $n = 9$ as a function of the maximum window size

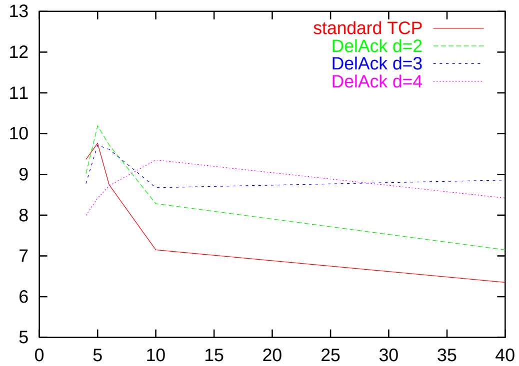

Figure 3: Throughput in pkt/sec for $n = 20$ as a function of the maximum window size

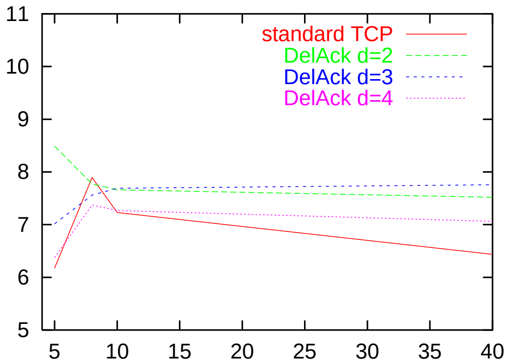

Figure 4: Throughput in pkt/sec for $n = 30$ as a function of the maximum window size

We see that the standard Delayed Ack option ($d = 2$) slightly outperforms the standard TCP (yet with another value of maximum window size) for $n = 9$, and largely outperforms (more than 10%) the standard TCP for $n = 30$. A further improvement is obtained by the Delayed Ack with $d = 3$ (for both $n = 9$ as well as $n = 20$). But the most important improvement that we see is that all delayed ACK versions are better than the standard TCP for maximum window sizes of more than 10, with the options of $d = 3$ or $d = 4$ outperforming the standard delayed ACK option. For $n = 9$, the Delayed ACK version with $d = 3$ is seen to yield between 30% to 40% of improvement over standard TCP for any maximum window sizes larger than 10; in that range it also outperforms standard TCP by 20%-30% for $n = 20$ and by $6\% - 20\%$ for $n = 30$. The version $d = 4$ performs even better for $n = 20$ for maximum windows between 10 to 25. An even better performance of delayed ACK can be obtained by optimizing over the timer duration of the Delayed Ack options, as we shall see later.

Yet the most important conclusion from the curves is the robustness of the Delayed Ack options. In practice, when we do not know the number of nodes, there is no reason to limit the maximum window size to a small value, since this could deteriorate the throughput considerably. When choosing large maximum window, the delayed ACK versions considerably outperform standard TCP. They achieve almost the optimal value that the standard TCP could achieve if it knew the number of nodes and could choose accordingly the maximum window.

For a fixed small size of maximum window size, the Delayed Ack option does not outperform the standard TCP version since most of the time, the window size limits the number of transmitted TCP packets to less than d, which means that the delayed ACK option has to wait until the timer (of 100ms by default) expires before generating an ACK; during that time the source cannot transmit packets.

Next, we plot the window size evolution for $n = 9$ for standard TCP and for TCP with delayed ACK option with $d = 3$. The window size is sampled every 0.1 sec. We see that although

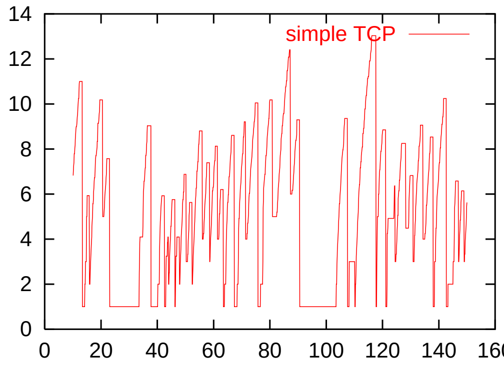

Figure 5: Window size evolution for standard TCP with maximum window of 2000

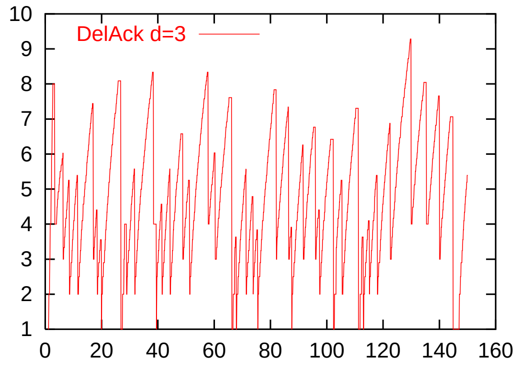

Figure 6: Window size evolution for DelAck TCP with $d = 3$, with maximum window of 2000

