A Dynamic Adaptive Acknowledgment Strategy for TCP over Multihop Wireless Networks

Ruy de Oliveira and Torsten Braun

Abstract— **Multihop wireless networks based on the IEEE 802.11 MAC protocol are promising for ad hoc networks in small scale today. The 802.11 protocol minimizes the well-known hidden node problem but does not eliminate it completely. Consequently, the end-to-end bandwidth utilization may be quite poor if the involved protocols do not interact smoothly. In particular, the TCP protocol does not manage to obtain efficient bandwidth utilization because its congestion control mechanism is not tailored to such a complex environment. The main problems with TCP in such networks are the excessive amount of both spurious retransmissions and contention between data and acknowledgment (ACK) packets for the transmission medium. In this paper, we propose a dynamic adaptive strategy for minimizing the number of ACK packets in transit and mitigating spurious retransmissions. Using this strategy, the receiver adjusts itself to the wireless channel condition by delaying more ACK packets when the channel is in good condition and less otherwise. Our technique not only improves bandwidth utilization but also reduces power consumption by retransmitting much less than a regular TCP does. Extensive simulation evaluations show that our scheme provides very good enhancements in a variety of scenarios.**

I. Introduction

The Transport Control Protocol (TCP) is the most widely deployed transport protocol in the Internet today. TCP has been successful due to its robustness in reacting dynamically to changing network traffic conditions and providing reliability on an end-to-end basis. This wide acceptance has driven the development of many TCP applications, motivating the extension of this protocol for wireless networks. These networks pose some critical challenges to TCP since it was not originally designed to work in such complex environments, where the level of bit error rate (BER), due to the physical medium, is not negligible. High mobility may further degrade the end-to-end performance, because TCP reduces its transmission rate whenever it perceives a dropped packet.

We do not address mobility related issues in this paper, since it has been investigated in detail in [1]–[5], and also because we target scenarios in which the level of mobility is relatively low (pedestrian movement). The disturbance of such movements should not be too disruptive to the transport protocol. Nonetheless, our technique is expected to work satisfactorily for a moderate level of mobility.

IEEE 802.11 [6] is the standard Medium Access Control (MAC) protocol for ad hoc networks, consisting of both link and physical layer specifications. 802.11 implements a robust link layer retransmission strategy along with the RTS/CTS (request-to-send/clear-to-send) control frames for recovering locally (link level) most of the potentially lost frames. There is also a virtual carrier sense mechanism that is used by every node to announce to all other nodes within a given area when the medium is busy. This mechanism aims at preventing the well-known hidden node problem. 802.11 works efficiently for topologies of at most 3 hops between sender and receiver, which is commonly referred to as a 3-hop scenario [7].

For larger scenarios in terms of number of hops, the hidden node problem still exists due to the spatial reuse property inherent in the propagation model of such wireless networks. Basically, the spatial reuse imposes that within a certain geographical area only one node can transmit at a time [5], [8], [9]. This causes adverse impact on traditional TCP since it is always probing the network for bandwidth by increasing its transmission rate until a lost packet is detected. Hence, unless an efficient coordination between MAC and transport protocols is in place, the end-to-end performance can be severely impaired.

There has been a belief in the research community that TCP can improve its performance by simply ignoring medium induced losses and not slowing down when reacting to those. However, recent research developments on this subject [5], [8], [9] have indicated that this procedure may not be really effective. Rather, this aggressive behavior can adversely degrade the protocol performance by inducing more losses. Actually, the main problem of TCP over 802.11 is the excessive number of medium accesses carried out by TCP. This is caused not only by ACK packets that compete with DATA packets for the medium, but also by the retransmissions performed by TCP when reacting to losses.

This paper presents a dynamic adaptive strategy for decreasing medium contention as much as possible. Specifically, our technique generalizes the concept of delayed acknowledgments recommended by RFC 2581 [10] in which the receiver should only send ACK packets for every other DATA packet received. In our proposal, the receiver may delay up to four ACK packets when the wireless channel is in good condition, and less for lossy channels.

The limit of four packets is imposed by the sender's congestion window ($cwnd$) limit that is also fixed at four packets. This low limit for the sender's $cwnd$ is proper for minimizing collisions and more than enough for scenarios having up to ten hops, which are the target of this paper. By using the strategies just explained, our protocol outperforms a regular TCP whenever the channel is able to provide higher bandwidth and should perform as effectively as a regular TCP does when the wireless channel is heavily constrained.

These simple changes considerably reduce the number of transmissions and retransmissions over the wireless medium. As a result, the overall energy consumption is greatly reduced, which is a key issue for battery-powered devices. Besides, the proposed mechanisms keep the TCP-friendly behavior, which is neglected by many existing proposals. We focus our discussions on short chains of nodes of at most 8 hops, because this is a reasonable limit for today's networks. Nevertheless, the concepts presented here are general and expected to be effective in larger scenarios as well.

The remainder of this paper is organized as follows. The next section describes the main related work on TCP over wireless networks. Section III provides an overview on spatial reuse impacts on TCP. In section IV, we introduce our proposal, where the design decisions are explained and the main features are discussed in detail. Section V presents and discusses the simulation results. Section VI concludes the paper pointing out the main achievements and potential future work.

II. Related Work

TCP performance in wireless environments has been investigated in several studies such as [11]–[14]. Most of them target cellular networks where only the last hop communicates through a wireless link. In this section, we focus on multihop networks in which long chains of wireless links (hops) may exist. For the sake of clarity, we classify the main related work into two main groups, as follows. In the first group, we describe the techniques that deal with packet loss discrimination in wireless environments to prevent the TCP sender from slowing down in case of losses by the wireless medium. In the second group, we address proposals to improve bandwidth utilization by reducing the number of medium access requests, because they are costly in a shared medium.

In the first group, Chandran et al. [1] proposed TCP-feedback, Holland and Vaidya [2] proposed a similar approach based on ELFN (Explicit Link Failure Notification) messages, and Liu and Singh [3] proposed the ATCP protocol. These three proposals are quite similar regarding the way they detect and react to failures within the network. All of them attempt to distinguish the actual packet loss reasons inside the network and not to slow down, if the loss is believed to be due to wireless-related errors rather than congestion. A specific protocol is used to notify the sender when link interruptions due to mobility occur. Fu et al. [4] investigated TCP improvements by using multiple end-to-end metrics instead of a single metric. They claim that a single metric may not provide accurate results in all conditions. They used four metrics: 1) inter-packet delay difference at the receiver, 2) short-term throughput, 3) packet out-of-order delivery ratio, and 4) packet loss ratio. These four metrics are cross checked for accurate detection of the network internal state. Their evaluations focused on mobile scenarios, and channel errors were not fully investigated. Biaz and Vaidya [15] evaluated three schemes for predicting the reason for packet losses inside wireless networks. They applied simple statistics on observed Round-trip Time (RTT) and/or observed throughput of a TCP connection for deciding whether to increase or decrease the TCP congestion window. The general results were discouraging in that none of the evaluated schemes performed really well. Liu et al. [16] proposed an end-to-end technique for distinguishing between packet loss due to congestion from packet loss by a wireless medium. They designed a Hidden Markov Model (HMM) algorithm to perform the mentioned discrimination taking RTT measurements over the end-to-end channel. We proposed in [17] a similar idea to discriminate between congestion and wireless medium induced losses for short chains of nodes in ad hoc networks. The discrimination is performed by a fuzzy logic engine to make the inference robust against the uncertainties inherent in typical RTT measurements.

