A Smart TCP Acknowledgment Approach for Multihop Wireless Networks

Ruy de Oliveira and Torsten Braun, *Member*, *IEEE*

Abstract—Reliable data transfer is one of the most difficult tasks to be accomplished in multihop wireless networks. Traditional transport protocols like TCP face severe performance degradation over multihop networks given the noisy nature of wireless media as well as unstable connectivity conditions in place. The success of TCP in wired networks motivates its extension to wireless networks. A crucial challenge faced by TCP over these networks is how to operate smoothly with the 802.11 wireless MAC protocol which also implements a retransmission mechanism at link level in addition to short RTS/CTS control frames for avoiding collisions. These features render TCP acknowledgments (ACK) transmission quite costly. Data and ACK packets cause similar medium access overheads despite the much smaller size of the ACKs. In this paper, we further evaluate our dynamic adaptive strategy for reducing ACK-induced overhead and consequent collisions. Our approach resembles the sender side's congestion control. The receiver is self-adaptive by delaying more ACKs under nonconstrained channels and less otherwise. This improves not only throughput but also power consumption. Simulation evaluations exhibit significant improvement in several scenarios.

Index Terms—Wireless multihop networks, transport control protocol, delayed acknowledgments.

1 Introduction

The phenomenal growth experienced by the Internet over the last decade has been supported by a wide variety of evolving mechanisms to meet the requirements of emerging, demanding applications. The basic TCP/IP protocol suite has been instrumental in developing today's Internet. In particular, 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 to 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) is not negligible due to the physical medium. High mobility may further degrade the end-to-end performance because TCP reduces its transmission rate whenever it perceives a dropped packet. We target scenarios in which the level of mobility is relatively low (pedestrian movement) and, so, the focus of this paper is the interaction between TCP and the MAC layer.

IEEE 802.11 [1] 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 most of the potentially lost frames locally on link level. 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 three hops between sender and receiver, which is commonly called a 3-hop scenario (Fig. 1) [2].

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 [3], [4], [5]. This causes an 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 [3], [4], [5] have indicated that this procedure may not be really effective. Rather, this aggressive behavior can 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 TCP retransmissions when reacting to losses.

We present in this paper a dynamic adaptive strategy for decreasing medium contention as much as possible. This paper is an extension of our previous publication [6], where our proposal was introduced. Our approach generalizes the concept of delayed acknowledgments first recommended by RFC 1122 [7] and later refined in RFC 2581 [8], in which the receiver should only send ACK packets for every other data packet received. In our proposal, the receiver may

• R. de Oliveira is with CEFET-MT, Rua Prof. Aline Tocantins 140, 78030-370, Cuiaba-MT, Brazil. E-mail: roliveira@cefetmt.br.
• T. Braun is with the University of Bern, Neubrueckstrasse 10, CH-3012, Bern, Switzerland. E-mail: braun@iam.unibe.ch.

Manuscript received 10 Aug. 2005; revised 14 Mar. 2006; accepted 27 June 2006; published online 14 Dec. 2006.
For information on obtaining reprints of this article, please send e-mail to: tmc@computer.org, and reference IEEECS Log Number TMC-0238-0805.

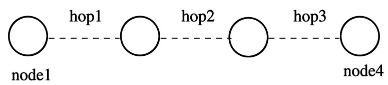


Fig. 1. Typical 3-hop scenario.

combine 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 10 hops, which are the target of this paper. These concepts lead our mechanism to outperform a regular TCP whenever the channel is able to provide higher bandwidth. Our mechanism should perform as effectively as a traditional TCP does when the wireless channel is facing losses.

These simple changes considerably diminish the number of transmissions and retransmissions over the wireless medium. As a consequence, the overall energy consumption is significantly reduced, which is a key issue for battery-powered devices. We focus our discussions on short chains of nodes of at most eight hops because this is a reasonable size for today's networks. Nevertheless, the concepts presented here are general and expected to be effective in larger scenarios as well.

The key optimization in this paper concerns robustness against highly noisy environments since the initial mechanism in [6] is tailored to moderate loss rates. This paper proposes a more conservative strategy for the TCP receiver in terms of reaction and recovery associated to frequent packet losses.

The remainder of this paper is organized as follows: The next section describes the main related work on TCP acknowledgment strategies. In Section 3, we introduce our proposal, where the design decisions are explained and the main features are discussed in detail. Section 4 presents and discusses the simulation results. Section 5 concludes the paper, pointing out the main achievements.

2 Related Work

Several proposals for improving TCP performance, or replacing its mechanisms, over multihop wireless networks have emerged in recent years [9], [10], [11], [12], [13], [14], [15]. The strategy of these proposals is to enhance the TCP sender to react properly to lost packets caused by reasons other than congestion [6]. We focus here on proposals that aim to minimize traffic overhead caused by redundant ACKs because this is a primary goal of our proposed mechanism as well. We also discuss important research work (under standardization through RFCs) on TCP spurious retransmissions as this issue may play a crucial role in supporting our proposed mechanism.

Jimenez and Altman [16] 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 three to four ACKs. In their approach, unless the sender's retransmission timer expires, the receiver always delays four packets, except at session startup. During startup, the receiver begins delaying one ACK only and increases it until four based on the sequence number of the received data packets. The receiver uses a fixed interval of 100 ms for timing out 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. The main weakness of this mechanism, hereafter called LDA (Large Delayed ACK), regards the lack of adaptability to the medium changing conditions. Our proposed algorithm addresses this issue by computing the timeout interval at the receiver "on the fly" and reacting immediately to out-of-order packets, as we will see later. This renders our approach more general. In addition, while the LDA scheme is evaluated under a single flow only, our mechanism is assessed under a varying number of flows, giving encouraging results.

Johnson [17] investigated the impact of using extended delayed acknowledgments intervals on TCP performance. He has performed various experiments in a testbed implemented on a SunOS 4.1.2 workstation. In the experiments, he has changed the kernel's TCP algorithm to allow different numbers of combined ACKs by the receiver instead of only two as proposed in the specification of RFC 1122. In this way, the receiver was adjusted to delay a higher number of ACKs, ranging from one to 20 ACKs. The main outcome is that delaying ACKs in large numbers is always beneficial in short-range networks but may be inappropriate for long-distance networks, especially if congestion is present. This is a consequence of the high interference on RTT estimation caused by the delayed ACKs. The longer the end-to-end connection, the longer the time for the TCP sender to detect lost packets, which may jeopardize the gains obtained by delaying more ACKs. The evaluations were restricted to wired networks and no timeout control at the receiver was implemented. Conversely, our proposed approach targets multihop wireless environments, where typical packet loss rates are much higher than in the wired world. Yet, it relies on an adaptive timeout control at the receiver side.

Allman [18] conducted an extensive simulation evaluation on Delayed Acknowledgment (DA) strategies. This work showed that TCP performance may be hurt by delayed ACKs mainly during the slow start phase. One reason is that the exponential growth of TCP *cwnd* in that phase may produce data bursts in the network, inducing packet drops in the routers buffer. Another problem lies in the ACK-clocked behavior of TCP, in which the sender only increases its *cwnd* by one upon each received ACK. This limits the sender data rate in scenarios where slow start is often invoked. The author proposes two mechanisms to handle the side effects of delayed ACKs: *delayed ACKs after slow start* and *byte counting*. The former requires signaling between sender and receiver to keep the receiver informed about whether slow start is active or not at the sender, so the receiver only delays ACKs when slow start is over. This speeds up data rate recovery during slow start. *Byte counting* allows the sender to increase its *cwnd* on the basis of the number of bytes acknowledged by each ACK instead of the number of ACKs. This procedure can lead to prohibitive bursty traffic conditions and, so, the author also suggested limiting the number of packets sent in response to each incoming ACK to a value of 2. The results showed that both mechanisms can improve performance for implementations using delayed ACKs, but the main concern is the potential increase in packet drops that may happen. The simulation scenario here was also limited to wired networks. We argue that the first proposed mechanism (*delayed ACKs after slow start*) is not relevant for current ad hoc networks were the *cwnd* limit is usually low. Besides, as the media in place is inherently noisy, the signaling from the receiver to the sender is quite unstable. In our proposed mechanism, which is tailored to short-range ad hoc networks, the receiver manages adaptively the acknowledgment rate on the basis of the channel condition. As a result, no explicit signaling is needed. The second mechanism proposed by Allman (*Byte counting*) is a sender-side improvement that may be incorporated into our approach, which is, at this time, a receiver-side mechanism only.

