© The Author 2005. Published by Oxford University Press on behalf of The British Computer Society. All rights reserved.
For Permissions, please email: journals.permissions@oupjournals.org
doi:10.1093/comjnl/bxh000

Design and Performance Evaluation of a Transport Protocol for Ad hoc Networks (TPA)

Giuseppe Anastasi[1], Emilio Ancillotti[2†*], Marco Conti[2], Andrea Passarella[2]

Pervasive Computing and Networking Laboratory (PerLab)

[1] *Dept. of Information Engineering, University of Pisa, Italy*
[2] *CNR-IIT National Research Council, Italy*
[†] *Corresponding Author: Emilio Ancillotti, National Research Council (CNR), Institute for Informatics and Telematics (IIT), Via G. Moruzzi 1, 56124 Pisa, voice: +39 050 315 2437 - Italy*
Email: emilio.ancillotti@iit.cnr.it

Providing efficient transport services over multi-hop ad hoc networks is a fundamental building block for this wireless technology. The typical approach is modifying TCP to fix one (or a few of) its inefficiency while preserving compatibility with the original protocol. However, a complete solution should include a significant number of modifications, such that the original TCP design is deeply modified. In this paper we explore a different approach. We include the desired modifications to TCP in the design of a new transport protocol (TPA). In this way we are able to blend together these features in a unique design framework, and better control interactions among the different (modified) components. We then compare TCP and TPA through field tests, in terms of throughput and total number of transmitted segments. We consider several possible configurations of the protocols parameters, different routing protocols, and various networking scenarios. In all the cases taken into consideration TPA significantly outperforms TCP. To achieve a more thorough understanding of the TPA behavior, we compare TPA and TCP also in terms of fairness and scalability (both in static and mobile configurations) over a wide range of representative topologies. To this end, we adopt a simulation approach, which is more suitable to this kind of analysis. Simulation results confirm field tests, and show that TPA is able to outperform TCP with respect to all analysed performance figures.

Keywords: Ad hoc networks, TCP, Transport Layer, Performance Evaluation, Experimental Analysis, Routing protocols

Received 00 Month 2004; revised 00 Month 2004

1. INTRODUCTION

Research on efficient transport protocols for ad hoc networks is one of the most active topics in the MANET community. Such a great interest is basically motivated by numerous observations showing that, in general, TCP is not able to efficiently deal with the unstable and very dynamic environment provided by multi-hop ad hoc networks. This is because some assumptions in its design are clearly inspired by the characteristics of wired networks dominant at the time when it was conceived. More specifically, TCP implicitly assumes that segment loss is almost always due to congestion phenomena causing buffer overflows at intermediate routers. Furthermore, it also assumes that nodes are static (i.e., they do not change their position over time). Unfortunately, these assumptions do not hold in MANETs since in this kind of networks packet losses due to interference and link-layer contentions are largely predominant, and nodes may be mobile.

To address TCP inefficiencies over ad hoc networks, a number of proposals have been presented, which are

*This work has been partly carried out while Emilio Ancillotti was with the Dept. of Information Engineering of the University of Pisa

overviewed in Section 2. The vast majority of these proposals are TCP modifications that address some particular inefficiency. The main design requirement is indeed to keep the improved transport protocol backward compatible with the legacy TCP, so that "improved" and "legacy" users may be able to communicate with each other.

While we acknowledge the importance of TCP compatibility, in this paper we advocate a different design approach. The TCP inefficiencies over ad hoc networks are so many that a single modification is in general not sufficient to address them all. Thus, TCP would need a large number of modifications to work in a real environment. The practical integration and interoperability between such patches is a critical point, which is still to be addressed. In this paper we propose to turn this methodology upside down. Firstly, we review the main inefficiencies of TCP over ad hoc networks, and the desired protocol's features to address them. These features are blended together in a unique design framework, which results in a new transport protocol (Transport Protocol for Ad hoc Networks – TPA). Finally, we discuss how TPA and TCP can coexist.

The resulting TPA is a lightweight transport protocol that provides a connection-oriented, reliable type of service. It differs from TCP in a number of ways. Specifically, the data transfer and the congestion control algorithms have been re-designed. Furthermore, TPA explicitly detects and deals with both route failures and route changes. TPA can leverage cross-layer interactions with the routing protocol, when available. For example, it is able to intercept and interpret route failure and route re-establishment messages. However, TPA works also with routing protocols that do not provide this type of information. The complete description of TPA is provided in Section 3, and a brief discussion about TPA/TCP interoperability is provided in Section 6.

The second contribution of this work is a detailed comparison between TCP and TPA. We adopt a mixed evaluation methodology, made up of tests over a *real* multi-hop ad hoc network, and simulations. Actually, most of the research on ad hoc networks has been carried out by adopting simulation and analytical approaches only. We acknowledge that simulation and analysis are great tools to manage large scales, and understanding the sensitiveness of a protocol to particular parameters. At the same time, we argue that, in the field of ad hoc networking, they should be complemented with real experiments. Indeed, the behaviour of wireless links is so difficult to model, that sometimes results from real experiments and from simulation/analysis are quite different (e.g., [1, 2, 3, 4, 5, 6]). Therefore, a careful check of simulation/analytical results against experiment outcomes is highly advisable. On the other hand, it is also clear that experiments alone are not able to provide the necessary flexibility and control of the environment to study systems behavior in general cases, and at large scales, while simulation is more suitable for this. In this paper we thus adopt a mixed evaluation methodology, based both on real experiments and on simulation.

We exploit a TPA prototype (briefly described in Appendix A) to set up a multi-hop network testbed to compare TPA with TCP along the lines discussed in Section 4. We configure the testbed to replicate relevant scenarios for small-scale ad hoc networks. Besides allowing us to control the testbed setup, focusing on small-scale networks is aligned with the well-known definition of the "ad hoc horizon" [7], which, based on both theoretical and experimental results, states that the reasonable "horizon" for practical multi-hop ad hoc networks is between 10 and 20 nodes, with connections spanning 2 to 3 hops. Firstly, we focus on a string network (throughout referred to also as chain topology) with variable number of hops (Section 4.3). We use this setup to investigate the sensitiveness of TPA to its main parameters and its behaviour over different routing protocols (i.e., OLSR and AODV). The experimental results show that TPA is able to improve TCP performance both in terms of increased throughput (between +4% and +20%), and reduced number of (re)transmissions (between -60% and -98%). This is an important result, as it states that TPA delivers greater throughput, while reducing energy consumption and network congestion *at the same time*. We then consider different topologies (i.e., a cross topology, Section 4.4), and mobile scenarios (Section 4.5). Also in these cases TPA outperforms TCP with respect to both performance figures.

We then present simulation results targeted at comparing TPA and TCP in terms of i) large networks, and ii) fairness (Section 5). Simulation is clearly a better choice than experiments when considering large networks, as controlling large testbeds is very hard. Simulation is also more suitable to investigate fairness among concurrent connections, as it permits to replicate the exact external environment (e.g., interference level) on different connections, which is clearly not possible in reality. Therefore, simulations highlight the real properties of the protocols in terms of fairness.

The trends of throughput and re-transmission performance highlighted by experiments are confirmed also at larger scales. As far as fairness, TPA is generally fairer than TCP, and this is not paid with throughput degradation. However, the fairness level of TPA is not satisfactory. Therefore, we also evaluate the protocols' performance when the adaptive pacing mechanism (proposed in [8]) is used. TPA with adaptive pacing achieves a drastic improvements in terms of fairness. TCP with adaptive pacing is even a bit fairer, but this is paid with a drastic reduction of the throughput. We briefly show how to slightly modify the adaptive pacing mechanisms so as to make TPA fairer than TCP, without compromising its advantages in terms of

throughput. In summary, also the simulation results confirm that TPA is able to significantly outperform TCP over all the investigated performance figures, including fairness.

2. RELATED WORK

This section presents various proposals available in literature to improve TCP performance over MANETs. To classify them we use an approach similar to that proposed in [9]. Specifically, we grouped the proposals in five categories: i) distinguishing between route failures and congestion; ii) reducing the effect of route failures; iii) reducing wireless channel contention; iv) improving TCP fairness; and v) designing new transport protocols. A separate subsection is devoted to each approach.

As described in Section 3, one of the main TPA objectives is blending together the main modifications proposed for TCP improvement in a single design framework (specifically, TPA deals with issues discussed in Sections 2.1, 2.3, and 2.4). This is the main difference with respect to the approaches presented hereafter, which focus on single TCP inefficiencies. We separately discuss the differences between TPA and the other new protocols for ad hoc networks in Section 2.5.

2.1. Distinguishing between losses due to route failures and congestion

Node movements may cause route failures and route changes which result in packet losses and late ACKs at the sender side. TCP misinterprets these events as a sign of congestion and activates the congestion control mechanism. This may lead to unnecessary retransmissions and throughput degradation [10, 11, 12, 13, 14, 15]. Several TCP modifications thus aim to discriminate between packet losses due to route failures and packet loss due to congestion.

Works in [14], [10], and [16] propose similar mechanisms based on a feedback to the TCP sender (called TCP-F, ELFN, and TCP-BuS, respectively). Explicit information is provided to the TCP sender upon route breakage. The sender waits for an explicit message of route re-establishment (in TCP-F and TCP-BuS), or periodically probes for a new route (in ELFN). Liu et al. [17] propose to leave the TCP implementation unchanged, and insert a thin layer (ATCP) between IP and standard TCP to improve the TCP behaviour. ATCP deals with problems related to high Bit Error Rate, node mobility, and classic network congestion. To discover the network state and to react consequently, ATCP exploits ECN (Explicit Congestion Notification) [18] and ICMP "Destination Unreachable" messages.