In the second group, Yuki et al. [18] proposed a technique for combining TCP DATA and ACK packets into a single packet for mitigating the interference problems found in multihop networks. Their evaluations are performed over ideal channels without typical losses. Altman and Jimenez [19] investigated the impact of delaying more than two ACKs on TCP performance in multihop wireless networks. They concluded that in a chain topology of nodes, substantial improvement may be achieved by delaying 3-4 ACKs. In their approach, unless the sender's retransmission timer expires, the receiver always delays 4 packets, except at the session startup. The receiver uses a fixed interval of 100 ms and does not react to packets that are out-of-order or filling in a gap in the receiver's buffer, as opposed to the recommendation of RFC 2581. Their results are positive but obtained for a single flow only. Kherani and Shorey [20] proposed a simple analytical model for TCP over 802.11 based wireless ad hoc networks. In their model, the congestion window limit size defines the number of ACKs to be delayed. They assume no timeout for the delayed ACKs, contrasting the standard delayed ACK algorithm (RFC 2581) in which the delay should not be higher than a given interval (typically 100 ms). Although their model is very simplified, the results are claimed to be quite accurate in predicting TCP throughput. Finally, Allman [21] conducted an extensive evaluation on Delayed Acknowledgment (DA) strategies and presented a variety of mechanisms to improve TCP performance in presence of side effects of delayed ACKs. The results are interesting but tailored to wired networks.

Another important aspect concerning TCP over multihop networks that has been investigated recently, lies in the existing constraints caused by the spatial reuse property of 802.11 [5], [9]. Since this is an essential subject for understanding some of the design decisions behind our approach, we discuss it in detail in the next section.

III. Impact of spatial reuse on TCP over multihop wireless networks

A. Spatial Reuse in Multihop Wireless Networks

In this section, we introduce the spatial reuse property of IEEE 802.11 standard, which has been investigated in detail in [8] and revisited in [5], [9] from a TCP perspective. The

propagation model of 802.11 defines an interference range that is typically slightly higher than twice its transmission range. This means that a given node can only communicate with nodes placed at a maximum distance defined by its transmission range but can interfere with other nodes located at distances up to its interference range.

To prevent the classical hidden node problem, 802.11 includes a four-way RTS/CTS/DATA/ACK exchange, which may be attempted up to generally 7 times in order to effectively recover typical losses by the wireless medium. In addition, 802.11 encompasses a virtual carrier sense mechanism by which every node overhears other nodes' transmission and backs off for a random interval to avoid collisions [6], [7]. These mechanisms are efficient in scenarios with at most 3 hops, as explained in the following.

Fig. 1 [8] depicts a chain topology of 6 nodes. The transmission range is represented by the full line and the interference range by the dashed line. According to the explanation above, each node in Fig. 1 can only transmit to its immediate neighbors within its transmission range but can prevent transmission of the other nodes placed up to 2 hops away from it. From Fig. III-A, it is clear that the transmission of nodes 2 and 3 will prevent node 1 from transmitting since node 1 is within their transmission and interference ranges, respectively.

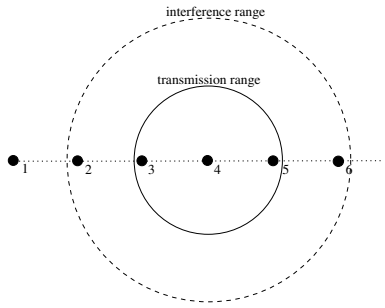


Fig. 1. Spatial reuse in multihop networks

Furthermore, the transmission of node 4 will prevent node 2 from sending a response (CTS) to node 1, thereby impacting node 1 transmission as well. Thus, node 1 will only succeed in its transmission when node 4 is done with its transmission to node 5. Hence, in a group of 4 subsequent hops, only one can be active at a time, which imposes a maximum channel utilization of 1/4 relative to the 1-hop capacity.

This analysis considers an ideal MAC protocol that can schedule the nodes transmission evenly. Nevertheless, 802.11 does not work so smoothly, and because of that if the sender at the beginning of the chain is not able to pace its transmission rate to match the channel capacity, the channel utilization will drop. In other words, the factor of 1/4 can only be achieved if the sender is able to inject into the network the exact amount of data that the channel can forward [8], [22].

B. The Optimal TCP Congestion Window Limit

The spatial reuse discussed above suggests that a TCP sender should limit the size of its congestion window ($cwnd$) in order to achieve better performance. The $cwnd$ defines the maximum number of DATA packets a TCP sender may inject into the network at once without waiting for an ACK packet from the receiver. Thus, according to the discussion above, a limit of $h/4$ for $cwnd$ is presumably an optimal setting, where h is the number of hops in the chain topology. This means that long chains with many groups of consecutive 4 hops will have higher limit for $cwnd$ than short chains. Interestingly, a 4-hop chain will have an ideal limit of 1 packet for the TCP's $cwnd$.

In [9] the authors found that a limit of 3 packets for $cwnd$ provided best channel utilization in a 7-hop chain topology, which is slightly higher than the theoretical limit explained above. Their simulation results showed that better match to the $h/4$ factor is achieved for long chains of nodes, where better distribution of the packets in flight among the nodes occurs. A more extensive investigation into this problem was performed in [5], where the authors propose limiting the $cwnd$ according to the round number of hops in place.

By either investigation above, a short chain of nodes with up to 10 hops, should have a $cwnd$ limit of 3 to 4 packets. We have confirmed such results by simulation (with slight variations) as depicted in Fig. 2. These simulations were conducted using the ns2 simulator with default settings apart from the packet size that was set to 1460 bytes.

Fig. 2 allows us to draw very important conclusions. First, for short chains of nodes of with at most 3 hops, even a very small $cwnd$ of 1 or 2 packets is sufficient to guarantee maximum throughput. For scenarios with up to 10 hops, which might be likely in many scenarios, a $cwnd$ limit of 3 packets suffices.

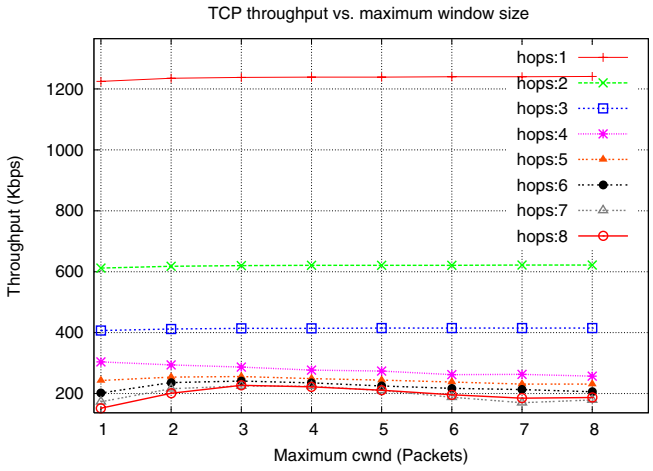


Fig. 2. The optimal limit for the sender congestion window

In contrast to some related work, such results indicate that, at least for short chains of nodes, it is does not help much to avoid decrease in $cwnd$. The problem is that $cwnd$ should be set to very small sizes anyway, and in such cases the difference in throughput for different limits of $cwnd$ is not that much, as shown in Fig. 2.

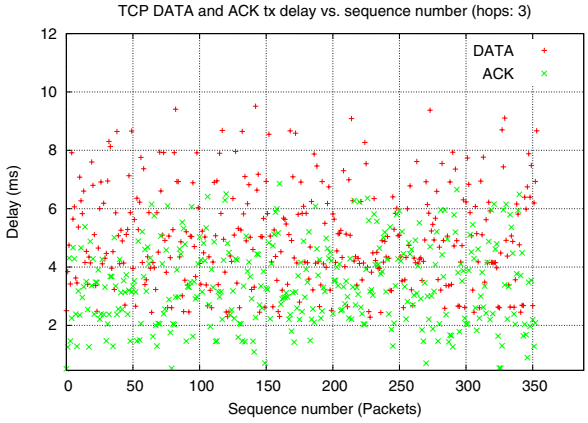
IV. Dynamic Adaptive Acknowledgment

In this section, we present the main features of our proposed approach, which we call TCP-DAA (DAA: Dynamic Adaptive Acknowledgment). As earlier mentioned, TCP-DAA targets feasible scenarios in which the IEEE 802.11 standard may provide acceptable performance. Recent investigations on 802.11 have shown that such a protocol is effective in recovering most of the wireless induced losses in typical scenarios, but it does not scale as the number of wireless hops increase due to the spatial reuse property discussed in the last section. There are a number of important applications and scenarios in which the number of hops involved will be far below 10 hops, and the number of nodes will normally not exceed 100 nodes. Typical examples include ad hoc networks in classrooms, meeting and workshop spots, working offices, and wifi in home buildings, among many others.