In [6], we published an earlier version of the mechanism proposed in this paper. In that reference, we confirmed important results of previous work [4], [5] in the sense that TCP should limit its congestion window as a function of the number of hops in place to achieve optimal performance. In particular, we showed that a short chain of nodes of up to 10 hops should have a congestion window limit of approximately three packets. This was shown to be caused by the limited spatial reuse property inherent in multihop networks relying on the IEEE 802.11 standard as the MAC protocol. In fact, this limitation is imposed by the hidden node problem present in such environments. The simulation results in [6] were quite encouraging by showing substantial improvements over various scenarios. As mentioned above, this paper proposes an enhancement to the initial mechanism that was not intended to highly noisy environments. The enhancement is described in Section 3.4.

Concerning the approaches that propose to improve TCP performance against spurious retransmissions typical in wireless environments, we discuss here two mechanisms: the Eifel [19], [20], [21] and the F-RTO [22], [23]. Note that these mechanisms do not use delayed acknowledgments strategies, but only improve the sender response to misdetected lost packets. In other words, these approaches are reactive ones while our scheme is proactive by minimizing losses instead of only reacting to them.

The Eifel algorithm aims to eliminate the TCP retransmission ambiguity in order to solve the problems caused by spurious timeouts and spurious fast retransmits. This algorithm uses the TCP timestamp option, so the sender may effectively determine whether a given packet is transmitted for the first time or it is a retransmission. By checking the timestamp in the ACKs, the sender is able to infer spurious retransmissions. In case a retransmission is found to be spurious, the sender restores the parameters of the congestion control that were in place just before the unnecessary retransmission has occurred. As a consequence, the $cwnd$ returns to its previous value and the transmission rate is not reduced wrongly. In its latest version, the algorithm encompasses specific techniques for noisy networks, including a more appropriate way of updating the retransmission timer and a better policy for the $cwnd$ restoration.

F-RTO is an algorithm implemented at the sender side only and does not require any TCP options. In fact, it aims at detecting spurious TCP retransmission timeouts only. A sender using this algorithm keeps track of the incoming acknowledgments (sequence number) after it has transmitted the first unacknowledged packet triggered by a timeout. In this way, it can decide whether to send new packets or retransmit unacknowledged ones. As stated by the authors, F-RTO can be seen as a sort of "Limited Transmit" algorithm [24], but applied to the RTO recovery. Both Eifel and F-RTO are potential algorithms to be used in conjunction with our mechanism for robustness against the usual unnecessary retransmissions in ad hoc networks.

3 Dynamic Adaptive Acknowledgment

We call our mechanism TCP-DAA (DAA: Dynamic Adaptive Acknowledgment), which targets feasible scenarios where the IEEE 802.11 standard may provide acceptable performance. TCP-DAA was first introduced in our prior publication in [6]. Recent investigations on 802.11 have shown that such an ACK delaying protocol is effective in recovering many of the wireless induced losses in typical scenarios, but it does not scale as the number of wireless hops increase. This happens because the well-known hidden node problem imposes a limited spatial reuse property in these networks as discussed in detail in [6]. 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, small working offices, Wi-Fi in home buildings, wireless mesh networks, and many others.

3.1 Design Issues

TCP-DAA design is based on the following observations: TCP reliability requires that transmitted packets are acknowledged by the receiver side. 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 RTS/CTS control frames, the overall overhead at the MAC layer, for transmitting ACKs, is not negligible.

The problems associated to ACK overhead can be mitigated if the receiver merges several acknowledgments into a single ACK, which is possible due to the cumulative ACK scheme used in TCP. This scheme uses later acknowledgments as a confirmation that all previous acknowledgments were successfully received. We showed in [6] that the action of delaying ACKs is really effective in scenarios without the classical hidden node problem, i.e., scenarios of at most three hops. We emphasize here that such an observation is valid for scenarios facing the hidden node problem as well.

Table 1 illustrates how significant an ACK transmission might be in such environments. These results are the outcome of simulation runs in a chain topology of one hop

TABLE 1
Time Medium Is Busy for Data and ACK Transmissions in a 1-Hop Scenario (in msec)

	DATA	ACK	ACK/DATA
No delayed ACK	678.8	60.3	0.089
Standard DA	737.2	32.8	0.044
TCP-DAA	781.4	17.5	0.022

(no intermediate nodes) for a single flow that lasts 10 seconds. Throughout this paper, we use the TCP NewReno flavor [25] as the regular TCP. The values in Table 1 represent the total time the medium is busy transmitting either data or ACK packets. From this table, it is evident that techniques for delaying ACKs can be indeed efficient in multihop environments. The last column of the table, which exhibits the ACK/DATA ratio in percent, shows that the standard delayed acknowledgment (DA) provides significant enhancements. Likewise, Table 1 highlights the remarkable performance of TCP-DAA for this scenario by bringing down the ACK overhead, relative to data packets, from approximately 8.9 percent to about 2.2 percent. Note that the time values in Table 1 do not include all the delays involved in the transmission, but only the data and ACK transmission delay.

The lower number 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 congestion window toward the limit upon receipt of ACKs. So, the fewer ACKs per data, the longer the sender takes to enlarge its congestion window fully. This problem is not so critical in our technique, however, as the *cwnd* limit in place (four 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, as discussed in Section 3.4. One question that may arise here is how to notify the TCP algorithm to use DAA or not. We do not address this problem specifically here, but a mechanism monitoring the channel condition would serve this purpose [26].

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. We believe that solutions like the ones proposed by the F-RTO [22] or the Eifel algorithms [21] (Section 2) might be useful here. This evaluation is left for future work, though. The standard delayed acknowledgment (DA) proposed in RFC 2581 recommends that a receiver should send one ACK for every other data packet received (combine two ACKs into a single one) and should not delay an ACK when either an out-of-order packet or a packet filling a gap in the receiver's buffer is received. Besides, the maximum delay should not exceed a given time interval (typically 100 ms).

Table 1 confirms the findings in [27] in that the standard DA improves performance in wireless environments. Nevertheless, higher enhancements are possible by combining more than two ACKs, as shown in the LDA approach described in Section 2.

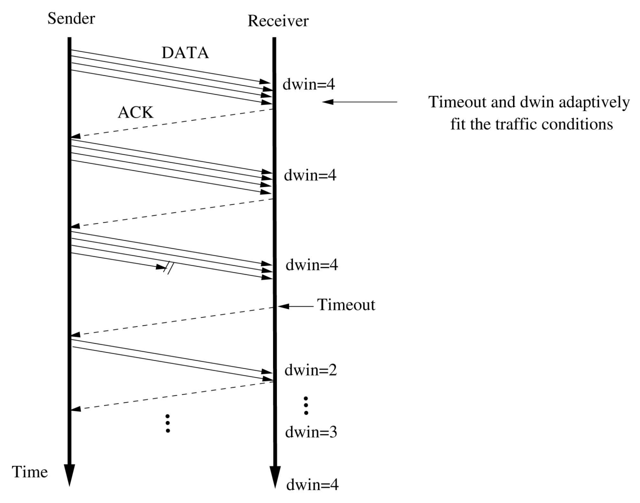


Fig. 2. TCP-DAA approach.

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

TCP-DAA combines the idea of a 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 incoming packet interarrival 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 showed in [6], TCP-DAA outperforms the standard DA and LDA in several scenarios.

3.2 Algorithm

The current development of TCP-DAA is focused on the receiver side, while a comprehensive investigation on the sender side is still to be done. 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 three to two packets, which is in line with [24] in the sense that we work with a small *cwnd* limit, and 2) the regular retransmission timeout interval RTO is increased fivefold for compensating the maximum of four combined ACKs. These are the only two changes performed on the regular TCP sender, which proved to be effective in most of our evaluations.

The dynamic behavior of TCP-DAA is depicted in Fig. 2. After startup and having no losses, the receiver always merges four ACKs. This means that, for every four received data packets, the receiver replies with a single ACK. The delay management is performed through a delaying window (*dwin*) at the receiver that limits the maximum number of ACKs to be delayed. Under normal conditions,

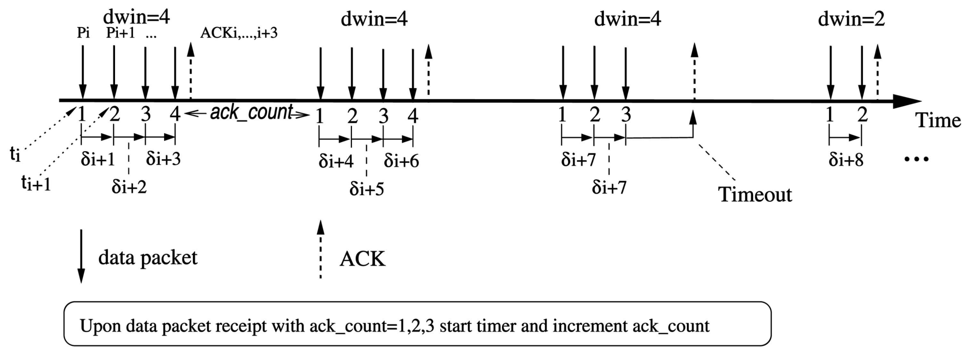


Fig. 3. TCP-DAA receiver mechanisms.

dwin is initialized to one and increases gradually for each received data packet until it reaches four. The limit of four is imposed by the sender *cwnd* limit that is also set to four. Higher *dwin* would not work because the sender would not have enough data packets to transmit to meet the *dwin* value, which would lead the sender permanently to timeout. It is important to note that, despite the influence of the sender setup on the *dwin* size, this is an exclusive variable of the receiver. In other words, there is no transfer of the *dwin* value to the sender.