Works in [12], [19], and [20] infer route loss and re-establishment based on measures taken directly at the transport layer (without relying on support from the routing layer). Dyer et al. [12] detect route loss after two consecutive timeout expirations, and blocks the TCP Retransmission TimeOut (RTO) exponential increase. TCP-Door [19] interprets out-of-order segments (at the receiver) as an indication of a route failure, which is piggybacked to the TCP sender in the next ACK. The sender disables congestion control upon receiving this information. Finally, ADTCP [20] exploits joint end-to-end metrics to distinguish between different network states like congestion, channel error, route change, and disconnection.

2.2. Reducing the effect of route failures

The proposals in [21] and [22] predict route failures likely to occur in the near future to reduce the route reconstruction latency. They basically differ in the way route failures are predicted. Klemm et al. [22] also include mechanisms at the MAC level to manage routes during link breakages. Split TCP [23] splits long TCP connections into shorter localized segments to reduce the number of route failure in connections that have a large number of hops. This also improves the fairness between multiple flows. He et al. [24] propose a scheme to use a narrow-bandwidth, out-of-band busy tone channel to reserve the data channel for broadcast transmissions, and exploits the busy channel also to perform link error detection. Lim et al. [25] use multipath routing protocols, which may result in inaccuracy in average RTT measurement and out-of-order packet delivery. Thus, they introduce *backup path routing*, which uses only one path at a time but maintains backup paths so as to be able to rapidly switch to an alternative path if needed. Finally, with the strategy proposed by Chung et al. [26], after a link failure occurs, the old route will continue to be used until a new one can be established through a route-discovery procedure.

2.3. Reducing wireless channel contention

Many papers [27, 28, 29, 30, 31, 32, 33] have shown that TCP performs poorly even in static MANETs, due to erroneous interactions between the original TCP and the MAC level. The problem is essentially a by-product of the exposed-node configuration, and manifests itself as soon as nodes in the path are not within the same carrier sensing range. TCP typically grows its average congestion window size much larger with respect to the optimal size (identified in [30]), and earlier nodes in the path block later nodes. To address this issue, in [28, 30] authors propose to explicitly bound the TCP congestion window to a fixed size to reduce channel contention. Chen et al. [31] define an algorithm to dynamically adjust the maximum congestion window size according to the round-trip hop-count of the connection. Papanastasiou et al. [32] and Nahm et al. [33] propose Slow Congestion Avoidance Scheme (SCA) and Fractional Window increment (FeW), which do not limit the congestion window, but increase the sending rate more slowly than in the original TCP protocol.

Another direction explored to reduce the effects of channel contention is modifying the way in which ACK segments are sent. Authors of [28, 34, 35] tune the delayed ACK mechanism to reduce the number of ACK segment in transit. Cordeiro et al. [36] instead propose COPAS (COntention-based PAth Selection), which uses disjoint forward (for TCP data) and reverse (for TCP ACK) paths to reduce the conflicts between TCP segments travelling in opposite directions. In addition, COPAS continuously monitors network contention and selects routes with minimum contention.

A further approach to reduce contention issues is proposed in [30]. Link Random Early Detection (Link-RED) monitors the average number of retransmissions at the link layer. When this number becomes greater than a given threshold (i.e., too high contention is occurring), the probability of dropping/marking packets is computed in a similar way to the RED algorithm [37]. As a side effect this slows down the TCP sender. Furthermore, they force nodes to add an extra backoff interval (at the MAC level) after a successful transmission, to mitigate the effects of the well-known MAC-level unfairness problems.

2.4. Improving TCP fairness

Transport-level fairness problems have been reported in several papers [29, 38, 39, 40], both in wireless and in mixed wired/wireless environments. A number of approaches have been therefore proposed to cope with these issues.

Yang et al. [41] propose a "non work-conserving scheduling" mechanism. The link layer queue sets a timer whenever it sends a data packet to the MAC. The queue outputs another packet to the MAC only when the timer expires. The duration of the timer is updated according to the queue output rate value. Xu et al. [39] show that legacy RED does not completely solve TCP's unfairness in MANETs due to lack of coordination among nodes, and propose a Neighbourhood RED (NRED) scheme. With respect to legacy-RED, NRED manages all the the queues of a set of neighbours at once, both as far as monitoring the queues' level, and as far as deciding drop actions. Jiang et al. [42] propose and evaluate the use of a distributed *max-min air-time allocation* algorithm to approximate the proportional fairness objective. Finally, Huang et al. [42] propose a distributed algorithm that allows each node to locally determine its *max-min per-link* fair share without knowledge of the global network topology.

With respect to transport-level fairness, we specifically mention the work in [8], where authors propose a modified version of the *adaptive pacing* mechanism available for the Internet (named TCP-AP). TCP-AP spreads the segments transmission according to a dynamically computed transmission rate. Moreover it incorporates a mechanism to identify incipient congestion and to consequently adjust the transmission rate. The detailed presentation of this mechanism is reported in Section 5, as TPA exploits the adaptive pacing mechanism too.

2.5. Designing new transport protocols

The main proposals which design a new transport protocol instead of modifying a particular TCP aspect are ATP [43] and TPA.

ATP exploits cross-layer interactions, and includes rate-based transmissions, network-supported congestion detection and control, no retransmission timeout, decoupled congestion control and reliability. Each node in the ATP path piggybacks its available rate in data segments. This information is collected and consolidated by the ATP receiver, and is then periodically sent back to the ATP sender, which computes the transmission rate accordingly. Another ATP feature is that it doesn't use retransmission timeouts for reliability. Instead, it uses selective ACKs to periodically report back to the sender any new hole observed by the receiver in the data stream. A drawback of ATP is that it requires assistance from all intermediate nodes along the connection path. This may make it unsuitable for those environments where the network layer protocol does not provide such a support. Opposite to ATP, even if novel in many respects, TPA conserves some TCP characteristics that are actually suitable for the ad hoc environment. For example, TPA uses a window-based transmission scheme and preserves the TCP end-to-end semantic. In addition, TPA works properly also without any assistance from the underlying protocol (e.g., when ELFN messages are not provided by the network layer protocol), and does not require assistance from all nodes along the path.

This paper complements our previous work on TPA [44], [45], and [46]. In [44] we provided an early description of the TPA design features. In [45] we discussed a preliminary *simulation* analysis of TPA in comparison with TCP. In [46] we extended the TPA evaluation through an *experimental* analysis on a chain topology, evaluating the effect of different routing protocols. In this paper we extend the experimental analysis to different topologies, and we also consider mobile scenarios. We also present a detailed simulation analysis to evaluate the TPA performance in large topologies (both static and mobile), and to analyse issues related to fairness.

3. TPA DESCRIPTION

The TPA protocol provides a reliable, connection-oriented type of service. The main TPA design goals are defined by observing the TCP limitations when used over multi-hop ad hoc networks, and can be summarized as follows:

- The data transfer policy should be resilient to late and out-of-order segments, and smoothly

work with multi-path routing. In TCP all these circumstances typically trigger timeout or useless re-transmissions at the sender, both reducing the throughput, and wasting energy and bandwidth.
- Several papers (e.g., [30]) have shown that the optimal transmit window size of a TCP connection over multi-hop ad hoc networks is limited to a few segments. The flow and congestion control algorithms should take this into consideration, and can thus be greatly simplified with respect to TCP. Furthermore, when congestion is over, the transport protocol should try to exploit the available bandwidth more promptly than the TCP additive increase policy does.
- Congestion and route failures are different phenomena in ad hoc networks. However, TCP basically has no notion of route failure, and manages this phenomenon through congestion control. Instead, congestions and route failures should be managed via different algorithms, to accommodate distinct and more refined policies.
- Route changes in ad hoc networks can be frequent events, and new paths can provide significantly different performance in terms of delay, congestion level, etc. The transport protocol should adapt quickly to new paths' features, and discard statistics about old paths. This is not the case in TCP, where, for example, the algorithm used to estimate Round Trip Times and set the Retransmission Timer privileges old statistics with respect to new samples. While this prevents statistics' flapping, sometimes this might not be the best way to manage route changes in ad hoc networks.
- Previous papers have shown that Delayed ACK techniques can be helpful to reduce the network congestion, and ultimately increase the transport-protocol efficiency. We thus include this mechanism in the TPA design.

In the rest of this section we present the TPA aspects that permit to meet the above goals. A description of the TPA implementation is presented in Appendix A.

3.1. TPA segment structure

The TPA segment consists of a *header field* and a *data field*. The *data field* contains a chunk of application data. The MSS (Maximum Segment Size) limits the maximum size of a segment's data field. The smallest TPA header is composed of 16 bytes. Figure 1 shows the structure of the TPA segment. Since TPA is full-duplex, the TPA header contains the fields of both data and ACK segments. Specifically, the header includes the following fields (note that the precise use of some of these will be explained in detail in the following sections):

- **SourcePortNumber and DestinationPortNumber fields**: are the usual port numbers.

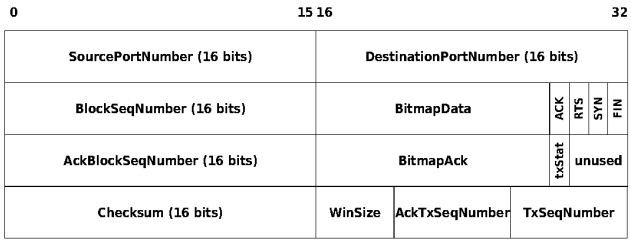

FIGURE 1. TPA header.