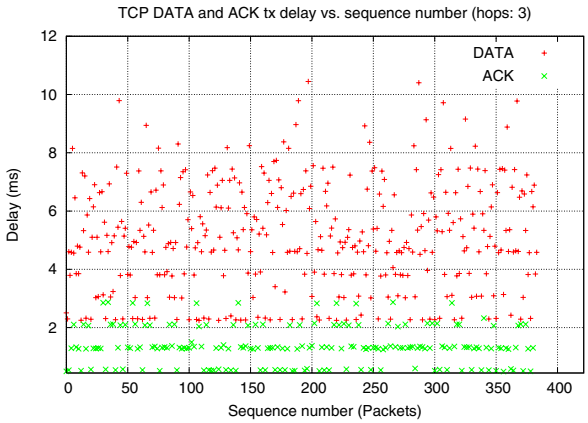
TCP-DAA takes advantage of the fact that the 802.11 protocol schedules the contending transmission requests in such a way that normally every node cannot transmit more than one frame at a time, but must contend for the medium again in order to prevent capture of the medium by a single node. This feature allows a TCP source to send more DATA at once without receiving ACKs, as the MAC layer scheduling prevents potential bursts. However, if the receiver acknowledges every incoming DATA packet, then the probability of collisions between DATA and ACK packets increases considerably. Moreover, since the receiver must also contend for the medium by using CTS/RTS control frames, the overall overhead at the MAC layer, for transmitting ACKs, is not negligible.

These problems may be mitigated if the sender merges several acknowledgments into a single ACK, what is possible due to the cumulative ACK scheme used in the TCP congestion control. Fig. 3 shows the end-to-end transmission delay experienced by both DATA and ACK packets in a chain topology of 3 hops for a simulation run that lasts 10 seconds. It is easy to see that in a configuration without delayed acknowledgments (Fig. 3(a)), the delays experienced by ACK packets are in the same order of magnitude of the DATA packets' delay.

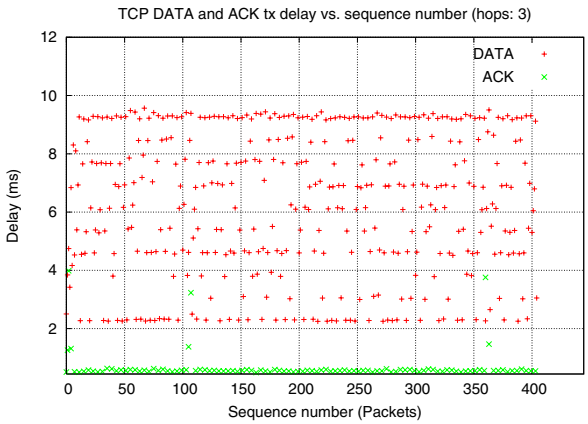
Using the standard DA of RFC 2581, such delays drop significantly, and with TCP-DAA the improvement is even higher. Taking the average delay for each run, from top to bottom in Fig. 3, the ACK delay accounted for 40%, 21% and 9,6% of the total transmission time, respectively. This shows how costly an ACK may be in a shared medium, where any transmission requires a medium access procedure. In other words, the less ACKs the more time for DATA transmission. Note that in Fig. 3(c), the ACK packets with sequence number in the neighborhood of 105 and 360 experienced much higher delay than the average. Such a phenomenon is triggered by spurious timeout at the receiver. We will discuss this issue further below. The fewer amount of ACKs for the sender might lead TCP to low performance in typical wired scenarios where the congestion window ($cwnd$) limit is usually high. This might happen because a TCP sender may only enlarge its



(a) No delayed ACK



(b) Standard DA



(c) TCP-DAA

Fig. 3. DATA and ACK transmission delay in a typical wireless channel

$cwnd$, toward the limit, upon receipt of ACKs. This problem is not so critical in our technique, however, as the $cwnd$ limit in place (4 packets) is rather low. This means that after a reduction of $cwnd$ due to a lost packet, it will quickly reach the limit again upon receiving a few ACKs. Nevertheless, some improvement seems to be possible if the sender increases its $cwnd$ in such a way that compensates the fewer number of ACKs received, as investigated in [21]. This issue is left for future work.

By delaying the acknowledgment notification to the sender, the receiver may trigger a retransmission by timeout at the sender if the receiver delays excessively. Thus, the receiver has to be well adjusted in order to avoid such spurious retransmissions. The standard delayed acknowledgment (DA) proposed in RFC 2581, recommends that a receiver should send one ACK for every other DATA packet received, and should not delay an ACK when either an out-of-order packet or a packet filling in a gap in the receiver's buffer is received. Besides, the maximum delay should not exceed a given time interval (typically 100 ms).

These concepts in the standard DA work reasonably well in wireless environments, as shown in [23], but higher enhancements are possible by delaying more than 2 ACKs, as shown in [19]. In the latter, the authors observed that in a chain topology of nodes, substantial improvement may be achieved by delaying 3-4 ACKs. In their approach they simply delay 4 packets (except at startup) or wait for a timeout expiration, which has also a fixed interval, for sending an ACK to the sender. They also considered a large limit of 10 packets for the sender's $cwnd$ to keep several packets in transit. We will hereafter call this approach LDA (Large Delayed Acknowledgment).

The main problem with both the standard DA and the LDA scheme is the fixed timeout interval (100 ms), since the packet inter-arrival at the receiver changes not only with the channel data rate, but also with the intensity of traffic going through the network.

TCP-DAA combines the idea of higher number of delayed ACKs with the dynamic reaction proposed in RFC 2581, i.e, reaction to packets that are either out-of-order or filling in a gap. Furthermore, our protocol adjusts itself to the channel conditions, in that it adaptively computes the timeout interval for the receiver on the basis of the packet inter-arrival time. In this way, the receiver delays just enough to avoid spurious retransmissions by the sender and is able to adapt itself to different levels of delays imposed by the wireless channel, thereby being independent of both channel data rate and number of concurrent flows crossing the network. As we will show in section V, TCP-DAA outperforms the two schemes above (standard DA and LDA) in several scenarios.

A. Algorithm

As mentioned above, a smooth interaction between sender and receiver may provide optimal performance in our approach. We are currently investigating the sender side with respect to response to losses and timeout interval computation since both are affected by the delayed ACKs. The technique we used for minimizing unnecessary retransmissions by timeout consists of two adjustments: 1) the number of duplicate ACKs for triggering a retransmission by the fast retransmit mechanism is decreased from 3 to 2 packets, which is in line with [24] in the sense that we work with a small $cwnd$ limit; 2) the regular retransmission timeout interval is increased fivefold for compensating the maximum of four delayed ACKs. These are the only two changes performed on the regular TCP sender, which proved to be effective in most of our evaluations.

Fig. 4 depicts the timing diagram of TCP-DAA under normal conditions, i.e., after startup and without any loss. For every four DATA packets transmitted, the receiver replies with an ACK. Whenever a given ACK $_{i,i+1,i+2...}$ is to be delayed, an associated timer is started (t_i), or restarted (t_{i+1}, t_{i+2}) if there is one already running. The receiver also measures the DATA packet inter-arrival gap between the packets for which the ACK is to be delayed ($\delta_{i,i+1,i+2...}$).

The receiver keeps track of the number of ACKs delayed by maintaining an ack_count variable which increases from 1 to the current value of its delaying window ($dwin$). By checking the value of ack_count, the receiver is able to determine whether the received packet is the first one from the group that is going to have their acknowledgment delayed.