the maximum window size is 2000, the actual congestion window does not exceed the value of

13. We see that from the figures that in standard TCP, losses are more frequent and more severe (resulting in timeouts) whereas the $d = 3$ version of delayed ACK does not give rise to timeouts.

In Figure 7 we present the evolution of the congestion window size for standard TCP with maximum window size of 3 for the case of 9 nodes. We know from [6] that a maximum size of between 2 and 3 should indeed give optimal performance (and this is confirmed in Figure 2). We see in Figure 7 that there are almost no losses. Note that the actual window size is the minimum between the congestion window (depicted in the Figure) and the maximum window size (whose value here is 3).

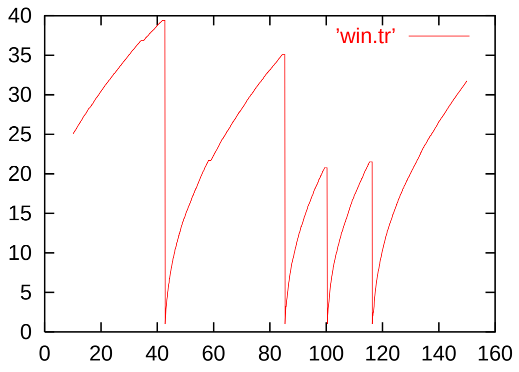

Figure 7: Window size evolution for standard TCP (delayed ACK disabled) with 9 nodes and maximum window size of 3

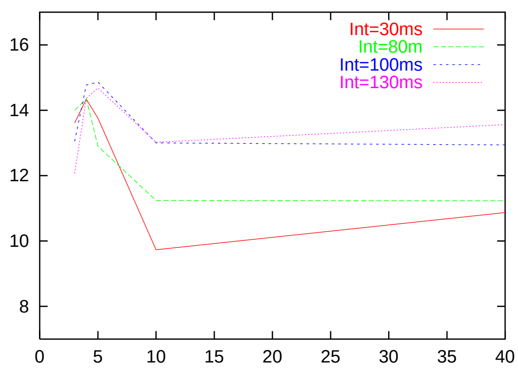

Figure 8: The influence of Delayed Ack interval on TCP throughput, as a function of the maximum window size. $d = 3$

3.2 Varying the delayed Ack time interval

In the previous Figures, all versions using delayed Acks had the default interval of 100msec (as explained in the Introduction). Next, we vary the interval length and check its impact on throughput, see Fig. 8. We consider the delayed ACK version with $d = 3$.

We see that the default value performs quite well, although for small maximum windows, shorter intervals perform slightly better, whereas with large maximum window, a larger interval (130ms) is slightly better. We tried to further increase the time interval beyond 130ms but then the throughput decreased.

3.3 $n = 3$: no hidden terminals

Finally, we consider the case of $n = 3$ nodes. In that case the hidden terminal phenomenon does not occur anymore, and we have not observed TCP losses during the simulation for any value of window size. Even then, delayed ACKs can be used to improve considerably the performance. This is illustrated in Table 1 that gives the number of TCP packets successfully received within 149 sec for $n = 3$. Since there are no losses, then as long as d is greater than the max window, we

	Standard TCP	*Delayed Ack Versions*		
WinMax	Standard	$d = 2$	$d = 3$	$d = 4$
3	6068	6602	6763	2699
2000	6094	6565	6779	6888

Table 1: Number of transmitter packets during 149sec for $n = 3$ as a function of the maximum window size

expect to improve the performance as d gets larger, since TCP packets compete with less ACKs. This is indeed confirmed in Table 1. The improvement that increases from 10% to 15% as d grows from 2 to 4, does not depend on the maximum window (as long as it is greater than d). However for $d = 4$ we see, as can be expected, that we get a bad performance for a maximum window of 3, since the destination always needs to wait till the 100ms interval of the Delayed Ack option expires in order to send an ACK (since the windows allows for sending only 3 data packets).