- **BlockSeqNumber**: identifies the block to which the data segment belongs (see Section 3.2).
- **BitmapData**: consists of 12 bits and identifies the position of the data segment within the block (see Section 3.2).
- **AckBlockSeqNumber**: identifies the block to which the acknowledged segment belong. This field is valid only if the ACK flag is set.
- **BitmapAck**: consists of 12 bits and describes all the segments belonging to the current transmission block correctly received by the destination. A bit set in the *BitmapAck* indicates that the corresponding segment within the block *AckBlockSeqNumber* has been correctly received by the destination. The receiver can acknowledge more than one segment by setting the corresponding bits in the BitmapAck. This field implements a SACK-like functionality, and is valid only if the ACK flag is set.
- **Flag field**: contains 4 bits: The **ACK** bit indicates that the value carried in the *AckBlockSeqNumber*, *BitmapAck*, and *AckTxSeqNumber* fields are valid. The **RST** bit resets the connection. The **SYN** and **FIN** bits are used for connection setup and teardown.
- **WinSize**: is the receiver advertised window.
- **txStat**: is used by the TPA sender to announce its status (*congested* or *not congested*) to the receiver (see Section 3.6).
- **Checksum**: usual checksum, calculated by the sender over the header and the data fields.
- **TxSeqNumber**: This field is used by the sender to identify the data segment sent. Each time TPA sends a data segment, it increments the *TxSeqNumber* field by one. When TPA changes the transmission block, it resets the *TxSeqNumber* field.
- **AckTxSeqNumber**: used by the sender to identify the segment which the ACK corresponds to. For example, if the ACK was generated by a data segment with the *TxSeqNumber* field set to i, then the receiver sets the *AckTxSeqNumber* field to i. This field is needed since the *BitmapAck* field does not identify the segment that generates the ACK. The *AckTxSeqNumber* field is valid only if the ACK flag is set.

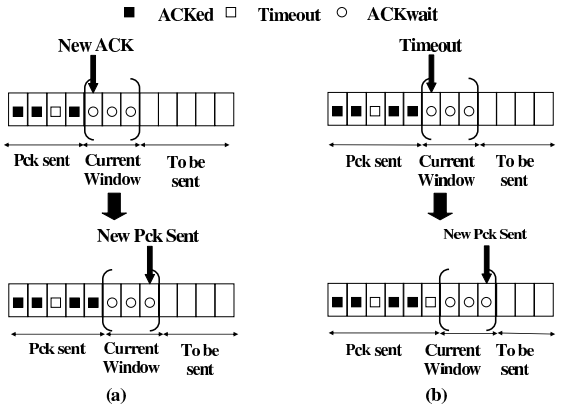

FIGURE 2. ACK reception (a), and timeout expirations (b).

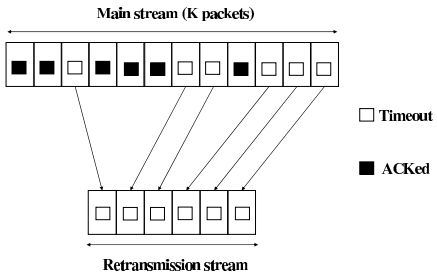

FIGURE 3. Retransmission Stream.

- **unused**: this field is reserved for future use.

3.2. Data transfer

TPA is based on a sliding-window scheme where the window size varies dynamically according to the flow control and congestion control algorithms. The flow control mechanism is similar to the corresponding TCP mechanism [47] while the congestion control mechanism is described in Section 3.5.

TPA tries to minimize the number of (re)transmissions in order to save energy. To this end, data to be transmitted are managed in blocks, with a block consisting of K segments, whose size is bounded by the Maximum Segment Size (MSS)[2]. The source TPA grabs a number of bytes - corresponding to K TPA segments. - from the transmit buffer, encapsulates these bytes into TPA segments, and transmits them reliably to the destination. Only when all segments belonging to a block have been acknowledged, TPA takes care to manage the next block.

Possibly, using large values of K might introduce additional delays to early segments in the block, due to the time required to fill the block with K segments. TPA has no control on this time, as it depends on the data generation pattern of the application. To overcome this problem, TPA defines a timeout for filling the buffer, and transmits a block consisting of less then K segments if the timeout expires. We have run some preliminary experiments to understand the impact of the K parameter on the performance of *ftp-like* traffic (not shown here due to lack of space). They have shown just a minor dependence of the considered performance figures on this parameter.

Segment transmissions are handled as follows. Whenever sending a segment, the source TPA sets a timer and waits for the related ACK (i.e. a segment with the corresponding bit in the ACK bitmap set) from the destination. Upon receiving an ACK for an outstanding segment the source TPA performs the following steps: i) derives the new window size according to the congestion and flow control algorithms (see below); ii) computes how many segments can be sent according to the new window size; and iii) sends next segments in the block (see Figure 2a). On the other hand, whenever a timeout related to a segment in the current window expires, the source TPA marks the segment as "timed out" and executes steps i)-iii) as above, just as in the case the segment was acknowledged (see Figure 2b). In other words, for each block TPA first performs a transmission round during which it sends all segments within the block, without retransmitting timed-out segments. Then, the sender performs a second round for retransmitting timed-out segments, which are said to form a "retransmission stream" (see Figure 3). In the second round the sender performs steps i)-iii) described above with reference to the retransmission stream instead of the original block. This procedure is repeated until all segments within the original block have been acknowledged by the destination. If an ACK is received for a segment belonging to the retransmission stream, that segment is immediately dropped from the stream.

The proposed scheme has several advantages with respect to the retransmission scheme used in TCP. First, the probability of useless transmissions is reduced since segments for which the ACK is not received before the timeout expiration are not retransmitted immediately (as in the TCP protocol) but in the next transmission round. Second, TPA is resilient against ACK losses because a single ACK is sufficient to notify the sender about all missed segments in the current block. Third, the sender does not suffer from the duplicated ACKs generated in TCP by the receiver upon receiving out-of-order segments. This implies that TPA can operate efficiently also in multi-hop ad hoc networks using multi-path forwarding [48].

[2] All segments but - possibly - the last one are long MSS bytes.

3.3. Route failure management

Like many other solutions [14, 10, 17, 49], TPA can exploit, if available, the *Explicit Link Failure Notification* (ELFN) service provided by the network-layer for detecting route failures.

Upon receiving an ELFN, the source TPA enters a *freeze* state where the transmission window size is limited to one segment. In general, we assume that the network layer does not provide route re-establishment notifications. Thus, at the expiration of each retransmission timeout (see Section 3.4), TPA sends segments in the main or retransmission streams (as per the Data Transfer algorithm), probing the network for a new route. To limit the number of segments sent when there is no available route, while in the freeze state, the value of the retransmission timer doubles after each timer expiration. Therefore, TPA realizes that the route has been re-established as soon as it receives an ACK for the latest segment sent. Upon reception of such an ACK, TPA i) leaves the *freeze* state; ii) sets the congestion window to the maximum value $cwnd_{max}$; and iii) starts sending new segments. On the other hand, if route re-establishment messages are available, the TPA behaviour can be further optimized. Specifically, in the freeze state TPA can refrain from transmitting any segment, waiting for a route re-establishment message.

Even if the underlying layer does not provide the ELFN service, the sender TPA is still able to detect route failures as it experiences a number of consecutive timeouts. Specifically, the sender TPA assumes that a route failure has occurred whenever it detects th_{ROUTE} consecutive timeouts. In this case it enters the *freeze* state, and behaves as described above.

3.4. Route change management

We argue that a transport protocol for multi-hop ad hoc networks should react more quickly to route changes than TCP does. To understand why, let us briefly explain how TCP and TPA evaluate the Retransmission Timeout (RTO).

Similarly to TCP, TPA estimates the connection RTT, and uses this estimate to set the Retransmission Timeout (RTO). Both parameters are derived in the same way as in the TCP protocol, i.e.:

$$ERTT_{rtt}(n) = g \times RTT(n) + (1 - g) \times ERTT_{rtt}(n - 1)$$
$$DEV_{rtt}(n) = h \times |RTT(n) - ERTT_{rtt}(n)| + (1\text{-}h) \times DEV_{rtt}(n\text{-}1)$$
$$RTO(n) = ERTT_{rtt}(n) + 4 \times DEV_{rtt}(n)$$

where: i) $ERTT_{rtt}(n)$ and $DEV_{rtt}(n)$ are the average value and standard deviation of the RTT estimated at the *n-th* step, respectively; ii) RTT(n) denotes the *n-th* RTT sample; iii) RTO(n) is the retransmission timeout computed at the *n-th* step; and iv) g and h ($0 < g, h < 1$) are real values [47].

Multi-hop ad hoc networks are far more dynamic that wired networks, and route changes can be fairly frequent even in static configurations. Whenever a route changes, the new path may differ from the previous one in number of hops. Or, new links in the path may have different properties e.g. in terms of interference. This means that, after a route change, segments may experience a significant variation in the RTT and the retransmission timeout might be no longer appropriate for the new path. However, the standard TCP essentially uses a low-pass filter to estimate RTT, and thus RTO may converge to a value appropriate to the new path after long time possibly resulting in excessive retransmissions. To avoid this, the TPA protocol must detect route changes as soon as they occur, and modify the RTT estimation method to quickly achieve a reliable estimate for the new RTT. In practice, TPA detects that a route change has occurred either i) when a new route becomes available after a route failure; or ii) when th_{RC} consecutive samples of the RTT are found to be external to the interval $[ERTT_{rtt} - DEV_{rtt}, ERTT_{rtt} + DEV_{rtt}]$ (th_{RC} consecutive late segments or th_{RC} consecutive early segments). Upon detecting a route change, TPA replaces the g and h values in the ERTT and DEV estimators with greater values (*g1* and *h1*), so that the new RTT estimates are heavily influenced by the new RTT samples. This allows to achieve a reliable estimate of the new RTT immediately after the route change has been detected. Finally, after n_{RC} updates of the estimated RTT, the parameter values are restored to the normal g and h values.