In case such a packet is the first one, the inter-arrival interval between the last received packet and the current one is not taken. This is needed to avoid that unnecessary intervals such as the one between δ_{i+3} and δ_{i+4} in Fig. 4 are improperly considered for the timeout interval computation. By using this strategy, we assure that in normal conditions, the inter-arrival measurements will reflect very closely the gap between the received DATA packets triggering delayed ACKs. Note that under packet loss, the receiver will not need such measurements as it will not delay out-of-order packets. Rather, it will await until it receives in-order packets again.

Similarly to what the TCP sender does, the receiver uses a low-pass filter to smooth the packet inter-arrival intervals. Upon arrival of a given DATA packet p_{i+1}, it calculates the smoothed packet inter-arrival as indicated in (1), where $\overline{\delta}_i$ refers to the last calculated value, δ_{i+1} is the packet inter-arrival sampled, and α is the inter-arrival smoothing factor, with $0 < \alpha < 1$.

$$\overline{\delta}_{i+1} = \alpha * \overline{\delta}_i + (1 - \alpha) * \delta_{i+1} \quad (1)$$

The value computed from (1) is used to set the timeout interval at the receiver. In our design, we established that after the receipt of a DATA packet that causes an ACK to be delayed, it is reasonable to wait for at least the time the second next packet is expected. The rationale here is that the delay variations are relatively high in such environments and in case of a single dropped packet, the next DATA packet will arrive out-of-order, which will trigger immediate transmission of an ACK, as recommended in RFC 2581. However, if it was only a delay variation, and the DATA packet arrives before the expected time for the subsequent packet, no timeout

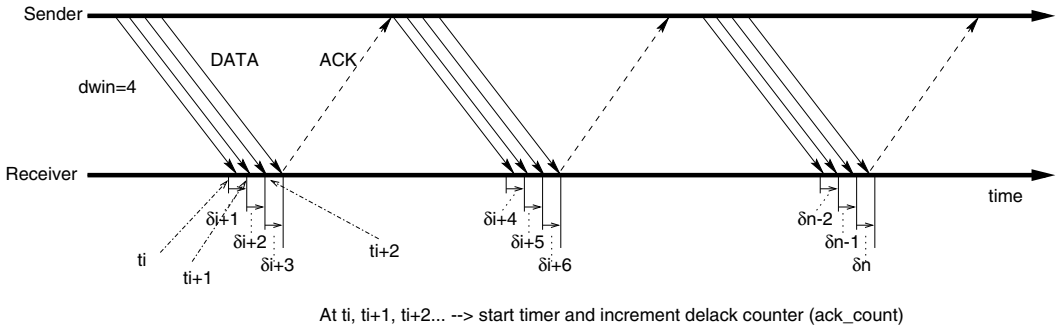


Fig. 4. Timing diagram of TCP-DAA

is triggered and the receiver avoids sending an extra and unnecessary ACK packet into the network.

Hence, we use a timeout interval T_{i+1} as shown in (2). Note that the factor 2 in (2) refers to the estimated time for the second expected DATA packet to arrive. This equation also includes a timeout tolerance factor κ, with $\kappa > 0$, defining how tolerant the receiver may be in deferring its transmission beyond the second expected DATA packet. In short, the effective timeout interval T_i is at least twice the the smoothed value $\overline{\delta}_i$ and may be higher depending on the value of κ.

$$T_i = (2 + \kappa) * \overline{\delta}_i \tag{2}$$

Fig. 5 illustrates how the receiver dynamic window ($dwin$) changes. Under normal conditions, it is maintained at maximum size of four packets. Whenever the receiver gets a packet that is either out-of-order or filling in a gap in the receiver's buffer, or when its timer expires, it sends immediately an ACK to the sender and reduces $dwin$ to the size of two packets. Another design option here would be to restart the $dwin$ from size one. We chose to cut it down to the size of two to make our technique to work in such a condition, just after a response to a lost packet, the same as the regular DA, except for the timeout interval that is adaptive in our case.

In our algorithm, subsequent DATA packets will trigger $dwin$ growth toward the maximum size again, provided they are not in one of the three conditions above. Using this dynamic behavior, the receiver prevents the sender from missing ACKs when packet losses occur. As mentioned above, the LDA proposal [19] works with a fixed $dwin$'s size of four packets (except at startup), and uses a large $cwnd$ limit at the sender to keep the channel full of DATA packets in flight. While this procedure may prevent a lack of ACKs at the sender, it may also induce excessive number of retransmissions at the sender, as shown in section V-B.

The $dwin$ growth (additive increase) is governed by (3) which shows that such an increase may be fixed at one (packet) or variable according to the startup speed factor μ, with $0 < \mu < 1$. The reason for this factor is that during the startup phase (initial of the session), the sender starts with a window size of two packets and then increases it by one at every ACK received. Although $dwin$ size is initialized with

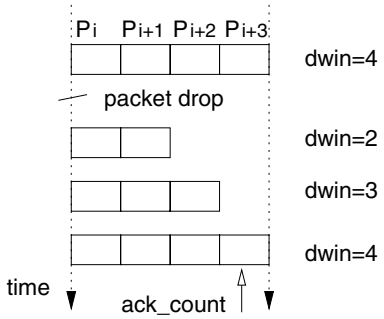


Fig. 5. Receiver dynamic window for delaying packets

one, if it started from startup being increased at the rate of one packet, then there would happen a shortage of ACKs at the sender which would not be able to send packets into the network before its retransmission timer had expired.

Thus, the threshold $maxdwin$ is used to define the instant the startup phase is over, which occurs when $maxdwin$ first reaches its maximum value and becomes $false$. From our evaluations, we noticed that by properly setting the μ parameter, our algorithm achieved better performance for short-term transmissions, as shown in section V-E

$$dwin = \begin{cases} dwin + \mu, & \text{if } maxdwin = false \\ dwin + 1, & \text{otherwise} \end{cases} \tag{3}$$

The mechanisms just explained make TCP-DAA effective because whenever it is possible it uses the scarce channel bandwidth efficiently, and when the channel is facing really poor conditions, it performs in general as effective as the regular TCP. Using its dynamic adaptive window, TCP-DAA somehow probes the network for resource availability, since it will always delay more packets (up to 4) when the network so permits. Furthermore, our protocol is TCP friendly as it preserves the basics of TCP-friendly behavior that is to decrease transmission rate under loss detection and to increase it if the network so allows.

B. Reaction to Packet Losses

In order to fix the concepts explained above, we show here a typical response of our mechanism when reacting to lost packets. We include the response of the LDA scheme

to highlight the difference of our proposal to that one. Fig. 6 exhibits part of a simulation run in which both strategies faced a dropped packed in a chain topology of 5 hops. Throughout this paper, we use the TCP Newreno flavor [25] as the regular TCP.

Let $packetn$ be the DATA packet of sequence number (n). Fig. 6(a) shows that the sender transmits four packets (320-323) at time 12.6 seconds. In this run, $packet322$ and $packet323$ are dropped, and so the receiver times out and acknowledges only two packets (320, 321) instead of four. The receiver also updates its $dwin$ size to two. Upon receipt of the ACK for $packet321$, the sender sends two new packets (324, 325) because two sent packets were acknowledged. At this moment there are only 2 packets in flight (324, 325). Since $packet324$ and $packet325$ are detected by the receiver as out-of-order packets, they trigger immediate acknowledgments at the receiver. By receiving the first duplicate ACK, the sender transmits a new packet (326) which will also be out-of-order.