3.4 Other routing algorithms

As we already mentioned, TCP performance is affected by the fact they have to share the channel with the ACK packets. But ACK are not the only packets that compete for the radio channel. Also packets involved in route discovery and in route maintenance take a share of the radio resources, so it is not surprising that different ad-hoc routing algorithms may result in different performance of TCP even in our simple static chain topology. We shall thus study TCP performance under various routing protocols.

We repeat the simulations for $n = 9$ with the DSR (Dynamic Source Routing) protocol [8] as well as for the DSDV (Destination-Sequenced Distance-Vector) [13], and depict the throughput as a function of the maximum window sizes in Figure 9 and 10, respectively. We make the

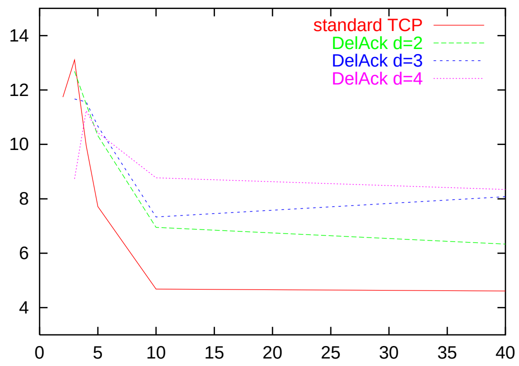

Figure 9: Throughput in pkt/sec for $n =$ 9 as a function of the maximum window size, DSR routing protocol

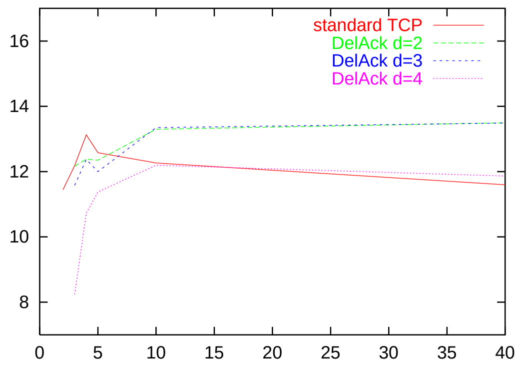

Figure 10: Throughput in pkt/sec for $n =$ 9 as a function of the maximum window size, DSDV routing protocol

following observations:

- In general, the throughputs obtained under AODV are of the same order as under DSDV, and both are larger than under AODV.
- IN AODV, a large degradation in throughput for standard TCP (without delayed ACK) is observed as the maximum window size increases from its best value of 3. The degradation here is from around 13pkt/sec to 4.6pkt/sec (a decrease by 65%) whereas for the AODV routing we obtained (as in [6]) a degradation of only 40%.
- In DSDV, there is a very small degradation of the throughput for standard TCP as the maximum window size increases from its best value of 4 (note also that the best window size is 4, here which is slightly larger than in the value of 3 obtained for the other routing protocols).
- For DSR, the improvement in TCP throughput with the delayed ACK versions of $d = 3$ and $d = 4$ with large maximum windows (which may be necessary when we do not know ahead what the number of hops is) is even larger than in AODV. It is almost twice that of standard TCP, where as in the AODV it was around 1.5 times larger. In DSDV we also obtain an improvement but it is lower than for AODV (around 17% of improvement).

4 Short TCP transfers and dynamic delayed Ack

In a recent paper [9], the authors show that conclusions drawn from simulating permanent TCP connections can be qualitatively quite different than those obtained from simulations of transfers that have large time scale variability. The latter is obtained by replacing the infinite source model by ones in which transfered files have heavy-tailed distributions.

4.1 Simulated scenario

We use TCP connections whose size has a Pareto distribution. Recall that for Pareto distribution, $E[size] = \beta k/(\beta - 1)$ where $\beta = 1.5$ is our shape parameter and where the average file size that we took is $E[size] = 30kbytes$; thus the parameter k equals in our case to 10000. The complementary distribution of a file size is $Pr(size > s) = (k/s)^{\beta}$. The time between the end of a connection till the beginning of the next connection has an exponential distribution with average of half a second. This characterization of TCP sessions is compatible with measured Internet traffic, see e.g. [16].