3.5. Congestion control mechanism

Also congestion phenomena are quite different in multi-hop ad hoc networks with respect to wired networks. The work in [30] shows that congestions in multi-hop ad hoc networks mainly occur because of link-layer contention and not because of buffer overflow at intermediate nodes, as in the case of wired networks. Congestions due to link-layer contentions manifest themselves at the transport layer in two different ways. An intermediate node may fail in relaying data segments to its neighbouring nodes and, thus, it sends an ELFN back to the sender node (provided that this service is supported by the network layer). This case, throughout referred to as *data inhibition*, cannot be distinguished by the sender TPA from a real route failure. On the other hand, an intermediate node may fail in relaying ACK segments. In this case, throughout referred to as *ACK inhibition*, the ELFN (if available) is received by the destination node (i.e., the node that sent the ACK), while the source node (i.e., the node sending data segments) only experiences one or more (consecutive) timeouts. Whenever the sender TPA detects th_{CONG} (with $th_{CONG} >= 1$) consecutive timeout expirations, it assumes that an *ACK inhibition* has occurred, and enters the *congested* state. The source TPA leaves the

congested state as soon as it receives th_{ACK} consecutive ACKs from the destination.

If the network layer does not support the ELFN service, the only way to detect both data and ACK inhibitions is by monitoring timeouts at the sender. Congestions and route failures are no longer distinguishable. Hence, th_{CONG} and th_{ROUTE} collapse in the same parameter, and the *freeze* and the *congested* states collapse in the same state.

Several papers (e.g., [30, 31]) have shown that for TCP connections spanning a small number of hops (below 20) a congestion window of 2 or 3 segments is the optimal choice. As, based on the ad hoc horizon definition [7], these connections are the most likely to occur in real ad hoc network settings, in TPA the congestion control mechanism is window-based as in TCP, but we just considered 2 or 3 as candidate values for the maximum congestion window (the value achieving the best performance depends on the network configuration, as shown by the experimental and simulation results). As a consequence, in TPA the maximum and minimum values of the congestion window are very close, and the TPA congestion control algorithm is thus very simple. In normal operating conditions, i.e., when TPA is not in the *congested* state, the congestion window is set to the maximum value, $cwnd_{max}$. When TPA enters the *congested* state, the congestion window is reduced to 1 to allow congestion to disappear.

3.6. ACK management

Based on the results in [34, 35, 28], showing the benefit of delayed ACK mechanisms to reduce contention, we firstly included in TPA a delayed ACK mechanisms similar to the one proposed in RFC 1122. Specifically, when the sender is not in *congested* state, the TPA receiver sends back one acknowledgement *every other* segment received, or upon timer expiration. Otherwise, if the sender is in *congested* state, the receiver sends back one ACK for *each segment* received (as in this case the sender window size is stuck to one, see Section 3.5). Recall that the sender uses the *txStat* flag of the TPA segment header (see Section 3.1) to announce its status (*congested* or *not congested*) to the receiver.

We then considered an alternative version of the *Delayed ACKs* scheme. Specifically, to minimize the number of ACKs in transit in the network, we modified the receiver to send a single ACK every $cwnd_{max}$ segments, when the sender is not congested. Suppressing more ACKs makes no sense, as the sender would not be able to transmit anything more until a timeout expires at the receiver (and an ACK is thus forcibly sent). Throughout the paper we will refer to TPA with modified version of the *Delayed ACK* technique as *TPA**.

In both TPA variants, the interval that triggers the ACK transmission is set to a constant value (typically, 100 ms).

4. EXPERIMENTAL ANALYSIS

In this section we present results of experiments aimed to compare TPA and TCP in realistic small-scale ad hoc networks. Specifically, we first consider a chain topology with varying number of hops (see Section 4.3), then we consider a cross topology (see Section 4.4), and, finally, we evaluate the impact of nodes' mobility (see Section 4.5). Before presenting the results, we discuss the setting of our testbed, and the adopted performance measures.

4.1. Testbed description

Our testbed consisted of IBM R-50 laptops equipped with integrated Intel Pro-Wireless 2200 wireless cards. All laptops were running the Linux Kernel 2.6.12 with the latest available version of the `ipw2200` driver (1.1.2). Wireless cards followed the IEEE 802.11b specifications with maximum default bit rate set to 2 Mbps (which is the setting used in the vast majority of related works). For completeness, we also replicated the experiments at 5.5 and 11 Mbps. The RTS/CTS mechanism was enabled and RTS/CTS threshold was set to 100 bytes so that RTS/CTS handshake was active for data segments and disabled for ACKs. This setting protected *long* data segments from collisions, while avoiding RTS/CTS overhead for *short* ACK segments, which are less likely to collide. In all configurations, neighbour nodes were placed at the limit of their transmission range.

We compared TPA and TCP performance by considering two different routing protocols, i.e., AODV and OLSR. AODV (Ad hoc On-demand Distance Vector) is a well-known reactive protocol [50]. It is worth recalling here that AODV can use two different mechanisms for neighbour discovery and local connectivity maintenance, i.e., link layer information provided by the underlying MAC protocol, or HELLO messages periodically broadcast by each node to announce its presence in the one-hop neighbourhood. In our testbed we used the AODV implementation for Linux by the Uppsala University [51], version 0.9.1. To maintain local connectivity we set AODV to use HELLO messages since our `ipw2200` driver didn't provide link-layer failure notifications. All the AODV parameters were set to their default values. OLSR (Optimized Link State Routing, [52]) is an optimization for mobile ad hoc networks of the classical link state algorithm (it is thus a proactive protocol). Similarly to AODV, OLSR uses a neighbour discovery procedure based on HELLO messages. In our testbed we used the OLSR_UniK implementation for Linux, version 0.4.10 [53]. We set all the parameters to their default values, and disabled the OLSR *hysteresis* mechanism, because

TABLE 1. TPA Operational Parameters.

Parameter	Value
th_{RC} (TPA)	3 segments
n_{RC} (TPA)	3
g	0.125
h	0.25
g1	0.25
h1	0.5
th_{ROUTE} (TPA)	1 segment
th_{ACK} (TPA)	1 segment
Block Size (TPA)	12 segments

it was shown to significantly degrade the transport-level throughput [54].

Table 1 shows the operational parameters for the TPA protocol. As far as TCP, we configured it to obtain a NewReno behaviour with Delayed-ACKs. We compared TCP and TPA in different configurations. Specifically, we clamped the maximum TCP congestion window to 2 and 3 segments (referred to as *TCP-2* and *TCP-3*), and we also considered – as reference – TCP with unclamped congestion window (referred to as *TCP-uc*). As far as TPA, we clamped the maximum congestion window to 2 and 3 segments as well (referred to as *TPA-2* and *TPA-3*), and we also considered the modified delayed ACK mechanism discussed in Section 3.6 (referred to as *TPA*-3*).

In all the experiments we used *ftp-like* traffic, i.e., the sender node(s) had always data ready to send. Indeed, file-sharing applications are expected to generate similar type of traffic. The MSS in all the experiments was set to 1460 bytes. It is worth mentioning that (as described in detail in Appendix A), the TPA implementation guarantees that the header overhead in TPA and TCP is exactly the same. To capture the TCP traffic we used *tcpdump*, while to analyse the experiments results we used *tcpstat* and *tcptrace* (enhanced by our shell scripts). Since TPA was implemented in the user-space (see Appendix A), to capture the TPA traffic we used our TPA code.

4.2. Performance measures

In our analysis we consider the following two performance measures:

- *Throughput*, i.e., the average number of bytes successfully received by the final destination per unit time.
- *Retransmission index*, i.e., the percentage of segments re-transmitted by the TPA/TCP sender.

The *throughput* was measured at the application layer as the number of bytes successfully received by the destination process in a given time interval, divided by the duration of the time interval. The *re-transmission index* (*rtx*) measures the average number of times a segment had to be retransmitted to be successfully received by the destination. Thus, it is defined as:

$$rtx = \frac{pktRtxSrc}{pktRcvDest}$$

where *pktRtxSrc* is the number of segments retransmitted by the source, and *pktRcvDest* is the number of non-duplicated segments successfully received by the destination.

The re-transmission index allows us to evaluate the ability of TPA/TCP to handle transmissions in an efficient way. It is worthwhile to emphasize that re-transmitted segments consume energy and generate congestion both at the sender and intermediate nodes. As nodes in a multi-hop ad hoc network may have limited power budget, and wireless bandwidth is a scarce resource, it is important to manage (re)transmissions efficiently. Therefore, a small value for the re-transmission index is highly desirable.

To achieve statistical accuracy, we replicated each experiment a number of times (at least five) so that the semi-confidence interval (computed with a 90% confidence level) of the *throughput* was within 10% of the average value. Due to fairness issues discussed in the following, we were not able to achieve this target only in the cross topology experiments, when using TCP on top of OLSR. As far as the retransmission index, it generally shows a higher variability than the throughput. However, in most cases either the retransmission index is so low that it can be approximated with 0, or the difference between TCP and TPA is large enough, so that the fact that TPA outperforms TCP holds true even if the results variability is not very small. The only exception is the experiment using AODV in the roaming-node scenario. In this case no conclusive results can be drawn due to the high variability of results. However, the average values of the retransmission index suggest that TPA outperforms TCP also in this case.

As will be clear in the following, depending on the network setup, TPA and TCP achieve their best performance with different settings (e.g., in terms of maximum congestion window size). Unless otherwise stated, we compare performance figures achieved by TCP and TPA in the respective best configurations. Note that for the same protocol, the best configuration may be different depending on the performance index considered.

4.3. Chain topology

We considered a chain topology with hop count ranging from 1 to 4. In addition, we also ran experiments over a 3-hop chain topology in the presence of interfering traffic. Figure 4 shows the indoor environment where the experiments were carried out. The testbed was deployed in a real working environment with possibly interfering electrical appliances, people

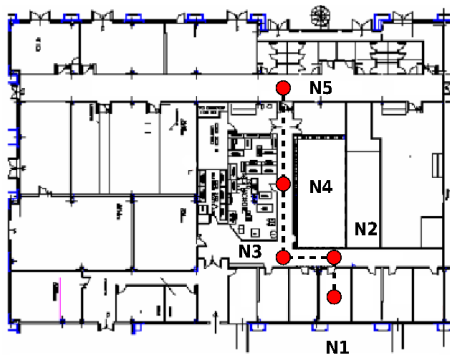

FIGURE 4. Chain Topology network.