As long as the wireless channel is unconstrained, it is advantageous to keep $dwin = 4$. When facing losses, however, *dwin* should be reduced in order to avoid further performance degradation. Thus, if *dwin* is kept set to four, it may inappropriately trigger retransmissions by timeout at the sender due to lack of ACKs. To detect a constrained channel, the receiver keeps a timer that is reset whenever it receives a data packet that is going to have its ACK delayed. Additionally, the receiver keeps track of the sequence numbers of incoming data packets. So, it may detect a poor channel when receiving out-of-order packets.

Whenever the receiver gets a packet that is either out-of-order or filling a gap in the receiver's buffer, or when its timer expires, it immediately sends an ACK to the sender and reduces *dwin* to the size of two packets. We chose to resume *dwin* growth from two instead of one because we aimed in such situations to go back to a behavior similar to that of the standard DA, which performs better than configurations without it. Cutting *dwin* down to one is more conservative and may be proper for highly noisy environments where considerable improvements are hard to achieve, as discussed in Section 3.4. Fig. 2 illustrates a situation in which the receiver timer expires due to a dropped packet. Note that *dwin* is first decreased to two, then increased to three and, subsequently, to four as new data packets arrive.

Fig. 3 illustrates in more detail how the receiver keeps track of the packet interarrival interval and handles the ACK delay. Under normal conditions, i.e., after startup and without any loss, for every four data packets received $(P_i, P_{i+1}, P_{i+2}, P_{i+3})$, the receiver replies with an acknowledgment $(ACK_{i,\ldots,i+3})$. Whenever a given acknowledgment $(ACK_i, ACK_{i+1}, ACK_{i+2}, \ldots)$ 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. This timer is used to prevent ACKs from being excessively delayed at the receiver when the network is facing packet losses.

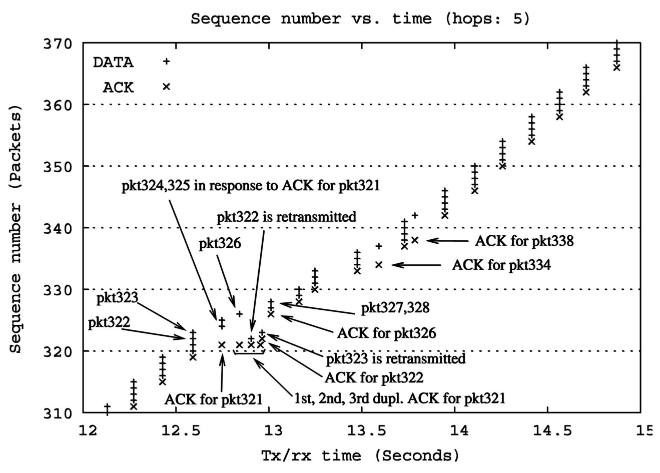
The receiver also measures the data packet interarrival time between the packets for which the ACK is to be delayed $(\delta_i, \delta_{i+1}, \delta_{i+2}, \ldots)$. The receiver keeps track of the number of ACKs delayed by maintaining an *ack_count* variable which increases from one to the current value 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 of packets that is going to have the acknowledgments delayed. In case a packet is the first one, the interarrival interval between the last received packet and the current one is not taken. This is needed to avoid that improper intervals, such as the one between δ_{i+3} and δ_{i+4} in Fig. 3, are considered for the timeout interval computation. By using this strategy, we assure that, in normal conditions, the interarrival 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 the TCP sender, the receiver uses a low-pass filter to smooth the packet interarrival intervals. Upon arrival of a given data packet p_{i+1}, it calculates the smoothed packet interarrival interval as indicated in (1), where $\overline{\delta}_i$ refers to the last calculated value, δ_{i+1} is the packet interarrival interval sampled, and α is the interarrival smoothing factor with $0 < \alpha < 1$.

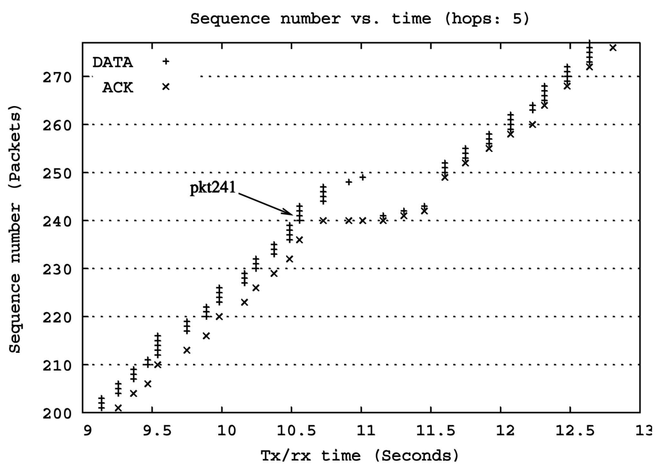
$$\overline{\delta}_{i+1} = \alpha * \overline{\delta}_i + (1 - \alpha) * \delta_{i+1}. \tag{1}$$

The value computed from (1) is used to set the timeout interval at the receiver. 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 is triggered and the receiver avoids sending an extra and unnecessary ACK packet into the network.

Hence, we use a timeout interval T_i as shown in (2). Notice that the factor 2 in (2) refers to the estimated time for



(a)



(b)

Fig. 4. Delayed acknowledgment strategies. (a) TCP-DAA. (b) LDA.

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 smoothed value $\overline{\delta}_i$ and may be higher depending on the value of κ. This equation only provides an upper bound (for robustness) to the delay imposed on the ACKs deferred at the receiver. Further investigations to optimize this variable are surely needed.

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

After a reduction in $dwin$, subsequent timely data packets trigger $dwin$ growth toward the maximum size again. Timely data packets here refer to the incoming data packets that are neither out-of-order nor filling a gap in the receiver's buffer. Using this dynamic behavior, associated to the timer-based monitoring, the receiver prevents the sender from missing ACKs when packet losses occur. As mentioned above, the LDA proposal [16] works with a fixed $dwin$ 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 the sender from missing ACKs, it may also induce an excessive number of retransmissions at the sender [6].

The $dwin$ growth is governed by (3), which shows that such an increase may be fixed at one (packet) or determined by the startup speed factor μ, with $0 < \mu < 1$. The reason for this factor is that, during the startup phase (beginning of the session), the sender starts with a window size of two packets and then increases it by one at every ACK received. Although $dwin$ is initialized to one, if it started from startup being increased at the rate of one packet per incoming data packet, there might happen a shortage of ACKs at the sender. As a result, either receiver or sender would only be able to transmit by timeout (i.e., after their respective timers expiration). 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 $true$. From our evaluations, we noticed that, by properly setting the μ parameter, our algorithm achieved better performance for short-lived flows [6].

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

The mechanisms explained above make TCP-DAA effective because they actively monitor the channel condition to use the scarce channel bandwidth efficiently. When the channel is facing really poor conditions, TCP-DAA should perform in general as effective as a standard TCP. Using its dynamic adaptive window, TCP-DAA somehow probes the network for resource availability, since it will always combine more ACKs (up to four) when the network condition permits. The sender's transmission rate is limited by the MAC layer that manages the contentions in the wireless medium.

3.3 Packet Loss Handling

In order to better understand 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 between our proposal and LDA. Fig. 4 exhibits a part of a simulation run in which both strategies faced a lost packet in a chain topology of five hops.

Let $packetn$ be the data packet of sequence number (n). Fig. 4a shows that the sender transmits four packets (320-323) at time 12.6 seconds. In this run, $packet322$ and $packet323$ are dropped. 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 packets were acknowledged. At this moment, there are only two 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 (first and second duplicate ACKs for $packet321$). 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 instant 12.97 seconds, the sender gets the acknowledgment for $packet322$, allowing it to retransmit the missing $packet323$, 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 the 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 increases toward 4. Fig. 4a shows two spurious retransmissions caused by timeout at the receiver. $Packet334$ 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. Notice that the problem here is not the same as the one addressed in [19], [23], where the spurious retransmissions take place at the sender.