In addition to the throughput, we now consider the average session delay (the time between the opening of a session till the last ack is received at the source). This performance measure of course does not have a meaning in the previous setting of a persistent TCP connection.

All simulations last 1500sec and use the AODV routing protocol. TCP data packet sizes are again 1000bytes plus 40bytes of headers (for IP and TCP). We used the same generators and seeds for simulations with different parameters so as to have always the same input traces (sizes of file transfers and times between transfers).

4.2 The dynamic delayed Ack

We already saw in the previous section that the delayed ACK performs better when the window size is large, since sufficiently packets arrive so that we do not have to wait till the Delayed Ack time interval (of 100ms) expires before generating an ACK.

We therefore introduce a modification of the Sink behavior in TCP that is based on an adaptive version of the delayed ACK, in which the parameter d changes dynamically according to the sequence number of the acknowledged packet; it increases gradually from 1 to 4 as the sequence number increases. More precisely, we define three thresholds: $l1, l2, l3$ such that $d = 1$ for packets with sequence N numbers smaller than $l1$, $d = 2$ for packets with $l1 \leq N < l2$, $d = 3$ for $l2 \leq N < l3$ and $d = 4$ for $l3 \leq N$.

This will then be tested in our framework of short TCP connections. Note that we could expect TCP to perform even better if the decision on the size of d could depend on the window size (and/or whether we are in slow start or in congestion avoidance phase, as was proposed in [2]) rather than on the sequence number. This would then allow to use dynamic delayed Acks also for persistent TCP connections. However, this information is not available at the sink and would require further complications that would either involve extra signaling either at the transport layer or perhaps at a higher layer.

4.3 Simulation results

We examine in this section the influence of different delay ACK schemes on three performance measures of the TCP connections: the throughput, the losses and the average session delay. The comparisons are done for various values of maximum window sizes. We consider the case of 9 and 20 nodes respectively, and depict the simulation results in figures 11-16.

The threshold that we used for the dynamic delayed Ack approach are $\{2, 5, 9\}$ which gave the best results for both throughput and delays and for both $n = 9$ and $n = 20$ nodes. The dynamic Delayed Ack was only tested for maximum window sizes of 10 or more for $n = 9$ and for 15 or more for $n = 20$ (as its goal is to be able to perform with large value of maximum window so as to be useful when we have no knowledge that allows us to choose the optimal value of the maximum window size).

In terms of throughput, we see in Figures 11-12 again that delayed ACK approaches have better performance than the standard TCP: any version of delayed ACK gives better throughput than the standard TCP for maximum window size larger than 7 (for $n = 9$) and larger than 10 (for $n = 20$). For large window sizes, the new dynamic DelAck has the best performance. The

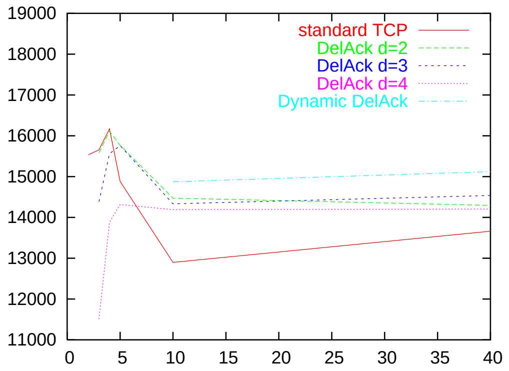

Figure 11: No. of transferred packets (not including retransmission) during 1500sec for $n = 9$ as a function of the maximum window size

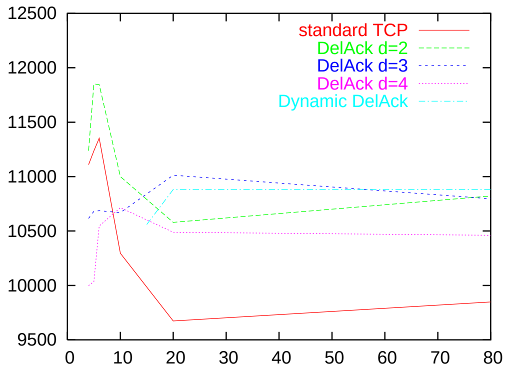

Figure 12: No. of transferred packets (not including retransmission) during 1500sec for $n = 20$ as a function of the maximum window size

standard delayed ACK ($d = 2$) outperforms the standard TCP for any value of maximum window size in the case of $n = 20$, and almost all values of maximum window size in the case of $n = 9$.