When the sender receives the second duplicate ACK at instant 12.9 seconds, it retransmits the first lost packet (322), and halves its $cwnd$ size to two packets (fast retransmit/fast recovery). The $cwnd$ will be expanded gradually after the sender exits the fast recovery phase. When the sender receives the third duplicate ACK, at time 12.96 seconds, it does nothing because it is in the fast recovery phase. At the instant 12.97 seconds, the sender gets the acknowledgment for $packet323$, retransmits this packet and then exits the fast recovery procedure. $Packet323$ fills in the gap at the receiver's buffer, which triggers the ACK of $packet326$ due to the cumulative property of TCP acknowledgment strategy.

At instant 13.01 seconds, the sender receives the acknowledgment for $packet326$, and so transmits two new packets (327, 328). These two packets cause the receiver to send one ACK only as its $dwin$ is set to two packets at this point. After that, $dwin$ increases and, as a consequence, the number of delayed ACKs increase toward 4. Fig. 6(a) shows two spurious retransmissions by timeout at the receiver. $Packet339$ and $packet338$ are unnecessarily acknowledged at the instants 13.62 s and 13.79 s, respectively. This means that the timeout interval computation may still be improved, but that is left for future work.

Fig. 6(b) shows the response of LDA to a packet loss. In this simulation run, $packet241$ is lost at about 10.55 seconds. Differently from our technique in which the amount of packets in flight are limited to four packets, the proposed LDA works with a large limit for the $cwnd$ (10 packets), so it has more packets in flight than TCP-DAA. One can notice in Fig. 6(b) that although only one packet has been dropped, various acknowledgments triggered the transmission of less than the optimal four packets by the receiver. This shows that the retransmission timer expired in several situations unnecessarily.

Additionally, the sender waits for the default three duplicate ACKs for retransmitting the dropped packet, which incurs in a longer time for it to take action. In short, by comparing Fig. 6(a) with Fig. 6(b), one can clearly see that TCP-DAA provides more stability regarding the number of delayed ACKs. As a result, less packet delay variation is perceived by the sender, which in turn tends to minimize the inaccuracy in the timeout interval computation at the sender.

V. Performance Evaluations

Here we evaluate and compare the performance of TCP-DAA with the other TCP flavors and with the LDA proposal presented in [19]. We compare our work with LDA because it also investigates a delayed acknowledgments strategy for improving TCP performance in multihop networks. We also compare our results with other TCP flavors in their theoretical best conditions, so as to make sure that our proposal is indeed efficient among a wide range of options. Hence, we simulate the other TCP flavors including two potential improvements: the standard delayed acknowledgment (DA), and a low limit for their $cwnd$ (3 packets), for the sake of efficient bandwidth utilization, as explained in section III-A.

A. Simulation Scenario

We used the ns2 simulator in our evaluations of the two scenarios depicted in Fig. 7 in which we have a single chain topology and a grid topology with 9 (maximum) and 25 nodes, respectively. In both topologies, each node is 200 meters away from its closest neighbors and the wireless data rate is 2 Mbps. In the simulator, the effective transmission range is 250 meters while the interference range is 550 meters, in accordance with 802.11. When applicable, the throughput r is calculated as $r = \frac{seq*8}{stime}$, where seq is the maximum sequence number (in bytes) transmitted and acknowledged and $stime$ is the simulated time.

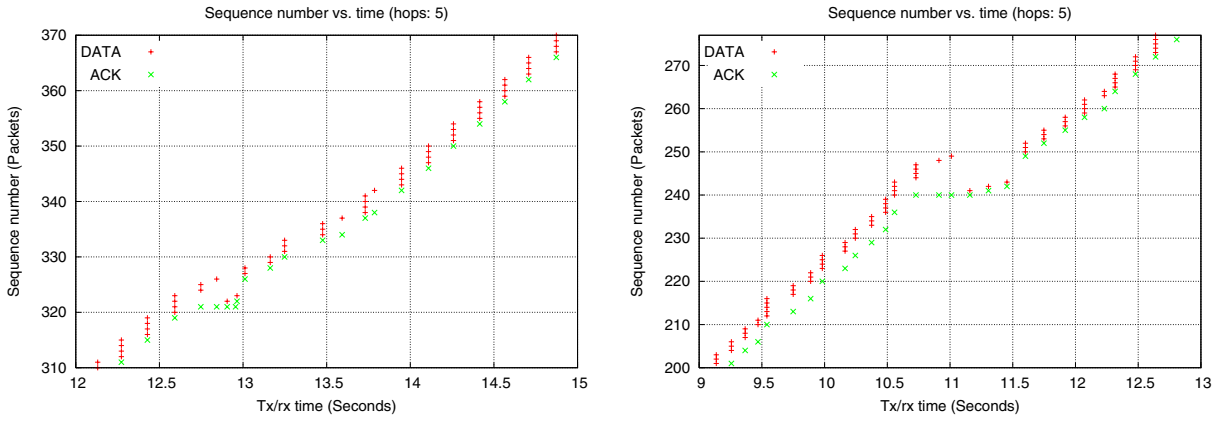
Unless otherwise stated, the parameters settings are as follows. Packet size is 1460 bytes, the window limit (WL) for the other flavors (except for the regular TCP) is 3 packets, the regular TCP is the Newreno flavor; and for TCP-DAA, α is 0.75, κ is 0.2, μ is 0.3, and the initial timeout interval is 200 ms. The other parameters use the default values of the simulator and, if not explicitly mentioned, the simulation runs last 300 seconds.

B. Retransmissions

Since TCP-DAA is more tolerant to packet delay variations by having the regular RTO (retransmission timeout) at the sender multiplied by a factor of 5 and a dynamic acknowledgment strategy at the receiver, it is expected that it minimizes spurious retransmissions by timeout.

Fig. 8 shows the result of a simulation run in which 10 flows share the medium in the chain topology of Fig. 7(a), under different number of hops. The figure exhibits the aggregate number of retransmissions including retransmissions by both timeout and the fast retransmit mechanism.

It can be seen that for the 2-hop scenario, our protocol does not retransmit any packet at all, while the regular TCP retransmits over 400 times. For 4, 6 and 8 hops, our mechanism provides significant enhancement by retransmitting much less than the other algorithms. This is the result of



(a) TCP-DAA

(b) LDA

Fig. 6. Delayed acknowledgment strategies

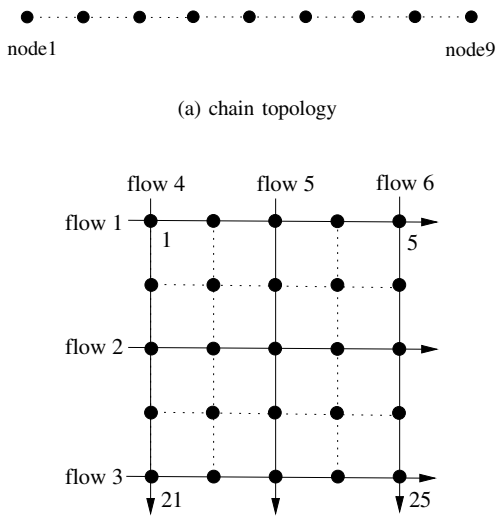
less collisions between DATA and ACK packets, given the fewer amount of ACKs generated by our mechanism. The conservative mechanism for the retransmissions by timeout in the TCP-DAA's sender contributes to such achievements as well. In these evaluations, TCP-DAA retransmits 72 to 100% less than the regular TCP and 65 to 70% less than the LDA scheme. This outcome is doubtlessly expressive in terms of energy consumption benefits [22].

C. Throughput in the Chain Topology

In this section, we investigate the bandwidth utilization over a wide range of situations. While most related work present results over either a single flow or a fixed number of hops, we present here our evaluation not only for different number of hops but also for different number of concurrent flows in the chain topology of Fig. 7(a). In addition, we compare our results with the main existing TCP flavors. It would be reasonable only to compare our algorithm with the other flavors, including the regular TCP, with DA enabled. However, as the vast majority of related work present results over the regular TCP without DA, we also evaluate this configuration here to allow better comparison with related work.