Fig. 4b 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 number of packets in flight is 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. 4b that, although only one packet has been dropped, various acknowledgments triggered the transmission of less than the optimal four packets at the sender. 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, and so it takes a longer time to take action. In short, by comparing Fig. 4a with Fig. 4b, 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.

3.4 An Improved Delaying Window Strategy for High Loss Scenarios

The basic delaying window strategy in Section 3.2 may be inefficient in scenarios facing considerable loss rates. In this section, we investigate improvements to such scenarios. We first observe that, if the channel is facing constant losses, then it seems to be more appropriate to reduce the delaying window ($dwin$) to one in order to avoid timeout at the receiver. Additionally, the $dwin$ should be enlarged by less than one for every data packet received. This is more conservative than the initial strategy above, which is needed to ensure robustness for the mentioned scenarios. Hence, we propose adjusting the receiver side as illustrated in Fig. 5.

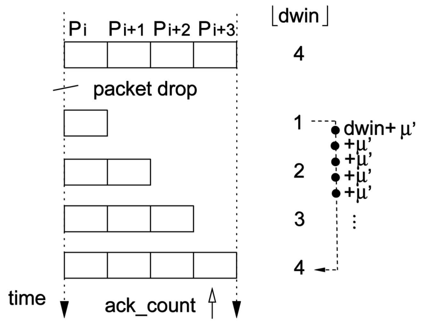


Fig. 5. An alternative delaying window strategy for robustness against losses. $dwin$ is reduced fully to one, and then increased slowly by steps defined by the μ' parameter

Upon loss detection by either timeouts or out-of-order packets, the receiver transmits an acknowledgment immediately and shrinks $dwin$ to one. By receiving new, in-order packets ($P_i, P_{i+1}, P_{i+2}, P_{i+3}$), the receiver gradually expands $dwin$ by steps smaller than one. The operator $\lfloor x \rfloor$ represents the mathematical floor function which is defined as follows: For a real number x, $\lfloor x \rfloor$ results in the largest integer less than or equal to x. In other words, $\lfloor dwin \rfloor$ represents the integer part of $dwin$.

Fig. 5 illustrates that only the integer part of $dwin$ is needed in the comparison with ack_count. This establishes three ranges in which $dwin$ increases without causing any impact on the number of packets to be delayed. These ranges are between the successive $\lfloor dwin \rfloor$ values in Fig. 5, i.e., between 1-2, 2-3, and 3-4. It is obvious that, the smaller the steps by which $dwin$ increases, the more points in each of these ranges and, consequently, the longer the interval to $dwin$ fully enlarges until four.

It is not trivial to determine the exact amount by which $dwin$ should be increased when an ACK is transmitted as many factors influence $dwin$ growth. For example, when the wireless channel is unconstrained, $dwin$ should increase as fast as possible, and under high loss rates, it should grow more slowly. We estimate here the worst case scenario as an upper bound only rather than a rigorous specification.

A TCP receiver should provide enough ACKs to its corresponding sender in order to prevent retransmission by timeout at the sender and also to trigger the sender $cwnd$ growth properly until its limit. Assume that the sender has just timed out while in steady state. Its $cwnd$ is reset to one and the slow start threshold $ssthresh$ is set to one half of the current $cwnd = 4$, i.e., $ssthresh$ is set to two. In this case, the sender increases its $cwnd$ by one when the next ACK arrives because it is in slow start phase ($cwnd < ssthresh$) and, then, it enters the congestion avoidance phase. The $cwnd$ increase (in packets) for the ith received ACK during congestion avoidance is given by (4), where $cwnd_{i-1}$ refers to the previous value of $cwnd$.

$$cwnd_i = cwnd_{i-1} + \frac{1}{cwnd_{i-1}}. \tag{4}$$

Although $cwnd$ grows exponentially in slow start and linearly in congestion avoidance, we can use the equation above for both phases because of our small window limit of four packets. In fact, since $ssthresh$ is set to two upon loss

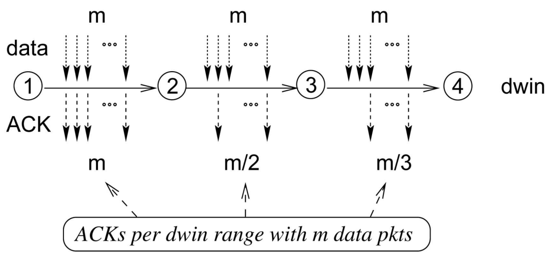


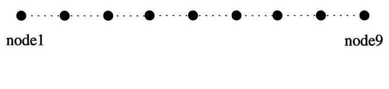
Fig. 6. Estimated number of ACKs necessary to support the sender to smoothly enlarge its congestion window fully in environments facing high loss rates. Each $dwin$ interval receives m data packets and transmits a different number of ACKs toward the minimum of one ACK per four data packets.

detection, only one ACK is enough to lead the sender to congestion avoidance. Moreover, replacing $cwnd_{i-1}$ in (4) with one (the reset value), the left-hand side of the equation results in two, which is exactly the same that is obtained with slow start. Hence, assuming that $cwnd$ increases continuously from one to four governed by (4), the accumulated window increase W is given by (5), where $cwnd_0$ is the value to which $cwnd$ is set just after a slowdown and $cwnd_i$ is the value of $cwnd$ at the ith increase step, which ranges from one to n and is given by (4). If a loss is detected by timeout, $cwnd_0$ is reset to one. A loss detected by the fast retransmit mechanism causes $cwnd_0$ to be reset to a value between 1 and 2, depending on the current $cwnd$ value. For simplicity, we assume $cwnd_0 = 1$ in the modeling below.

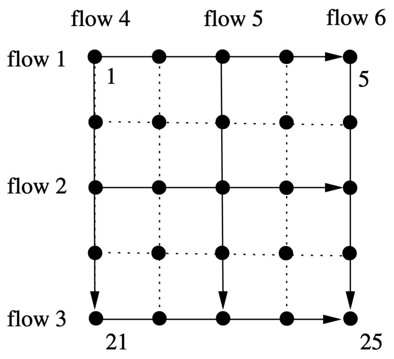
$$W = cwnd_0 + \sum_{i=1}^{n}\left(\frac{1}{cwnd_i}\right). \tag{5}$$

Solving (5) for $W = 4$, the $cwnd$ limit in our mechanism results in $n = 7$. This means that the window expansion process takes seven steps to reach the maximum size of four packets. Therefore, the receiver should take this value into consideration when enlarging its $dwin$. Fig. 6 illustrates how many steps the $dwin$ should follow to satisfy the sender demand for ACKs to avoid timeout at the sender. While $dwin$ is less than two (first range), each data packet received triggers the transmission of one ACK and $dwin$ increases by $1/m$. So, the receiver transmits m ACKs in response to m data packets received. When $dwin$ is between 2 and 3 (second range), every other data packet generates an ACK, which results in approximately $m/2$ ACKs being transmitted in this range. Likewise, for $dwin$ between 3 and 4 (third range), an ACK is sent for every three data packets and, so, about $m/3$ ACKs are transmitted in this range.

To meet the sender needs in terms of acknowledgments during the interval, the sender congestion window is growing toward four and the number of ACKs in the same period should be equal to the amount of expected $cwnd$ increases n that are necessary to expand $cwnd$ to the limit, i.e., seven ACKs. Thus, the sum of the ACKs generated in each range of Fig. 6 should result in seven. In other words, $m + \frac{m}{2} + \frac{m}{3} = 7$, which results in $m = 3.8$. The inverse of m gives us the $\mu\prime = 0.26$ parameter, which determines how much $dwin$ should increase per correct data packet received. Thus, the equation governing $dwin$ growth is



(a)



(b)

Fig. 7. Simulated scenarios. (a) Chain topology. (b) Grid topology.

changed from (3) above to (6). The switching between (3) and (6) has to be conducted by a proper mechanism monitoring the channel condition, such as the ones proposed in [14], [26].

$$dwin = \begin{cases} dwin + \mu, & \text{if in startup} \\ dwin + \mu', & \text{otherwise.} \end{cases} \tag{6}$$

Section 4.4 shows the performance evaluation of this improved algorithm, called TCP-DAAp (TCP-DAA plus). As addressed in that section, with TCP-DAAp, the algorithm at the sender side should react more promptly to losses. The reason is that the number of retransmissions caused by timeouts is assumed to be significantly higher in such cases. Hence, TCP-DAAp uses a regular RTO increased twofold only to speed up the sender reaction to losses.