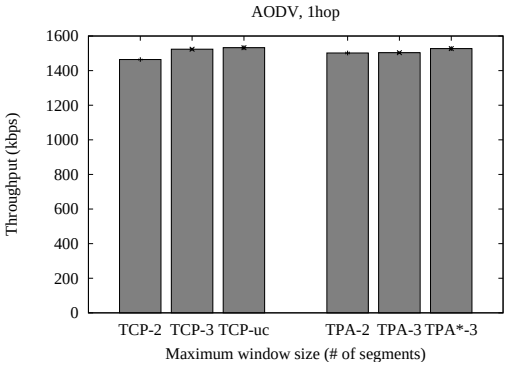

FIGURE 5. Throughput vs. window size in the 1-hop scenario.

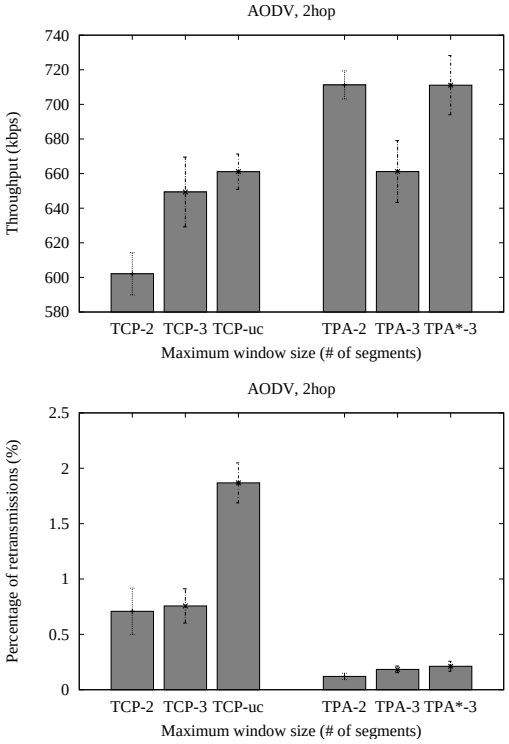

FIGURE 6. Throughput (up) and percentage of retransmitted segments (down) vs. window size in the 2-hop scenario.

roaming around, etc., so as to have a scenario representative of the expected indoor environments where ad hoc networks will operate in. The experiments consisted in a file transfer of about 180s. In all the experiments node *N1* was the sender, while the receiver (and the number of active nodes) depended on the specific chain length. For example, in the 3-hop scenario, node *N4* was the receiver (node *N5* was not active). We set the transmission power of wireless cards and the distance between nodes in such a way that only adjacent nodes were within the transmission range of each other. However, since the transmission range of nodes is not a perfect circle and may vary from time to time [1], we used the `iptables` firewall to filter MAC packets and enforce the desired topology.

4.3.1. Analysis with the AODV routing protocol

Figure 5 through Figure 9 show the throughput and retransmission index of both TCP and TPA in all the scenarios we considered.

Figure 5 shows that in the 1-hop scenario there are not significant differences between TCP and TPA performance. This was expected since in this simple scenario problems related to link-layer contention are managed efficiently by the 802.11 MAC protocol. All nodes are within the transmission range of each other and can thus coordinate efficiently their transmissions. This results in a *retransmission* index equal to zero for all values of the maximum *cwnd* size parameter. TCP and TPA perform almost the same, with a slight advantage for TCP in the best configurations (*TCP-uc* and *TPA*-3*, respectively).

Note that TPA is implemented in the user space (see Appendix A) and, thus it experiences a greater overhead due to a higher number of interactions between the user and the kernel spaces. This possibly explains the performance difference in the one-hop configuration. However, note that in this case TPA and TCP are basically equivalent, and the difference their throughput is below 2%. This is indeed not a multi-hop configuration, where TPA is expected - by design - to outperform TCP. Even assuming that TCP outperformed TPA by 2% in this configuration would be good enough by looking at the overall performance differences in the multi-hop experiments.

Figure 6 shows that in the 2-hop scenario TPA outperforms TCP both in terms of throughput (+7.5% in the best configurations), and retransmission index (-83%). Note also that in a 2-hop scenario the best TCP configuration is *TCP-uc*. This happens because in this scenario all nodes are in the same carrier sensing range (the carrier sensing range is about twice as large as the transmission range at 2 Mbps [1]), and thus the 802.11 MAC protocol and TCP does not wrongly

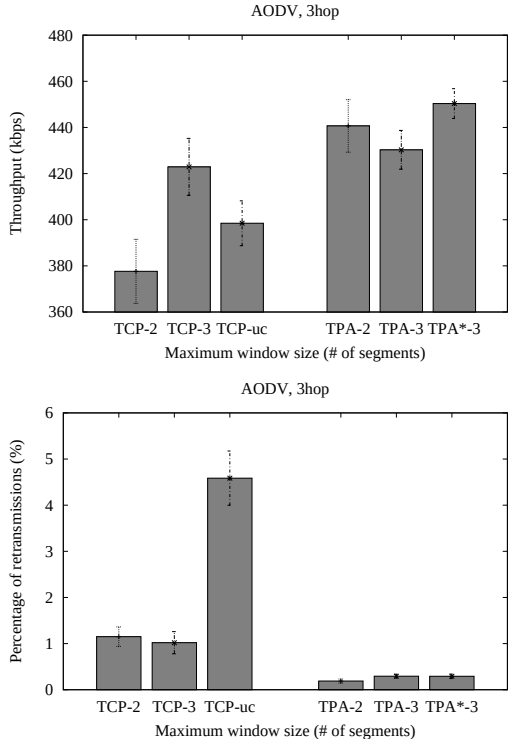

FIGURE 7. Throughput (up) and percentage of retransmitted segments (down) vs. window size in the 3-hop scenario.

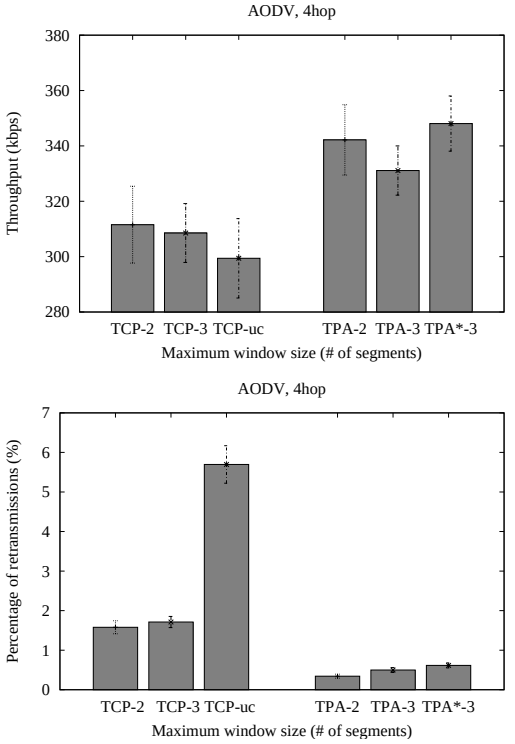

FIGURE 8. Throughput (up) and percentage of retransmitted segments (down) vs. window size in the 4-hop scenario.

interfere, as discussed in Section 2. Also note that the TPA best configuration is *TPA*-3* in terms of throughput, and *TPA-2* in terms of retransmissions. We have observed the same qualitative trend also in the other configurations presented hereafter. In general, the difference between the two configurations is usually not huge. While the difference in terms of retransmission index is very limited, sometimes *TPA*-3* achieves a significantly higher throughput. Therefore this TPA configuration often represents a very good tradeoff between the two performance indices.

By inspecting traffic traces, the main reason for the difference between TCP and TPA performance when all nodes are in the carrier sensing of each other resides in the mechanism of HELLO messages used by AODV to maintain local connectivity [55]. AODV assumes a link failure as soon as two consecutive HELLOs are lost. This is quite frequent, as HELLOs are broadcast packets, and results in frequent route failures. As expected, TPA is much more efficient than TCP in managing these events.

The 3-hop scenario is the first case where problems related to link layer contentions become evident [56], as nodes are not all within the same carrier sensing range anymore. Indeed, the best TCP configuration is *TCP-3* (both as far as throughput and retransmission index). Also in this case, TPA outperforms TCP in terms of throughput (+6.5%) and retransmission index (-71%).

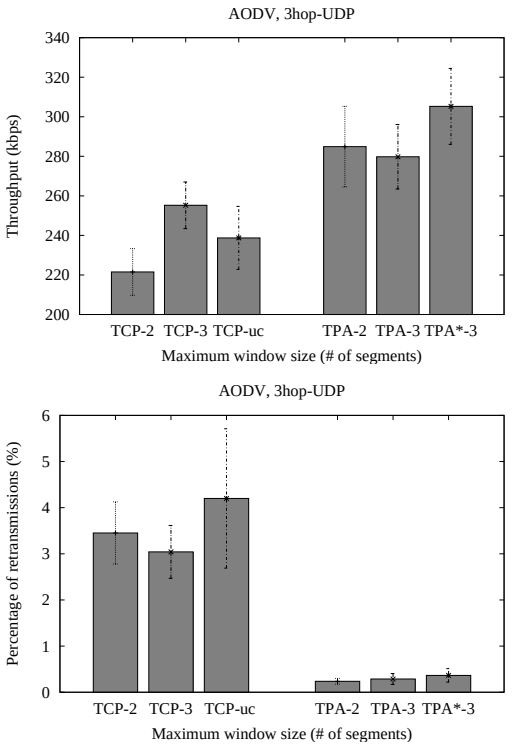

FIGURE 9. Throughput (up) and percentage of retransmitted segments (down) vs. window size in the 3-hop-UDP scenario.