Fig. 9 exhibits a remarkable achievement of TCP-DAA. These results are obtained by taking the average of 5 runs. TCP-DAA outperforms all the other algorithms in most situations. Only TCP Vegas and regular TCP, both with optimal setting (DA+WL), outperform slightly our mechanism in the scenario with 7 hops and 2 concurrent flows. Nevertheless, as the number of flows increase, the performance of both strategies degrades significantly, while it is not the case in our scheme. We believe that TCP-DAA will be further improved if the default sender's RTO calculation is fine tuned to its strategy.

It is interesting to note that, in general, the more flows the better the improvement of our algorithm over the other protocols. One reason for that is the high level of transmission delays due to the higher number of flows in the network. Under such high delays, the packet delay variance



(a) chain topology

(b) grid topology

Fig. 7. Simulated scenarios

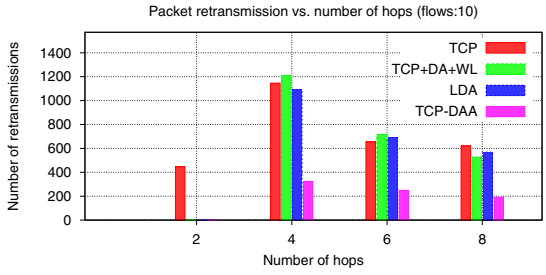
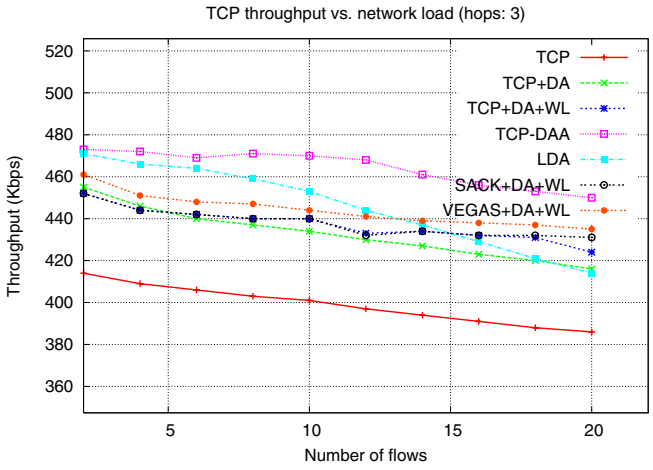
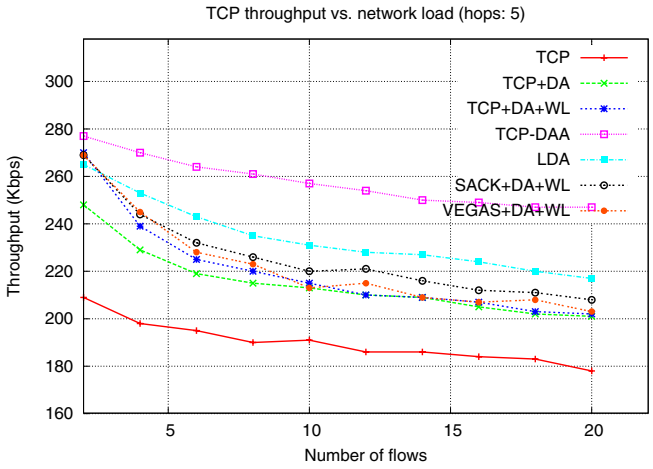


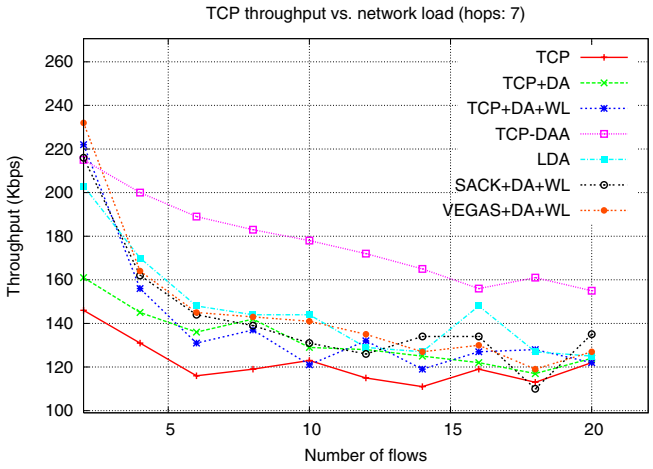
Fig. 8. Aggregate number of retransmissions



(a) 3-hop



(b) 5-hop



(c) 7-hop

Fig. 9. Aggregate Throughput in the Chain Topology

becomes less significant in the RTO calculation, and so less interference of the delayed ACKs is perceived by the sender. Another reason lies in the sender's high tolerance to invoke the timeout procedure, which renders the TCP-DAA's sender less aggressive than a regular sender. As shown in the prior section, this behavior is advantageous with regard to spurious retransmissions, resulting in more bandwidth to the concurrent flows. In case there is no concurrent flow to use the left bandwidth, while the sender is waiting for the timeout, then that bandwidth is simply wasted.

Overall, the observed improvements are up to about 50% over regular TCP, and over LDA improvements of up to 30% are obtained. We also performed simulations for 2, 4 and 6-hop scenarios and the results are similar, in some cases less improvement are observed, but in most cases our algorithm performs significantly better than all the others.

D. Throughput in the Grid Topology

Here we describe the investigations carried out in a more complex scenario, the grid topology illustrated in Fig. 7(b). In these evaluations, we have first only 3 flows crossing the topology horizontally (flows 1, 2 and 3 in Fig. 7(b)). In the next step, 6 flows (3 horizontal and 3 vertical) are injected into the network concurrently. The results, averaged over 5 runs, are depicted in Fig. 10.

This is a critical scenario, given the various interactions among the nodes in place. The level of dropped packets is high, and so is the degradation of our mechanism. As the scheduling strategy of 802.11 is inherently unfair, it may happen that in some circumstances TCP-DAA outperforms the other implementations [22], but its overall performance is expected to be similar to that of a regular, as illustrated in Fig. 10.

In these simulations, our mechanism performs roughly the same as the other implementations for the run with only horizontal flows (3 flows). Its efficiency deteriorates for the 6 flows case, going down to the level of the regular TCP with DA. Note that the window limit (4 packets) of TCP-DAA is slightly higher than the one of the regular TCP (3 packets), which may explain why TCP-DAA does not reach the performance of the TCP+DA+WL configuration. We have not used the same window limit in both algorithms because the limit of 3 packets was expected to perform better for the regular TCP, as shown in section III-B. TCP SACK and Vegas perform best in these evaluations. Our algorithm would most likely follow SACK and Vegas's performance closely if it had been implemented over these flavors, but it was implemented over TCP Newreno which performs well in a variety of scenarios. We believe that a more robust MAC protocol, concerning fairness, may favor our mechanism in such environments.