4 Performance Evaluations

This section presents the evaluation of TCP-DAA considering many aspects, such as throughput, energy consumption, and friendliness behavior. We compare the performance of TCP-DAA with the main TCP versions and with LDA [16]. The reason for the comparison with LDA is that this scheme also investigates a delayed acknowledgments strategy for improving TCP performance in multihop networks. We also compare our results with other TCP versions in their theoretical best conditions to make sure that our proposal is indeed efficient among a wide range of options. Hence, we simulate the other TCP flavors with two improvements: the standard delayed acknowledgment (DA) and a low limit of three packets for their $cwnd$. In this way, the other TCP flavors should provide their optimal performance.

4.1 Simulation Scenario

We used the ns2 [28], [29] simulator in our evaluations of the two scenarios depicted in Fig. 7, in which we have a single chain topology and a grid topology. The grid topology has 25 nodes and the chain topology has a varying

TABLE 2
General Simulation Parameters

Parameter	Value
Channel bandwidth	2 Mbps
Channel delay (wireless)	25 μs
Transmission range	250 meters
Interference range	550 meters
Packet size	1460 bytes
Queue size	50 packets
Window limit (WL)	3 packets
Regular TCP	NewReno
Routing protocol	AODV
Traffic type	FTP
TCP-DAA α	0.75
TCP-DAA κ	0.2
TCP-DAA μ	0.3
Initial TCP-DAA rec. timeout	200 ms
Simulation time	300 seconds

number of nodes with up to nine nodes. In both topologies, each node is 200 meters away from its closest neighbors. 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 other parameter settings are the ones shown in Table 2.

4.2 Performance in the Chain Topology

4.2.1 Throughput

The end-to-end throughput over a chain topology as depicted in Fig. 7a, but with five hops only, is investigated here. The simulations include varying levels of congestion and comparison with the key existing TCP versions. The regular TCP is simulated with and without DA for a better comparison with related work.

Fig. 8 exhibits a remarkable achievement of TCP-DAA. These results are obtained by taking the average of five runs. TCP-DAA outperforms all the other algorithms. 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 queuing delays due to the higher number of flows in the network. Under such high delays, the packet delay variance 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 [6], 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 exploit the bandwidth left while the sender is waiting for the timeout, then that bandwidth is simply wasted.

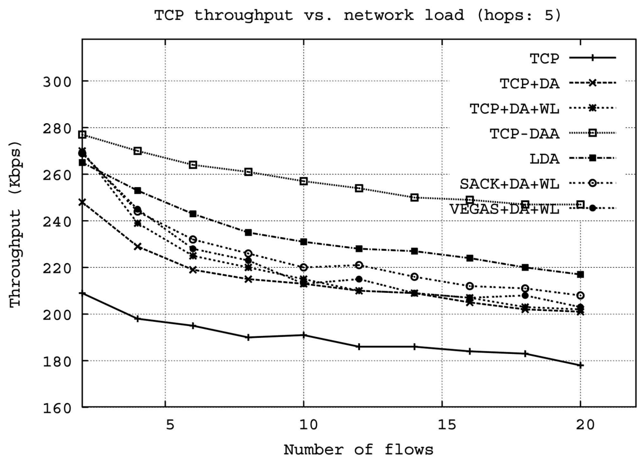


Fig. 8. Aggregate throughput in the chain topology.

TABLE 3
Fairness for 10 Flows Sharing the Medium (Chain Topology)

Algorithm/hops	1	3	5
TCP	1	0.99	0.86
TCP-DAA	1	0.99	0.84

Overall, the observed improvements here are higher than 40 percent over regular TCP. Compared to LDA, improvements of up to about 17 percent are obtained. We also conducted simulations for 1, 2, 3, 4, and 6-hop scenarios and the results are similar; in some cases, less improvement is observed, but in most cases, our algorithm performs significantly better than all the others [6].

4.2.2 Fairness

In order to assess the ability of our mechanism in allowing a fair distribution of bandwidth, we simulate here another scenario for the chain topology in Fig. 7a. We include scenarios with one, three, and five hops (number of wireless links between sender and receiver). In these simulations, a single run is conducted for each TCP version evaluated. In the first run, 10 flows of the standard TCP without any adjustment share the medium, and the next run simulates 10 flows of TCP-DAA. The well-known fairness index

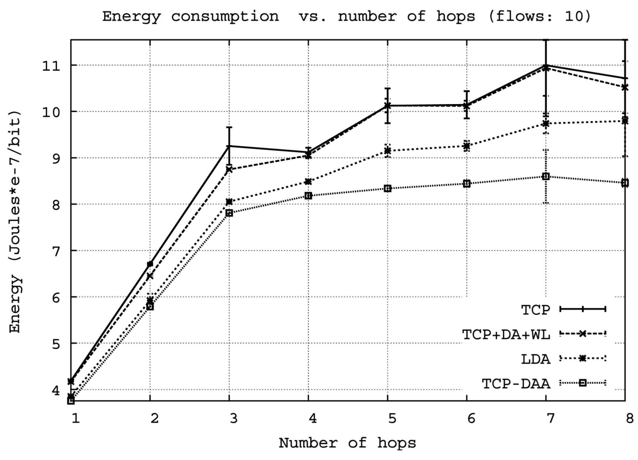
$$\left(\sum_{i=1}^{n} x_i\right)^2 \bigg/ \left(n\sum_{i=1}^{n} x_i^2\right)$$

as defined in [30] is presented in Table 3. By this index, a perfect share of the medium is given by one. That is, the fairer the protocol, the closer to one is the fairness index.

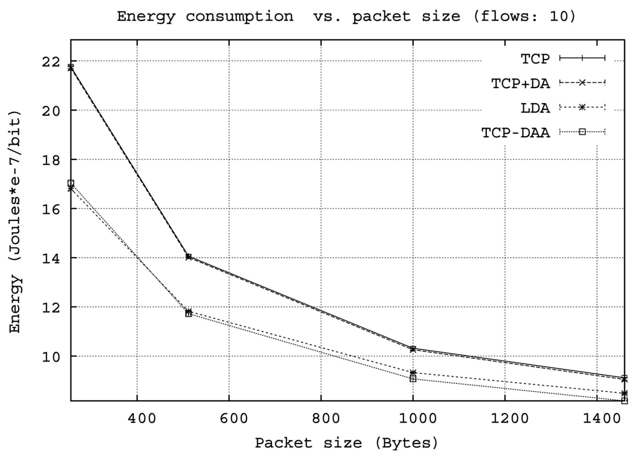
Table 3 suggests that, for short number of hops in this scenario, our mechanism can be as fair as the standard TCP. As the number of hops increases, TCP-DAA tends to perform slightly unfairer than its counterpart. We believe that this behavior might be improved by a more aggressive mechanism at the sender to retransmit packets in due time.

4.2.3 Energy Efficiency

TCP-DAA is expected to be energy saving as it minimizes spurious retransmissions. In this section, we evaluate the performance benefits of TCP-DAA in terms of energy consumption, as depicted in Fig. 9. We used the simple energy model implemented in the ns2 simulator that has been presented in [31]. By this model, a node starts with an initial energy level that is reduced whenever the node



(a)



(b)

Fig. 9. Energy consumption at the TCP sender. (a) Effect of number of hops. (b) Effect of packet size.

transmits, receives, or overhears a packet. Thus, the total amount of energy, $E(n_i)$, consumed at a given node n_i, is given by (7).

$$E(n_i) = E_{tx}(n_i) + E_{rx}(n_i) + (N-1) * E_o(n_i). \quad (7)$$

In (7), E_{tx}, E_{rx}, and E_o denote the amount of energy expenditure by transmission, reception, and overhearing of a packet, respectively. N represents the average number of neighbor nodes affected by a transmission from node n_i [32].

In order to account only for the reception and transmission expenditure, we have discarded the energy spent by overhearing (E_0). This is appropriate to highlight the energy due to TCP transmissions and receptions. Fig. 9a shows the result of a simulation run in which 10 flows share the medium in the chain topology of Fig. 7a for different number of hops. The figure exhibits the energy consumption per bit at the TCP sender. This is computed as $e = \frac{pkt * pkt\_size * 8}{e\_spent}$, where e is the energy/bit ratio, pkt is the amount of packet transmitted by the sender, pkt_size is the packet size in bytes, and e_spent is the energy in joules spent by the sending node.

One can see in Fig. 9a that TCP-DAA provides the best result over all situations. The performance enhancement is more noticeable at a large number of hops, where the probability of collisions is higher. This happens because our algorithm reduces the number of packets in transit. As a result, fewer collisions occur leading to fewer retransmissions and, consequently, higher energy savings. In these simulations, the regular TCP spent about 26 percent more energy than our scheme.