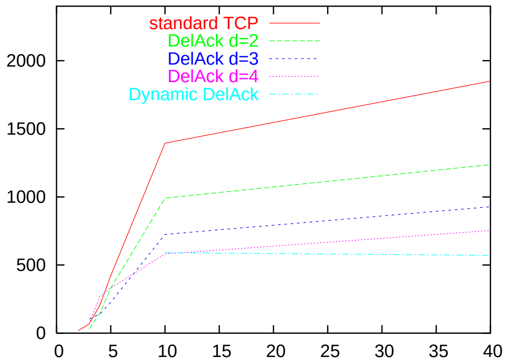

Figure 13: No. of losses of packets during 1500sec for $n = 9$ as a function of the maximum window size

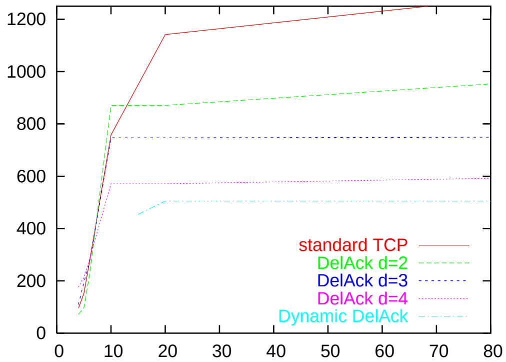

Figure 14: No. of losses of packets during 1500sec for $n = 20$ as a function of the maximum window size

Next, we consider the losses. We see from Figures 13-14 that almost for all values of maximum window, the number of losses is largest for the case of standard TCP, and it *decreases* in the delayed ACK versions as d increases. The smallest number of losses are obtained in our dynamic DelAck scheme.

Next we observe the delay in Figures 15-16. We see that the standard delayed ACK ($d = 2$) outperforms the standard TCP (without delayed ACK) for any value of maximum window for both $n = 9$ and $n = 20$. For large window size, the improvement is of around 18% for $n = 9$ and 22% for $n = 20$. Again, the best performance is obtained by the dynamic DelAck scheme for maximum window size larger than 10 (with $n = 9$) and larger than 20 (for $n = 20$).

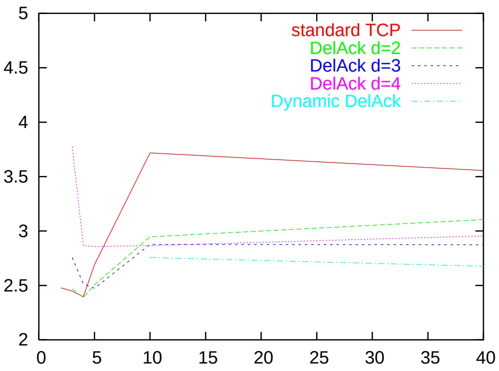

Figure 15: Average session delay for $n = 9$ as a function of the maximum window size

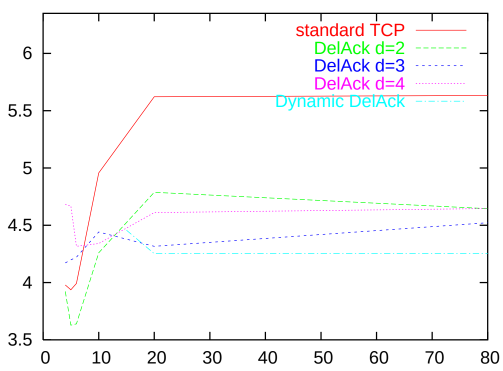

Figure 16: Average session delay for $n = 20$ as a function of the maximum window size

5 Concluding remarks

We have shown in this paper that Delaying Ack (as defined in RFC 1122) improves the performance of TCP over static multi-hop networks that use IEEE802.11 as MAC. Increasing the number packets that are acknowledged by an ACK to $d > 2$ (which is not recommended in RFC 1122) further improves the throughput. The improvements are due to the fact that ACKs and TCP packets contend over the same channel, so decreasing the throughput of ACK can improve the throughput of TCP. When we do not know the number of nodes in the system, we cannot limit the maximum window size to a small value, since this could result in a dramatic decrease of throughput if the number of nodes is large. Large values for maximum windows have to be used, for which the Delayed Ack options with $d > 2$ have been shown to yield an improvement of between 20% to 40%.