E. Short-lived Flows

An important aspect of approaches that defer acknowledgments at the receiver is their response to short-lived flows. The duration of such flows are usually not sufficient to lead

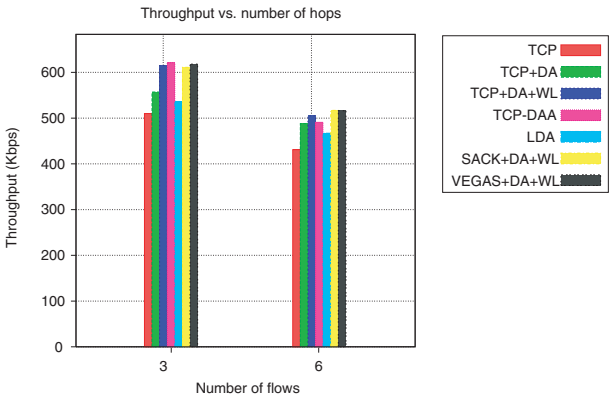


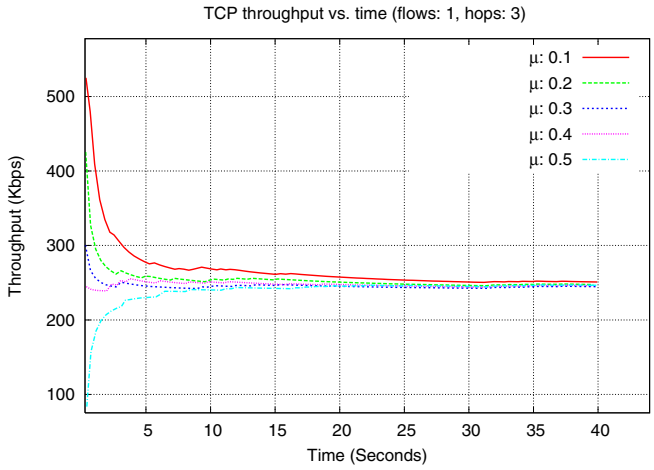
Fig. 10. Aggregate throughput in the grid topology with cross traffic

the congestion control mechanism rapidly to an equilibrium. The problem is that TCP is an ACK-clocked mechanism as its sender needs ACKs to clock out DATA. TCP-DAA addresses this issue by using the startup speed factor μ which defines how aggressive the sender may be during its startup. In this section, we evaluate the performance of our mechanism for short-lived flows.

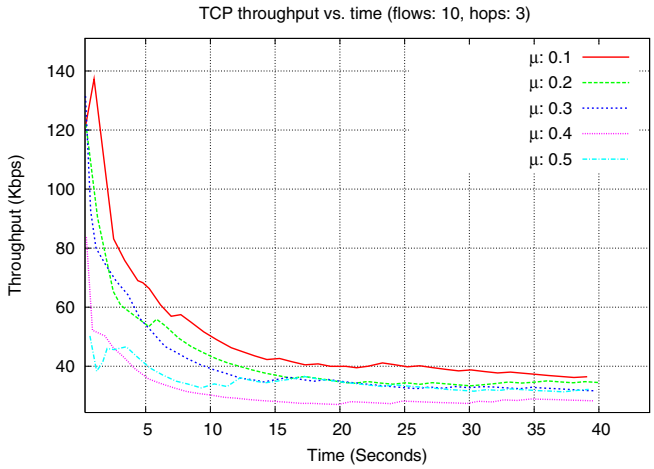
For the sake of clarity, we consider a scenario without the hidden node problem. Fig. 11 illustrates the bandwidth achieved for a run over the chain topology of Fig. 7(a). In this run, sender and receiver communicate over 3 hops, and different values of μ are simulated for a duration of 40 seconds. The curves plotted in the figure represent the bandwidth achieved by a single flow. The graphs are computed with a granularity of 0.3 seconds, and accumulated over the whole simulated interval, i.e., gradually accumulated from 0 to 40 seconds by steps of 0.3 seconds.

The results point out that smaller values of μ render better performance at startup than large values. As the session duration gets longer, the difference among the distinct values of μ become negligible, as shown clearly in Fig. 11(a). The same convergence would be noticed in Fig. 11(b), where 10 flows share the medium, if a longer simulation duration had been performed. The main outcome here is that short-lived connections should use low values for μ toward faster transfers.

In order to guarantee that the receiver does not lead the sender to miss ACKs at startup, the value of μ should then be in the range (0,0.5), i.e., $0 < \mu < 0.5$. Having μ=0.5 causes the receiver to increase its $dwin$ too quickly as follows. The sender first transmits two DATA packets (sender's initial window is set to 2). The receiver defers the acknowledgment of the first DATA packet because its $dwin$ (initialized with 1) is first increased by 0.5 (μ value) and so becomes 1.5 which is greater than the current ack_count that is equal to 1. By receiving the next DATA packet, $dwin$ reaches size 2, and so does ack_count. As ack_count is then as large as $dwin$, the receiver acknowledges the second DATA packet. When the sender gets such an ACK, it expands its $cwnd$ to 3, and transmits three new packets. These packets are going



(a) 1 Flow



(b) 10 Concurrent flows

Fig. 11. Accumulated Throughput of 1 flow for Short-lived Flows

to be delayed at the receiver, but they are not enough to make ack_count as large as $dwin$. The three packets drive ack_count to 3, and $dwin$ to 3.5.

As a result, the receiver has to wait for its timer expiration before sending the buffered ACK, and reducing $dwin$ to 2. We use a conservative initial timeout interval of 200 ms for avoiding spurious timeout during startup, primarily for congested scenarios. Hence, if TCP-DAA increases its $dwin$ too fast, it may not wait long enough for the transmission channel to be full of packets for delaying the ACKs. This may, in the worst case, trigger a retransmission by timeout at the sender. Fig. 11(a) illustrates how the curve for μ=0.5 performs much poorly than the others at the initial instants of the simulation time. Higher values of μ experience the same problem.

Since the μ value is not very significant for long-lived flows, we simply used a value of 0.3 in the previous evaluations for providing some tolerance in case of dropped packets at the very beginning of the simulation. Smaller values may not be appropriate for scenarios with many concurrent flows, because the smaller the μ the more packet exchanges, which may induce excessive drops during startup.

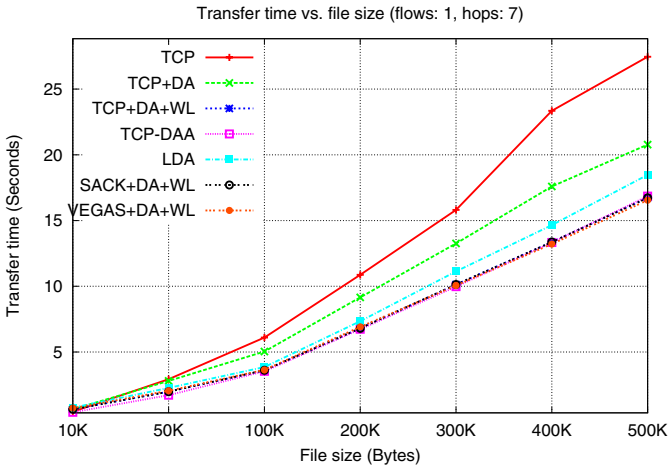
To further illustrate the performance of our algorithm for short-lived flows, we conducted simulations in which only short files are transferred over the chain topology. The results, averaged over 5 runs, are depicted in Fig. 12. In the evaluation shown in Fig. 12(b), a single flow crosses seven hops, the file size ranges from 10 to 500 Kbytes, and the startup speed factor μ is set to 0.1. Fig. 7(a) highlights that except for the regular TCP (with or without DA) and the LDA scheme, all the other implementations perform roughly the same. Short-lived flows indeed do not seem to benefit from our proposed mechanism. If the scenario is too much congested by many parallel flows, TCP-DAA may experience worst performance than the other flavors, given the lack of time it faces to reach equilibrium. This is shown in Fig. 12(b) for five concurrent flows.

The results confirm that by being too conservative, TCP-DAA does not perform well for very short file sizes as it does not speed up their transfer time. Such a time, in Fig. 12(b), refers to the time needed to fully transfer the slowest flow among the five concurrent ones. In these simulations, TCP-DAA does not provide any measurable benefit for packet sizes of 10, 50 and 100 Kbytes. In fact, it performs slightly worse than the other flavors for the very short packet size of 10 Kbytes. On the other side, for packets of size bigger than 100 Kbytes, it may provide some benefit. The enhancements are not very significant, though. In short, to be optimal for short-lived flows, our mechanism needs a more aggressive strategy at the sender. However, there will be always a tradeoff between short and long-lived flows optimizations.