We also evaluated the impact of the packet size on TCP energy consumption as shown in Fig. 9b. In this simulation, four different packet sizes are evaluated, namely, packets 256, 512, 1,000, and 1,460 bytes long. The sender and receiver are connected through four intermediate hops. The results show that the smaller the packet, the higher the energy consumption. This is intuitive because, with small packets, TCP needs to process more packets to transmit the same amount of data than it does when using larger packet sizes. Fig. 9b also shows that, in most cases, but for packet size of 256 bytes, TCP-DAA spends less energy than all the other configurations. The difference is not very significant, though. In this scenario, packet size does not seem to impact energy consumption significantly.

4.3 Performance in the Grid Topology

4.3.1 Throughput

We reproduce here the results shown in [6] regarding the investigation in a more complex scenario, the grid topology illustrated in Fig. 7b. In these evaluations, we firstly have only three flows crossing the topology horizontally (flows 1, 2, and 3 in Fig. 7b). In the next step, six flows (three horizontal and three vertical) are injected into the network concurrently. The results, averaged over five 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 [33], but its overall performance is expected to be similar to that of a regular TCP, as illustrated in Fig. 10.

In these simulations, our mechanism performs roughly the same as the other implementations for the run with only

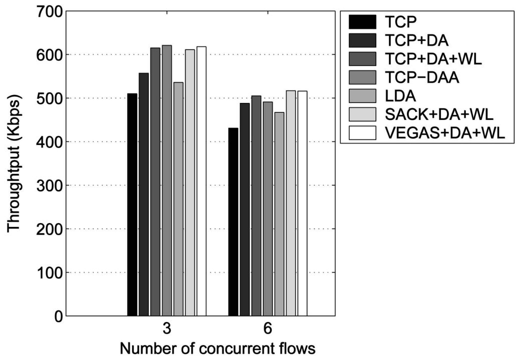


Fig. 10. Aggregate throughput in the grid topology with cross traffic.

TABLE 4
Fairness in the Grid Topology

Algorithm/flows	3	6
TCP	0.72	0.75
TCP-DAA	0.68	0.59

horizontal flows (three flows). Its efficiency deteriorates for the case with six flows, going down to the level of the regular TCP with DA ($\mathrm{TCP} + \mathrm{DA}$). Notice that, while TCP-DAA uses a window limit of four packets, the configuration $\mathrm{TCP} + \mathrm{DA} + \mathrm{WL}$ has a window limit set to three packets. This may explain why TCP-DAA does not reach the performance of the $\mathrm{TCP} + \mathrm{DA} + \mathrm{WL}$ configuration. As shown in [6], the limit of three packets was expected to render better performance for the regular TCP since larger values induce higher collisions. For TCP-DAA, however, a limit of three packets is not appropriate because it does not provide full improvement under moderate loss rates. A trade-off between performance under moderate and high loss rates clearly exists here. 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 versions, but it was implemented over TCP NewReno, which performs well in a variety of scenarios.

4.3.2 Fairness

To look more closely at the performance of our mechanism in the complex scenario made of the grid topology, we compare here the fairness of TCP and TCP-DAA. Table 4 depicts the fairness index of such algorithms for both three flows and six flows. One can see by these values that, in general, none of these algorithms can achieve high fairness. In fact, these results highlight that strategies based on delayed ACKs, like ours, are inherently more unfair than the regular TCP in constrained channels. This happens because the regular TCP increases its *cwnd* faster than approaches relying on delayed ACKs. While the former increases its *cwnd* for each packet received at the receiver, the latter receive roughly half of that. This lack of ACKs at the sender is not a problem in steady state conditions, but it may play a crucial role in this overloaded scenario.

4.4 Optimization for Highly Noisy Environments

In this section, we investigate the optimization proposed in Section 3.4, in which the strategy of the receiver is supposed to be more robust to environments facing nonnegligible losses. Upon losses, the receiver reduces *dwin* to one and slowly increases it again to prevent the receiver timer from expiring by lack of data packets. The analytical evaluation in Section 3.4 showed that, following a very conservative procedure, *ack_count* should increase by about 0.28 for each in-order data packet received. The simulation results illustrated in Fig. 11 conform closely to the analytical prediction.

These evaluations were conducted over the chain topology and each run lasted 1,000 seconds. Each curve indicates the throughput of a single flow competing with nineteen other flows in a 5-hop scenario. Since the scenario is quite constrained in this case, the sender retransmission timeout (RTO) must not be too tolerant as in the previous case. Hence, its tolerance was decreased from fivefold to twofold to conform with the concept in Section 3.4. Various values for the μ' parameter were simulated. Despite the varying behavior of the curves, one can see in Fig. 11 that $\mu' = 0.2$ and $\mu' = 0.3$ tend to provide highest performance.

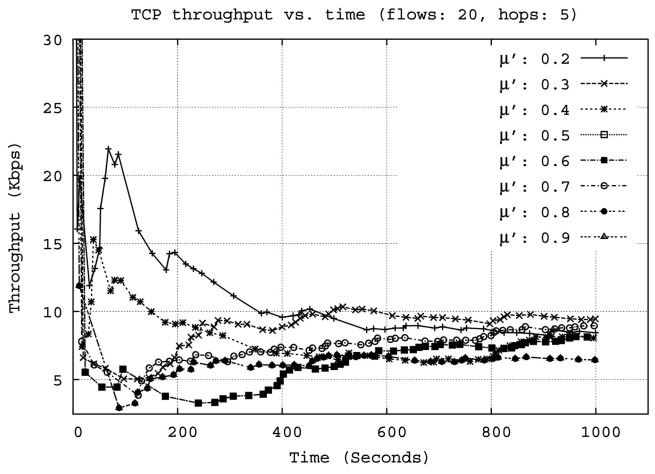


Fig. 11. Optimal μ' parameter for TCP-DAAp.

It is expected that TCP-DAAp does not provide the same improvements of TCP-DAA shown in Fig. 8. This happens because TCP-DAAp transmits more ACKs than the basic version and also because its sender is more aggressive as far as retransmissions are concerned. Fig. 12 exhibits the comparisons between the two TCP versions for the same conditions described in Section 4.2.1. The robustness to losses comes at the cost of throughput under moderate conditions. Nevertheless, the changed algorithm performs as effectively as the regular TCP in Fig. 8.

The justification for the TCP-DAAp strategy is to render our strategy as robust as the regular TCP mechanism under heavily constrained environments. TCP-DAA is not optimized to such environments and, because of that, it degrades substantially under high loss rates. Fig. 13 highlights the importance of TCP-DAAp in a scenario where just

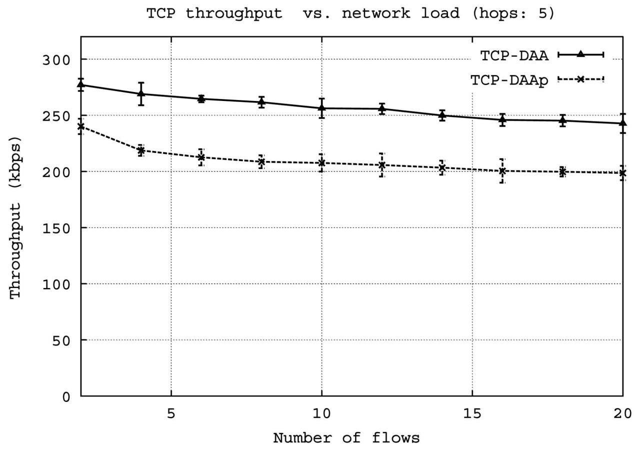


Fig. 12. Comparison between the two TCP-DAA versions.

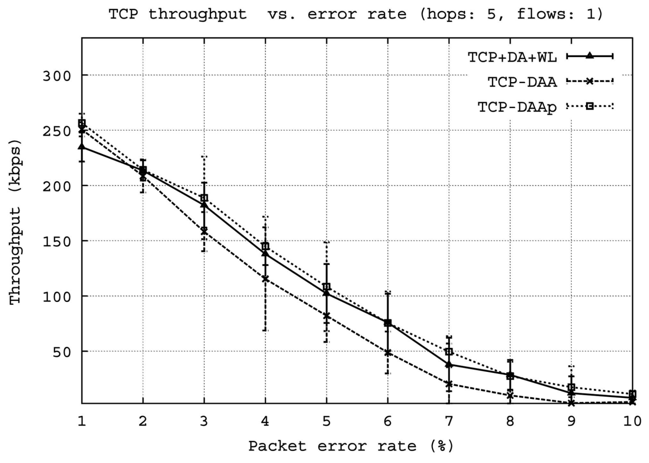


Fig. 13. TCP-DAAp provides robustness for highly noisy scenarios.