Note that previous *simulation* analyses [31, 30] in this configuration reported that the best TCP configuration for TCP should be *TCP-2*. By inspecting traces, we found that this discrepancy resides in a different behaviour between the TCP implementation in the Linux kernel, and the TCP implementation available in the ns-2 simulator [57]. When the maximum *cwnd* size is set to 2 segments, the simulated TCP receiver sends back one ACK *every other* segment, while the real (i.e. Linux) TCP receiver sends back one ACK *every segment*, as if the delayed ACK mechanism were disabled (see [55] for the details).

In the 4-hop scenario, the link layer contention increases and then the difference in performance between the two protocols becomes more evident. *TCP-2* is now the best TCP configuration (with respect to both performance indices), and significantly improves *TCP-uc* (most notably, -70% retransmissions). However, TPA is *always* better than *TCP-2* (+12% in terms of throughput and -60% in terms of retransmissions in the best configurations).

Finally, we also evaluate the performance of TCP and TPA in the presence of interfering traffic. To this end we considered the 3-hop network topology described above and added a CBR (Continuous Bit Rate) session, between node *N3* (source) and node *N2* (receiver). The CBR sending rate was 192 kbps, which corresponds to the bit rate of a typical MP3 stream. The results obtained, summarized in Figure 9, show that in this scenario there is no great *qualitative* difference with the results obtained in the 3-hop scenario. One important observation is that the performance improvement of TPA is much more evident because the probability of link layer contentions is now greater: the throughput increase is now about +20%, and the retransmission reduction is about -90%.

4.3.2. Analysis with the OLSR routing protocol

Table 2 and Table 3 summarize the throughput and the retransmission index, respectively, achieved with OLSR. We can see that the results obtained with OLSR are not dissimilar to those obtained with AODV. As soon as problems related to link-layer contention become evident, TPA outperforms TCP both in terms of throughput and retransmission index. Specifically, in the best configurations, the throughput increases range between +5.5% and +11%, while the reduction of retransmissions is between -93% and -98%.

Table 2 and Table 3 also show that TCP with a clamped congestion window is the best configuration for TCP in terms of throughput only in the presence of background traffic. This phenomenon can be explained considering the low percentage of retransmission achieved by *TCP-uc* when OLSR is used. For example, in the 4-hop scenario, *TCP-uc* over OLSR retransmits about 65% less segments that over AODV. The most likely reason for this difference in retransmission index

TABLE 2. Throughput (in kbps) vs. maximum cwnd size with OLSR.

	1hop	2hop	3hop	4hop	3hop UDP
TCP-2	1369 ±25	619 ±14	354 ±19	231 ±18	300 ±13
TPA-2	1464 ±5	711 ±23	412 ±15	259 ±17	340 ±14
TCP-3	1456 ±16	672 ±14	357 ±18	235 ±15	320 ±13
TPA-3	1461 ±19	680 ±14	396 ±14	263 ±11	322 ±13
TCP-uc	1524 ±5	691 ±13	371 ±26	253 ±15	315 ±16
TPA*-3	1495 ±3	730 ±24	411 ±11	267 ±13	347 ±13

TABLE 3. Retransmission index vs. maximum cwnd size with OLSR.

	1hop	2hop	3hop	4hop	3hop UDP
TCP-2	0 ±0	0.02 ±0.01	0.4 ±0.15	1.3 ±0.2	0.67 ±0.24
TPA-2	0 ±0	0 ±0	0.01 ±0.006	0.09 ±0.04	0.02 ±0.02
TCP-3	0 ±0	0.02 ±0.01	0.54 ±0.18	1.38 ±0.24	0.5 ±0.19
TPA-3	0 ±0	0 ±0	0.04 ±0.016	0.15 ±0.05	0.07 ±0.01
TCP-uc	0 ±0	0 ±0	0.77 ±0.24	2 ±0.3	0.96 ±0.4
TPA*-3	0 ±0	0 ±0	0.08 ±0.04	0.16 ±0.05	0.03 ±0.001

is the different parameter values used by AODV and OLSR to manage HELLO messages. Specifically, by considering the default parameter values for both AODV and OLSR, OLSR assumes that a link is broken if it fails to receive *three* consecutive HELLO messages from its neighbour, while AODV assumes a link failure when if fails to receive *two* consecutive HELLO messages. This makes OLSR more robust to false link failures [55]. Finally, Table 2 and Table 3 also show that *TPA*-3* always provides the best throughput for TPA. The only exception is in the 3-hop scenario, where *TPA-2* and *TPA*-3* achieve the same throughput.

4.3.3. Analysis with different transmission rates

To show that TPA improvements do not depend on the transmission rate, we replicated the experiments at 5.5 and 11 Mbps. For the sake of space, only the OLSR results in a 4-hop chain are shown here (Figures 10 and 11, respectively). At 5.5 Mbps, TPA outperforms TCP with optimal window size both in terms of throughput and retransmission index with all congestion window sizes. In the best configurations, the throughput increase is about +10%, and the retransmission reduction about -80%. At 11 Mbps, the difference in throughput between TCP and TPA is less evident (+4%). However, the reduction in terms of retransmission index is still about -80%.

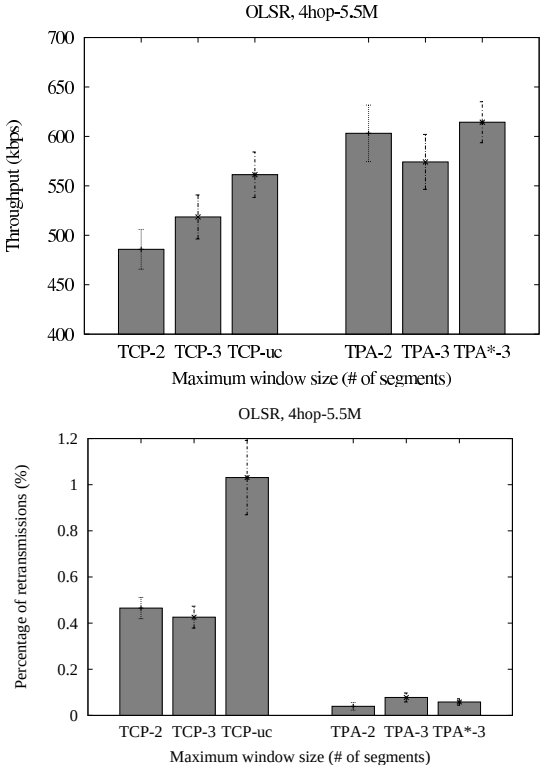

FIGURE 10. Throughput (up) and percentage of retransmitted segments (down) vs. window size in the 4-hop at 5.5 Mbps

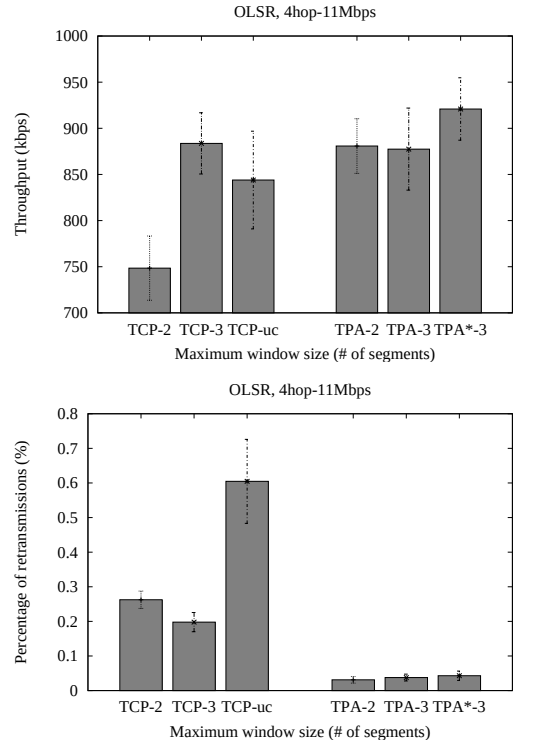

FIGURE 11. Throughput (up) and percentage of retransmitted segments (down) vs. window size in the 4-hop at 11 Mbps

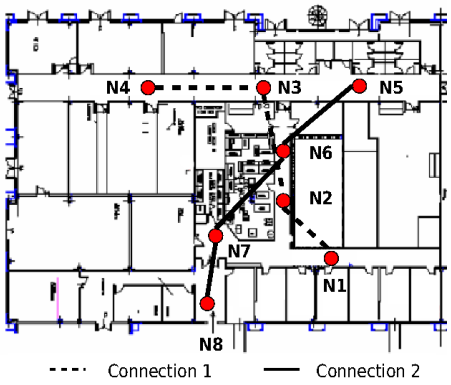

FIGURE 12. Cross Topology network.

4.3.4. Summary of the main results

Before proceeding on with the analysis for more complex scenarios, it is worth summarizing the main results obtained so far in chain topologies. Specifically we have shown that:

- the throughput of TPA (in its best configuration) is always higher than the throughput of TCP (in its best configuration), unless in the 1-hop case. Specifically, the TPA throughput is between 4% and 20% higher than the TCP throughput. We can reasonably expect the difference between TPA and TCP in the one-hop case to disappear if also TPA were implemented in the kernel.
- TPA (in its best configuration) retransmits between 60% and 98% less segments than TCP (in its best configuration) does. This results in lower energy consumption and lower congestion in the network.
- As noted in Section 6, *TPA*-3* is generally the best configuration for TPA. This is because it reduces the number of ACK in transit in the network.
- The best configuration for TCP varies with network topology and routing protocols. However, as soon as contention problems appears, *TCP-3* tends to be the best choice.

Based on the above remarks, in the following we only present results achieved by *TCP-3* and *TPA*-3*.