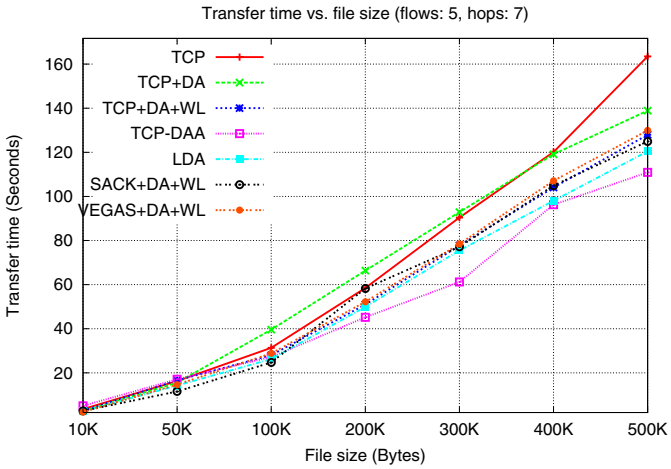
F. Discussions

In general the results were quite positive in the sense that TCP-DAA outperformed the other algorithms in most cases. Using our technique, a long-lived TCP connection may improve performance when the wireless channel is facing reasonable conditions and should perform as effective as a regular TCP does otherwise. The evaluation results went further to show that even under a certain level of loss rate, our mechanism can still provide enhancements. Furthermore, TCP-DAA provides not only bandwidth utilization enhancements but also energy consumption benefits.

Nonetheless, there is surely room for improvements. First, the end-to-end throughput enhancements were in general not very significant for scenarios with low number of concurrent flows. This may be changed by fine tuning the algorithm used by sender and receiver for computing the timeout intervals involved. Second, the performance of short-lived flows may also be eventually enhanced by appropriate timeout interval calculations at the end nodes. Third, the sender side may increase the $cwnd$ smartly to compensate the lack of ACKs due to the delayed strategy. Fourth, the strategy used for controlling the delaying window ($dwin$) at the receiver may be tailored for highly noisy environments. Fifth, it seems that better results may be achieved for grid topologies if the MAC layer's fairness is improved.



(a) 1 Flow



(b) 5 Concurrent flows

Fig. 12. Transfer time for short-lived flows

VI. Conclusions

We have introduced and evaluated our algorithm for improving TCP performance over multihop wireless networks. The key idea of our dynamic adaptive acknowledgment strategy is to provide capability to a TCP receiver to adjust itself, according to the channel conditions, in terms of the DATA to ACK ratio. Our current mechanism is tailored to networks comprising at most ten hops and facing moderate level of bit error rate.

The outcome of the simulation evaluations showed that our algorithm outperforms not only the regular TCP and the main TCP flavors, but also similar techniques that have been proposed in the literature, in a variety of conditions. Our scheme improves throughput and energy consumption, which are two key issues in such environments. Finally, it is easy to deploy since the changes are limited to the end nodes only.

We do not claim that our technique is the optimal acknowledgment strategy for a TCP implementation in multihop networks, but the achieved results are indeed encouraging, justifying further investigation on this direction. In particular, robustness to highly noisy scenarios is very much of interest for future work.

ACKNOWLEDGMENT

The work presented in this paper was supported (in part) by the National Competence Center in Research on Mobile Information and Communication Systems (NCCR-MICS), a center supported by the Swiss National Science Foundation under grant number 5005-67322.

REFERENCES

[1] K. Chandran, S. Raghunathan, S. Venkatesan, and R. Prakash. A feedback based scheme for improving tcp performance in ad-hocwireless networks. Amsterdam, May 1998. 18th International Conference on Distributed Computing Systems (ICDCS).

[2] G. Holland and N. H. Vaidya. Analysis of tcp performance over mobile ad hoc networks. Seattle, August 1999. Annual International Conference on Mobile Computing and Networking (Mobicom'99).

[3] J. Liu and S. Singh. Atcp: Tcp for mobile ad hoc networks. volume 19, pages 1300–1315. IEEE Journal on Selected Areas in Communications, July 2001.

[4] Z. Fu, B. Greenstein, X. Meng, and S. Lu. Design and implementation of a tcp-friendly transport protocol for ad hoc wireless networks. 10th IEEE International Conference on Network Protocosls (ICNP'02), November 2002.

[5] K. Chen, Y. Xue, and K. Nahrstedt. On setting tcp's congestion window limit in mobile ad hoc networks. Anchorage, Alaska, May 2003. IEEE International Conference on Communications (ICC 2003).

[6] IEEE. Wireless lan medium access control (mac) and physical layer (phy) specifications - std 802.11. The Institute of Electrical and Electronics Engineers, 1999.

[7] R. Oliveira and T. Braun. Tcp in wireless mobile ad hoc networks. University of Bern, Technical Report IAM-02-003, July 2001.

[8] J. Li, C. Blake, D. S. J. De Couto, H. I. Lee, and R. Morris. Capacity of ad hoc wireless network. Rome, Italy, July 2001. ACM MOBICOM'01.

[9] Z. Fu, P. Zerfos, H. Luo, S. Lu, L. Zhang, and M. Gerla. The impact of multihop wireless channel on tcp throughput and loss. San Francisco, USA, April 2003. Infocom'03.

[10] M. Allman, V. Paxson, and W. Stevens. Transmission control protocol. RFC 2581, IETF Network Working Group, April 1999.

[11] A. Bakre and B. Badrinath. I-tcp: Indirect tcp for mobile hosts. pages 136–143, Vancouver, Canada, May 1995. IEEE ICDCS'95.

[12] H. Balakrishnan, S. Seshan, E. Amir, and R. Katz. Improving tcp/ip performance over wireless networks. Vancouver, Canada, November 1995. 1st ACM Mobicom.

[13] K. Brown and S. Singh. M-tcp: Tcp for mobile cellular networks. volume 27, pages 19–43. ACM Computer Communications Review, 1997.

[14] C. Zhang and V. Tsaoussidis. Tcp-probing: Towards an error control schema with energy and throughput performance gains. New York, June 2001. 11th IEEE/ACM NOSSDAV.

[15] S. Biaz and N. H. Vaidya. Distinguishing congestion losses from wireless transmission losses:a negative result. New Orleans, LA, USA, October 1998. IEEE 7th Int. Conf. on Computer Communications and Networks.

[16] J. Liu, I. Matta, and M. Crovella. End-to-end inference of loss nature in a hybrid wired/wireless environment. INRIA Sophia-Antipolis, France, March 2003. WiOpt'03: Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks.

[17] R. Oliveira and T. Braun. A delay-based approach using fuzzy logic to improve tcp error detection in ad hoc networks. Atlanta, USA, March 2004. IEEE WCNC 2004.

[18] T. Yuki, T. Yamamoto, M. Sugano, M. Murata, H. Miyahara, and T. Hatauchi. Performance improvement of tcp over an ad hoc network by combining of data and ack packets. to apper in IEICE Transactions on Communications, 2004.

[19] T. Jimenez E. Altman. Novel delayed ack techniques for improving tcp performance in multihop wireless networks. Venice, Italy, September 2003. Personal Wireless Communications (PWC'03).

[20] A. Kherani and R. Shorey. Throughput analysis of tcp in multi-hop wireless networks with ieee 802.11 mac. Atlanta, USA, March 2004. IEEE WCNC'04.

[21] M. Allman. On the generation and use of tcp acknowledgements. volume 28, pages 1114–1118. ACM Computer Communication Review, 1998.

[22] R. Oliveira and T. Braun. A dynamic adaptive acknowledgment strategy for tcp over multihop networks. University of Bern, Technical Report IAM-04-005, July 2004.

[23] S. Xu and T. Saadawi. Does the ieee 802.11 mac protocol work well in multihop wireless ad hoc networks? volume 39, pages 130–137. IEEE Communications Magazine, june 2001.

[24] S. Floyd. Limited slow-start for tcp with large congestion windows. RFC 3742, IETF Network Working Group, March 2004.

[25] S. Floyd, T. Henderson, and A. Gurtov. The newreno modification to tcp's fast recovery algorithm. RFC 3782, IETF Network Working Group, April 2004.