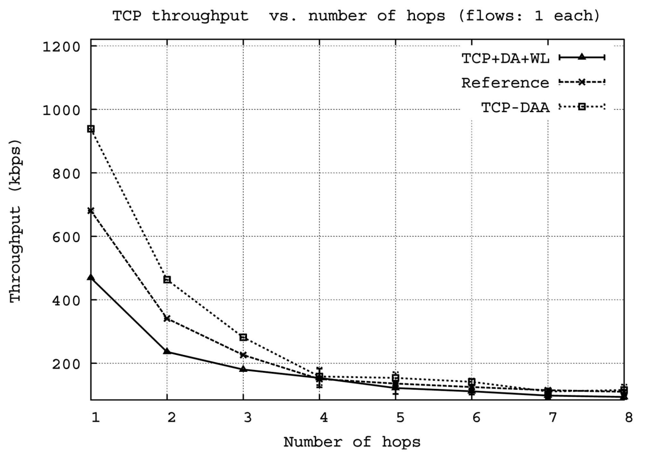


Fig. 14. TCP-DAA friendliness.

a single flow crosses a 5-hop chain of nodes under varying packet error rates. This is a very noisy scenario where not only are losses due to MAC collisions in place but also losses induced by a permanent external disturbance. The error model used follows a uniform distribution function. The results in Fig. 13 shows that, indeed, our strategy can handle losses in an effective way since it performs as effectively as the $\mathrm{TCP} + \mathrm{DA} + \mathrm{WL}$ version. Although it is not shown here, we emphasize that these results are even better over the regular TCP without any further adjustment.

The discussions in the two paragraphs above suggest that it is helpful to have an additional monitoring mechanism at the receiver to adjust the TCP-DAA strategy on the basis of the channel condition. This procedure, along with an improved TCP sender, regarding the RTO computation, can surely render our proposal very robust in a wide range of scenarios. Using such a mechanism, the basic TCP-DAA would be invoked under moderate loss rate and TCP-DAAp would take over when the channel condition deteriorated.

4.5 TCP Friendliness

Gradual deployment requires acceptable friendliness behavior when TCP-DAA is sharing the medium with other flows. This means that our mechanism ideally should not suppress regular flows, but allow them to achieve at least the same throughput they would obtain without any improved flow in parallel. We show here how friendly our mechanism can be when competing with regular flows in a multihop channel facing a moderate loss rate.

Fig. 14 depicts the result of a simulation run in which two flows of distinct versions share the medium. Namely, a TCP-DAA flow competes with a regular TCP that uses DA and window limit (WL). For simplicity, we will hereafter call the configuration $\mathrm{TCP} + \mathrm{DA} + \mathrm{WL}$ "adjusted TCP." It is clear that our mechanism outperforms the adjusted TCP in the range of one to eight hops. The difference between both protocols is noticeable for one to three hops, where the hidden node problem does not happen. After that, more collisions take place and both mechanisms perform similarly. One can say that TCP-DAA performs very aggressively against the adjusted TCP's flow for the cases of 1, 2, and 3 hops.

To measure the degradation imposed by our mechanism over the other flow, we include the "reference curve" in Fig. 14. This curve represents the performance of the adjusted TCP flows without any of our mechanisms in place. Thus, the reference curve is obtained when two adjusted TCPs are sharing the medium. In Fig. 14, the throughput of the adjusted TCP for the 1-hop scenario in Fig. 14 is 469 Kbps. The corresponding throughput for the reference curve is 681 Kbps. This shows an unfairness of our mechanism for this scenario, which leads the adjusted TCP to a decrease in throughput of up to 31 percent.

Another experiment is shown in Fig. 15, where the number of hops are fixed at three and a distinct number of flows are simulated. Since this is a scenario without the hidden node problem, the number of collisions is not very high. Note that, as the number of flows rises, our mechanism degrades performance, leaving more bandwidth to the regular flow. As in the previous case, TCP-DAA induces performance degradation to the adjusted TCP. In this case, the adjusted TCP would achieve about 223 Kbps of throughput if only adjusted TCP flows were being transmitted, but it obtains only 175 Kbps. This means a reduction in throughput of approximately 23 percent.

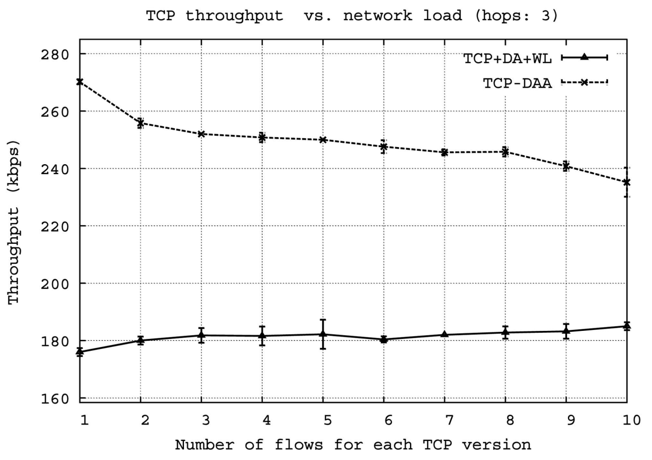


Fig. 15. Aggregate throughput for two distinct TCP versions in parallel under no hidden node problem effects.

The results above suggest that the basic TCP-DAA needs a sort of pacing for controlling its sending rate in mixed scenarios involving non-TCP-DAA flows. A possible mechanism for that is proposed in [18], in which the authors propose to limit (to two packets) the number of packets sent at once by the sender. This comes at the cost of the end-to-end bandwidth utilization, though.

4.6 Discussions

The general perception is that our mechanism is definitely valuable to multihop networks. The results presented here in addition to the ones published in [6] support our claim that a dynamic and adaptive mechanism is effective in such constrained environments. The results in Section 4.4 indicate that the mechanism can be refined to handle highly constrained conditions. Distinct parameter settings for moderate and elevated constraints are needed, though. The key remark here is that our mechanism automatically prevents waste of bandwidth under favorable conditions and performs as effectively as a conventional TCP when traffic conditions deteriorate.

As far as friendly behavior is concerned, our basic mechanism does not seem to be very friendly in scenarios without the hidden node problem. In these scenarios, the number of packet losses is quite negligible, leading our mechanism to use the typically wasted bandwidth efficiently. As a result, it is difficult to adjust TCP-DAA parameters toward very friendly behavior under such conditions. This indicates that a rate limitation at the sender may be beneficial to address this problem. On the other side, TCP-DAA parameters may be optimized under hidden node problem effects to make the protocol friendly [34]. This optimization includes the sender side, which has to be fully investigated in future work.

We believe that solutions like the one proposed in [35], in which the receiver controls the sender's *cwnd*, may be integrated into our final algorithm. Likewise, the work in [36] could be useful for improving the fairness of our mechanism by including the congestion window in the timeout computation at the sender.

It is important to emphasize that our proposal does not change the semantics of TCP, including its Additive-Increase/Multiplicative-Decrease (AIMD) congestion control algorithm. Our mechanism keeps the principles of the AIMD recommended in [37], in that a simple AIMD algorithm satisfies the sufficient conditions for convergence to an efficient and fair state. The unfairness detected in our proposed algorithm is essentially caused by the timeout mechanism rather than by the AIMD mechanism.

5 Conclusions

We have extended and further evaluated our algorithm for improving TCP performance over multihop wireless networks. Our dynamic adaptive acknowledgment strategy aims to minimize collisions resulting from mutual interference between data and ACK packets by transmitting as few ACKs as possible. The mechanism is self-adaptive and tailored to networks comprising at most 10 hops and facing moderate bit error rates.

The simulation evaluations showed that our algorithm can outperform not only conventional TCP, including 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 networks. Yet, it is easy to deploy as the changes are limited to the end nodes only. Future work includes the development of a customized sender algorithm toward an effective balance between throughput and fairness, an adaptive receiver mechanism to switch between DAA and DAAp strategies in scenarios susceptible to high bit error rates, and a more elaborate timeout strategy at the receiver.

Acknowledgments

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. The authors are thankful to Bharat Bhargava, from Purdue University, for his collaboration on this paper.

References

[1] *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications,* IEEE Standard 802.11, 1999.

[2] R. Oliveira and T. Braun, "TCP in Wireless Mobile Ad Hoc Networks," Technical Report IAM-02-003, Univ. of Bern, July 2001.

[3] J. Li, C. Blake, D.S.J. De Couto, H.I. Lee, and R. Morris, "Capacity of Ad Hoc Wireless Networks," *Proc. ACM MobiCom '01,* July 2001.