Yet the advantage of more ACKs is that they allow to increase rapidly the window size when it is small. Therefore further improvement can be expected if the value of d increases with the window size. Since the window size is not available at the sink, we proposed instead a dynamic DelAck scheme in which the value d increases with the sequence number of the acknowledged packet. We showed that its performances in terms of delay, loss probabilities and throughput outperform those of delayed ACK schemes with fixed d for large values of maximum window.

References

[1] Network Simulator, ns version 2.1, available at http://www.isi.edu/nsnam/ns/

[2] Mark Allman, "On the generation and use of TCP acknowledgements", ACM Computer Communication Review, Oct. 1998.

[3] A. Ahuja, S. Agrawal, J. P. Singh and R. Shorey, *Performance of TCP over different routing protocols in mobile ad-hoc networks*, *Proceedings of IEEE VTC*, Tokyo, Japan, 2000.

[4] V. Anantharaman and R. Sivakumar, "A microscopic analysis of TCP performance over wireless Ad-hoc networks", (extended abstract) *Proc. of ACM Sigmetrics*, 2002.

[5] T. D. Dyer, R. V. Boppana, "A comparison of TCP performance over three routing protocols for mobile Ad Hoc networks", ACM MOBIHOC, 2001.

[6] Z. Fu, P. Zerfos, H. Luo, S. Lu, L. Zhang, M. Gerla, "The impact of multihop wireless channel on TCP throughput and loss", to appear in IEEE INFOCOM 2003. Available on www.cs.ucla.edu/wing/publication/publication.html

[7] G. Holland and N. Vaidya, "Analysis of TCP performance over mobile Ad Hoc networks", *ACM Mobicom*, Seattle, Washington, 1999.

[8] D. B. Johnson, D. A. Maltz, Y.-C. Hu, J. G. Jetcheva, "The Dynamic Source Routing Protocol for Mobile Ad Hoc Networks (DSR)", IETF MANET Working Group INTERNET-DRAFT, available at http://www.ietf.org/internet-drafts/draft-ietf-manet-dsr-07.txt 21 February, 2002.

[9] Y. Joo, V. Ribeiro, A. Feldmann, A. C. Gilbert and W. Willinger, "TCP/IP traffic dynamics and network performance: a lesson in workload modeling, flow control, and trace-driven simulations", Sigcomm Computer Communications Review, vol 31 No. 2, April 2001.

[10] J. Liu and S. Singh, "ATCP: TCP for mobile ad hoc networks", *IEEE JSAC*, **19**(7), 2001.

[11] J. P. Monks, P. Sinha and V. Bharghavan, "Limitations of TCP-ELFN for Ad Hoc networks", MOMUC 2000.

[12] R. de Oliveira and T. Braun, "TCP in wireless mobile ad hoc networks", Tech. Report IAM-02-003, univ. of Bern, Switzerland, July, 2002.

[13] C. E. Perkins and P. Bhagwat, "Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers", *Proceedings of SIGCOMM*, pp. 234-244, 1994.

[14] C. E. Perkins and S. R. Das, "Ad hoc On-Demand Distance Vector (AODV) Routing", IETF MANET Working Group INTERNET-DRAFT, available on http://www.ietf.org/internet-drafts/draft-ietf-manet-aodv-11.txt, 19 June, 2002.

[15] V. Ramarathinam and M. A. Labrador, "Performance analysis of TCP over static ad hoc wireless networks", *Proc of ISCA 15th International Conf on Parallel and Dist. Computer Systems (PDCS)*, 410-415, Sept 2002.

[16] B. Sikdar, S. Kalyanaraman and K. S. Vastola, "An Integrated Model for the Latency and Steady-State Throughput of TCP Connections", *Performance Evaluation*, v.46, no.2-3, pp.139-154, September 2001.

[17] F. Wang and Y. Zhang, "Improving TCP performance over mobile Ad-Hoc networks with out-of-order detection and response", *ACM MOBIHOC*, June 2002.