4.4. Cross topology

In this section we consider a cross topology, which is one of the reference topologies used in the literature [8, 30]. Figure 12 shows the nodes position in our testbed. In this scenario there are two connections, the first one (referred to as connection 1) from node *N1* (sender) to node *N4* (receiver) and the second one (referred to as connection 2) from node *N8* (sender) to node *N5* (receiver). Precisely controlling the interference patterns between nodes of the different chain is not trivial in such a real testbed. We carefully checked that end points of different chains (i.e., node *N1* and

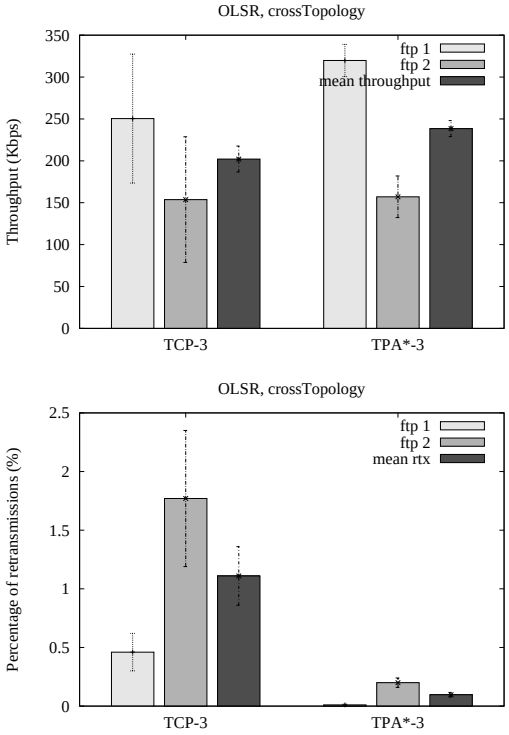

FIGURE 13. Throughput (up) and percentage of retransmitted segments (down) in the cross topology with OLSR.

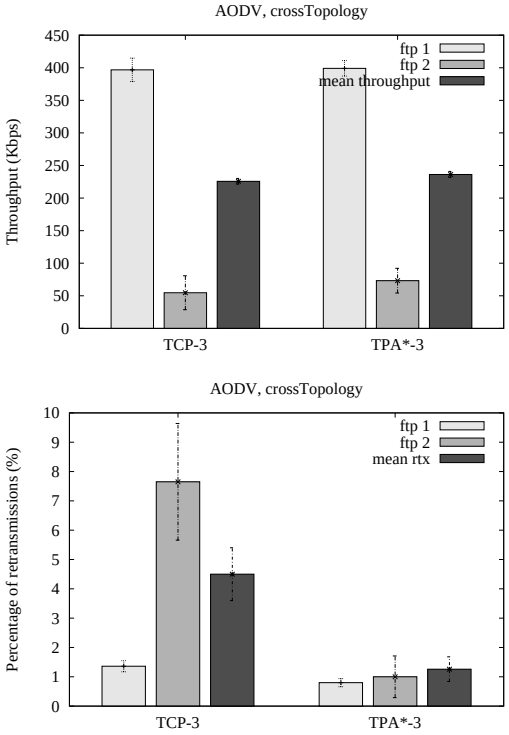

FIGURE 14. Throughput (up) and percentage of retransmitted segments (down) in the cross topology with AODV.

node *N8*, and node *N4* and node *N5*) were outside their respective transmission ranges. Furthermore, as in the single-chain case, we positioned nodes of each chain so that adjacent nodes of a chain were at the limit of the transmission range of each other, and enforced this configuration via `iptables`.

To generate the traffic we used two *ftp-like* connections also in this case. We started both connections at the same time and we made them last for 240 seconds. Hereafter, in addition to the per-connection throughput and retransmission indices defined in Section 4.2, we also show the *mean throughput*, i.e., the mean between the throughput of the two connections, and the *mean retransmission index*, i.e., the mean between the retransmission indices of the two connections.

Figure 13 shows the performance indices achieved over OLSR. Also in the cross topology TPA significantly outperforms TCP both in terms of throughput and retransmission index. Specifically, *TPA*-3* increases the mean throughput of about 18% with respect to *TCP-3* and reduces the mean retransmission index of about 73%.

As far as the TCP results, note that, while the semi-confidence interval of the mean throughput is below 10% of the average value, the per-flow throughput of the separate connections shows a higher variability. This is a by-product of TCP unfairness issues we discuss in detail in the following. Specifically, TCP turned out to suffer so much from unfairness issues in this configuration, than often one of the connections completely starved while the other proceeded. This resulted in very high variability across different replicas, and, in the end, in the large confidence intervals.

When AODV is used (Figure 14), the TPA performance improvement is less evident. However, *TPA*-3* still increases the mean throughput of about 5% with respect to *TCP-3*, and reduces the mean retransmission index of about 60%.

We can observe that both TCP and TPA suffer a severe unfairness between the two connections. This result motivated us to investigate unfairness issues more precisely via simulation, as described in Section 5. However, it is worth pointing out that *both* connections achieve on average higher throughput and lower retransmission indices when TPA is used, even though connection 1 benefits more than connection 2.

4.5. Mobility: the roaming-node scenario

To complete our experimental evaluation of TPA we considered the impact of nodes mobility on the performance of both TPA and TCP. To this end, we replicated the roaming-node scenario defined in [3].

The roaming-node scenario (Figure 15) consists of four nodes. Three of them are stationary (*N1*, *N2*, and *N3*) and one is mobile (*N4*). Nodes *N1*, *N2* and *N3* form a static two hop chain network. The mobility

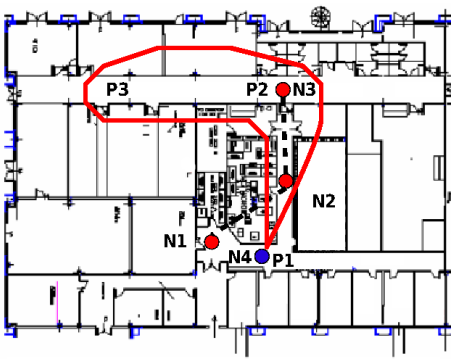

FIGURE 15. Roaming Node Scenario.

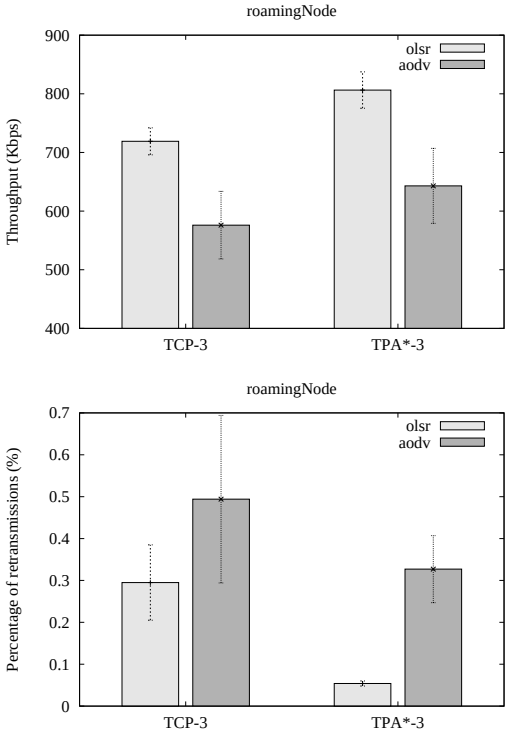

FIGURE 16. Throughput (left) and percentage of retransmitted segments (right) in the roaming node scenario.

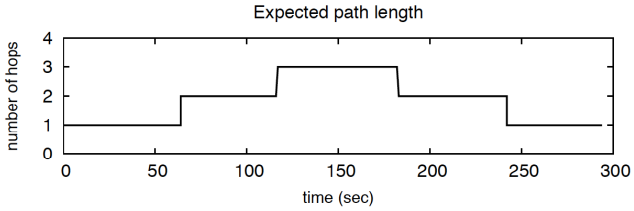

FIGURE 17. Ideal path length during our experiments.

pattern followed by node *N4* is as follows. At time 0 node *N4* is in position P1. After 40 seconds it starts to move toward positions P2, where it pauses for 40 seconds. Then it moves towards position P3, where it pauses for 40 seconds. Finally, it goes back to positions P2 and P1, pausing for 40 seconds in each one. The time needed to move from a position to the next position is about 20 seconds. Thus, the experiments lasted for about 280 seconds. A single ftp-like connection spanned between nodes N1 and N4 for the whole experiments duration. The length of the path between nodes *N1* and *N4* varied during the experiment. Specifically, in positions P1, P2 and P3, node *N4* could reach node *N1* in 1 hop, 2 hops and 3 hops, respectively. It should be noted that in this set of experiments, to allow routes to be dynamically recomputed we did not enforced paths through `iptables`. The roaming-node scenario represents typical human mobility patterns, in which no abrupt differences between path lengths are expected.

Figure 16 shows the throughput and retransmission index of TCP and TPA over both OLSR and AODV. Also in this scenario TPA significantly outperforms TCP both in terms of throughput (+12% both with OLSR and AODV) and retransmission index (-73% with OLSR and -26% with AODV). We can observe that in the case of AODV we do not obtain conclusive results in terms of retransmission index due to the high variability of the experimental results. However, the average values strongly suggest than TPA outperforms TCP also in this case.

Figure 16 also shows that both TCP and TPA work remarkably better over OLSR than over AODV. Specifically, the throughput in the AODV case decreases of about 25% with respect to the OLSR case. This may appear strange, since OLSR suffers from a non negligible *re-route* time [58], due the rules OLSR uses to generate and disseminate the information about network topology (Topology Control messages, see [52] for more details). Despite this drawback, OLSR is able to provide a more stable networking environment in our experiments. This is because AODV assumes that a link exists as soon as a node receives a HELLO or RREQ message from a neighbour. Therefore, it tends to use links that may be unidirectional in the wrong direction [59]. This is quite likely in a mobile configuration like the one we are considering. On the other hand, OLSR just uses links that have proved to be bidirectional. Specifically, in OLSR a link is deemed valid only if both endpoints announce each other in their respective neighbour list [52]. It is easy to show that this is a stronger check on the links' validity, which grants more stable routes.