[4] Z. Fu, P. Zerfos, H. Luo, S. Lu, L. Zhang, and M. Gerla, "The Impact of Multihop Wireless Channel on TCP Throughput and Loss," *Proc. INFOCOM '03,* Apr. 2003.

[5] K. Chen, Y. Xue, and K. Nahrstedt, "On Setting TCP's Congestion Window Limit in Mobile Ad Hoc Networks," *Proc. IEEE Int'l Conf. Comm. (ICC '03),* May 2003.

[6] R. Oliveira and T. Braun, "A Dynamic Adaptive Acknowledgment Strategy for TCP over Multihop Wireless Networks," *Proc. IEEE INFOCOM,* Mar. 2005.

[7] R. Braden, "Requirements for Internet Hosts—Communication Layers," *RFC 1122,* IETF Network Working Group, Oct. 1989.

[8] M. Allman, V. Paxson, and W. Stevens, "Transmission Control Protocol," *RFC 2581,* IETF Network Working Group, Apr. 1999.

[9] K. Chandran, S. Raghunathan, S. Venkatesan, and R. Prakash, "A Feedback Based Scheme for Improving TCP Performance in Ad-Hoc Wireless Networks," *Proc. 18th Int'l Conf. Distributed Computing Systems (ICDCS '98),* May 1998.

[10] G. Holland and N.H. Vaidya, "Analysis of TCP Performance over Mobile Ad Hoc Networks," *Proc. MobiCom '99,* Aug. 1999.

[11] S. Biaz and N.H. Vaidya, "Distinguishing Congestion Losses from Wireless Transmission Losses: A Negative Result," *Proc. IEEE Seventh Int'l Conf. Computer Comm. and Networks,* Oct. 1998.

[12] J. Liu and S. Singh, "ATCP: TCP for Mobile Ad Hoc Networks," *IEEE J. Selected Areas in Comm.,* vol. 19, pp. 1300-1315, July 2001.

[13] Z. Fu, B. Greenstein, X. Meng, and S. Lu, "Design and Implementation of a TCP-Friendly Transport Protocol for Ad Hoc Wireless Networks," *Proc. 10th IEEE Int'l Conf. Network Protocosls (ICNP '02),* Nov. 2002.

[14] J. Liu, I. Matta, and M. Crovella, "End-to-End Inference of Loss Nature in a Hybrid Wired/Wireless Environment," *Proc. Symp. Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks (WiOpt '03),* Mar. 2003.

[15] K. Sundaresan, V. Anantharaman, H.-Y. Hsieh, and R. Sivakumar, "ATP: A Reliable Transport Protocol for Ad-Hoc Networks," *IEEE Trans. Mobile Computing,* vol. 4, no. 6, Nov./Dec. 2005.

[16] T. Jimenez and E. Altman, "Novel Delayed ACK Techniques for Improving TCP Performance in Multihop Wireless Networks," *Proc. Personal Wireless Comm. (PWC '03),* Sept. 2003.

[17] S.R. Johnson, "Increasing TCP Throughput by Using an Extended Acknowledgment Interval," master's thesis, Ohio Univ., June 1995.

[18] M. Allman, "On the Generation and Use of TCP Acknowledgements," *ACM Computer Comm. Rev.,* vol. 28, pp. 1114-1118, 1998.

[19] A. Gurtov and R. Ludwig, "Responding to Spurious Timeouts in TCP," *Proc. INFOCOM '03,* Mar. 2003

[20] R. Ludwig and M. Meyer, "The Eifel Detection Algorithm for TCP," *RFC 3522,* IETF Network Working Group, Apr. 2003.

[21] R. Ludwig and A. Gurtov, "The Eifel Response Algorithm for TCP," *RFC 4014,* IETF Network Working Group, Feb. 2005.

[22] P. Sarolahti and M. Kojo, "Forward RTO-Recovery (F-RTO): An Algorithm for Detecting Spurious Retransmission Timeouts with TCP and the Stream Control Transmission Protocol (SCTP)," *RFC 4138,* IETF Network Working Group, Aug. 2005.

[23] P. Sarolahti, M. Kojo, and K. Raatikainen, "F-RTO: An Enhanced Recovery Algorithm for TCP Retransmission Timeouts," *Computer Comm. Rev.,* vol. 33, no. 2, 2003.

[24] M. Allman, H. Balakrishman, and S. Floyd, "Enhancing TCP's Loss Recovery Using Limited Transmit," *RFC 3042,* IETF Network Working Group, Jan. 2001.

[25] S. Floyd, T. Henderson, and A. Gurtov, "The NewReno Modification to TCP's Fast Recovery Algorithm," *RFC 3782,* IETF Network Working Group, Apr. 2004.

[26] R. Oliveira and T. Braun, "A Delay-Based Approach Using Fuzzy Logic to Improve TCP Error Detection in Ad Hoc Networks," *Proc. IEEE Wireless Comm. and Networking Conf. (WCNC '04),* Mar. 2004.

[27] S. Xu and T. Saadawi, "Does the IEEE 802.11 MAC Protocol Work Well in Multihop Wireless Ad Hoc Networks?" *IEEE Comm. Magazine,* vol. 39, pp. 130-137, June 2001.

[28] D. Estrin, K. Fall, S. Floyd, J. Heidemann, A. Helmy, P. Huang, S. McCanne, K. Varadhan, Y. Ya, and H. Yu, "Advances in Network Simulation," *Computer,* vol. 33, no. 5, pp. 59-67, May 2000.

[29] D. Estrin, M. Handley, J. Heidemann, S.S. McCanne, X. Ya, and H. Yu, "Network Visualization with Nam, the VINT Network Animator," *Computer,* vol. 33, no. 11, pp. 63-68, Nov. 2000.

[30] R. Jain, *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling.* Wiley-Interscience, 1991.

[31] Y. Xu, J. Heidemann, and D. Estrin, "Adaptive Energy Conserving Routing for Multihop Ad Hoc Networks," Research Report 527, Information Sciences Inst., Univ. of Southern California, Oct. 2000.

[32] D. Kim, J.J. Garcia-Luna-Aceves, K. Obraczka, J. Cano, and P. Manzoni, "Power-Aware Routing Based on the Energy Drain Rate for Mobile Ad Hoc Networks," *Proc. IEEE Int'l Conf. Computer Comm. and Networks (ICCCN '02),* Oct. 2002.

[33] R. Oliveira and T. Braun, "A Dynamic Adaptive Acknowledgment Strategy for TCP over Multihop Networks," Technical Report IAM-04-005, Univ. of Bern, July 2004.

[34] R. de Oliveira, "Addressing the Challenges for TCP over Multihop Wireless Networks," doctoral thesis, Inst. of Computer Science and Applied Math., Univ. of Bern, 2005.

[35] V. Tsaoussidis and C. Zhang, "TCP-Real: Receiver-Oriented Congestion Control," *Computer Networks J.,* vol. 40, no. 4, Nov. 2002.

[36] I. Psaras, V. Tsaoussidis, and L. Mamatas, "CA-RTO: A Contention-Adaptive Retransmission Timeout," *Proc. Int'l Conf. Computer Comm. and Networks (ICCCN '03),* Oct. 2005

[37] D.M. Chiu and R. Jain, "Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks," *Computer Networks and ISDN Systems,* vol. 17, no. 1, 1989.

Ruy de Oliveira received the MS degree from the University of Uberlandia, Brazil, in 2001, and the PhD degree from the University of Bern, Switzerland, in 2005. From 2001 to 2002, he was involved in the European Union research project SEQUIN for QoS across multiple management domains. From 2002 to 2005, he was a research fellow in the long-term Swiss research project NCCR-MICS on self-organizing wireless networks. He served as the local organizing committee chair for the Third International Workshop on Applications and Services in Wireless Networks (ASWN '03). Currently, he is a full professor in the Department of Computer Science at CEFET-MT, Brazil. His research interests are in data communication protocols and security in wireless networks.

Torsten Braun received the diploma and the PhD degrees from the University of Karlsruhe, Germany, in 1990 and 1993, respectively. From 1994 to 1995, he was a guest scientist with INRIA Sophia Antipolis. From 1995 to 1997, he worked as a project leader and senior consultant at the IBM European Networking Center, Heidelberg, Germany. Since 1998, he has been a full professor of computer science at the Institute of Computer Science and Applied Mathematics (University of Bern, Switzerland), heading the Computer Networks and Distributed Systems research group. He has been a board member of SWITCH (Swiss Education and Research network) since 2000. During his sabbatical in 2004, hewas a visiting scientist at INRIA Sophia-Antipolis and the Swedish Institute of Computer Science at Kista. He is a member of the IEEE and the IEEE Computer Society.