Based on these remarks, we further investigate the behaviour of TCP and TPA just in the OLSR case. Figure 17 shows the ideal path length between the sender (*N4*) and the receiver (*N1*) that the routing protocol is expected to compute during our experiments. Figure 18 shows the real path length (upper plot) and the throughput (lower plot) achieved by TCP (over OLSR) during a particular replica of our

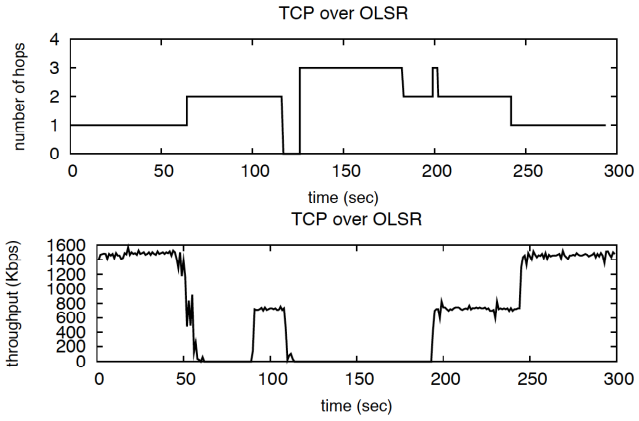

FIGURE 18. Path length and instantaneous throughput of TCP over OLSR in the Roaming Node Scenario.

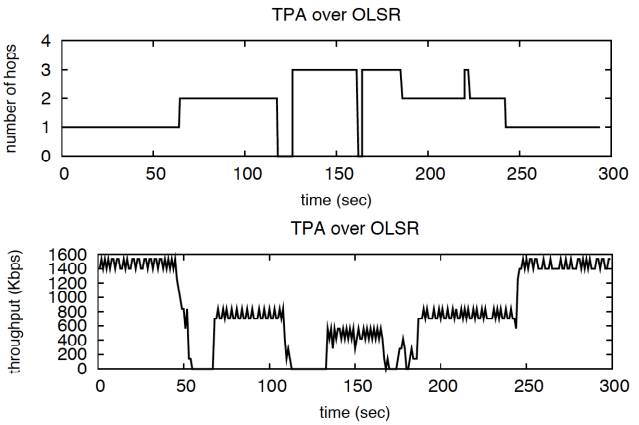

FIGURE 19. Path length and instantaneous throughput of TPA over OLSR in the Roaming Node Scenario.

experiment (other replicas show a similar behaviour). A number of hops equal to zero means that node N4 has no entries for node N1 in its routing table, and is thus not able to communicate with it. Figure 19 shows the same performance figures in the case of TPA. First of all, note that the path calculated by OLSR is comparable in both cases (small differences are due to the unavoidable differences in the external environment conditions experienced by different experiments). Actually, the evolution of the computed path is slightly better in the TCP case, as shown by the additional drop after 150s in Figure 19 (upper plot). As far as the throughput, TCP performs close to TPA only when the sender and the receiver are 1-hop away, i.e., before 50s and after 250s. When they are 2-hops away, TCP is able to transfer some segments, as shown by the plateaux around 100s and between 200s and 250s. However, in these time frames TPA is able to achieve a quite more stable throughput. Finally, when the sender and the receiver are 3-hops away, TCP is completely unable to carry on any transfer, while TPA still achieves a stable behaviour in terms of throughput. Therefore, TPA is able to reconfigure after route changes, and carry on data transfer more efficiently than TCP.

5. SIMULATION ANALYSIS

In the previous section, we reported the experimental results of TPA analysis over simple network topologies. As mentioned in Section 1, this allows us to give significant results in configurations similar to those expected in real ad hoc networks [7]. To complete the analysis, we considered a further set of experiments in a simulation environment using ns-2 [57]. The higher control over the environment parameters allows us to better investigate i) the scalability properties of TPA and ii) unfairness issues. As far as the latter aspect, we considered both the original TCP and TPA protocols, as well as their variants that include the adaptive pacing algorithm [8] described in Section 5.2 (these variants are called TCP-AP and TPA-AP respectively). This allows us to highlight the better performance of TPA also when adaptive pacing is not used, and to show, at the same time, the improvements brought about by adaptive pacing.

5.1. Simulation scenarios and performance measures

We focused on five scenarios. The first two are the standard cross and parallel topologies defined in [8]. This allows us to compare TCP-AP and TPA-AP in the same configurations used in [8]. The next two are a grid topology and a static random topology. The final one is a mobile scenario in which 50 nodes move according to the random waypoint (RWP) model. These scenarios allows us to jointly investigate scalability and fairness properties of TPA. We used AODV-UU [51] as the routing protocol implementation.

As far as TCP and TPA settings, we used the same configurations used in Section 4. For TCP-AP we only used an unclamped congestion window size, as suggested in [8].

As in all scenarios we have more connections running concurrently, we slightly modified the performance figures defined in Section 4.2. Specifically we considered:

- *Aggregate Throughput*, i.e., the sum of the throughputs achieved by each connection.
- *Mean Re-transmission index*, i.e., the percentage of segments re-transmitted by all the TCP/TPA senders, computed as

$$rtx = \frac{\sum_{i=1}^{n} rtx_i \cdot pktRcvDst_i}{\sum_{i=1}^{n} pktRcvDest_i}$$

where rtx_i is the retransmission index of the i-th connection, $pktRcvDest_i$ is the number of non-duplicated segments successfully received in the i-th connection, and n is the number of connections active in the network.

- *Jain's fairness index* [60], defined as:

$$F(x) = \frac{[\sum_{i=1}^{n} x_i]^2}{n \times \sum_{i=1}^{n} x_i^2},$$

where x_i is the throughput achieved by the i-th connection. In addition, we also considered the *instantaneous fairness* index, i.e., the value of the Jain's fairness index sampled every two seconds, and then averaged over the whole experiment. While the fairness index provides an indication of the long-term fairness, the instantaneous fairness tells how fair is a protocol over short-time scales. In both cases, the higher the index, the fairer the protocol.

Unless otherwise stated, we replicated each experiments 10 times. This allowed us to reach – in general – a semi-confidence interval of performance figures within 10% of the average value. As in the case of experimental measurements, the retransmission index was more variable than the other performance figures, and we thus achieved larger confidence intervals. However, the same remarks already pointed out in Section 4.2 apply also in this case. Finally, also a few semi-confidence intervals of the TCP fairness index (in the parallel and grid topologies) were larger than 10% (but always below 16%). However, also in those few cases the difference between TCP and TPA is large enough to allow us to fairly compare them despite the slightly larger variability.

Before presenting the results, it is worth recalling how the adaptive pacing algorithm works (the reader is referred to [8] for the complete description).

5.2. Adaptive pacing

Several papers have analysed the TCP unfairness problem over MANETs (see, for example, [8, 39]). Channel capture, hidden and exposed terminal conditions, and the binary exponential backoff of the IEEE 802.11 MAC are the main causes of TCP unfairness. Moreover, TCP's congestion control mechanism exacerbates the problem. Specifically, as noted in [8], TCP's window-based congestion control mechanism leads to segments' burst on ACK reception. This produces an increased channel contention and reduces the chance of neighbour nodes to access the channel. In [8], ElRakabawy et al. thus proposed a modified version of the adaptive pacing (AP) mechanism available for the Internet, and applied it to TCP. They show that TCP-AP can significantly mitigate the unfairness problem encountered by TCP.

To mitigate the segments' burst problem, AP spreads the segments transmission according to a rate that is dynamically computed. It incorporates a mechanism to identify incipient congestion and to adjust consequently the transmission rate. In more detail, AP calculates the segments transmission rate taking into account the spatial reuse constraint of IEEE 802.11 multi-hop network. The authors of [30] showed that, in a chain topology, only nodes 4 hops away from each other can transmit simultaneously. Based on this result, the authors of [8] use the *4-hop propagation delay (FHD)*, i.e. the time needed for a segment produced by the i-th node in the path to reach "node *i+4*", to calculate the segment transmission rate. In AP the sender calculates the *FHD* using the RTT estimation and the number of hops of the connection. The second ingredient used by AP is an estimate of the link-layer contention, computed as the *coefficient of variation of recently measured RTT (cov_{RTT})*:

$$cov_{RTT} = \frac{\sqrt{\frac{1}{N-1} \times \sum_{i=1}^{N} \left(RTT_i - \overline{RTT}\right)^2}}{\overline{RTT}}. \quad (1)$$

In Equation (1) N is the number of RTT samples, $\overline{RTT}$ is the mean of the samples, and RTT_i is the value of the *i-th* sample of RTT. AP evaluates the transmission rate R as follows:

$$R = \frac{1}{\overline{FHD} \times (1 + 2cov_{RTT})}, \quad (2)$$

where $\overline{FHD}$ is the standard exponentially weighted moving average[3] of *FHD* and $(1 + 2cov_{RTT})$ is the factor that takes into account the contention degree of the network. The transmission rate depends on the *FHD* index, and on the link-layer contention. Thus, the connection slows down when the contention increases.

5.3. Cross and parallel topologies

The cross and parallel topologies (Figure 20) are two reference topologies considered in [8]. In both scenarios, the distance between adjacent nodes is 200 meters, so as to make each connection 4-hops long. In the parallel topology the distance between the two chains is 400 meters. This way, nodes of connection 1 (connection 2) are out of the transmission range of nodes of connection 2 (connection 1), but inside the carrier sensing range of connection 2 (connection 1). In both topologies we considered two *ftp* flows, each starting at time 20 and lasting for 500 seconds.

Table 4 shows the results of the experiments in the Cross Topology scenario. TPA outperforms TCP in terms of throughput (+6%) and instantaneous fairness (+4%), while it's essentially equivalent in terms of fairness, and slightly worse in terms of retransmissions (+2%). Table 5 shows the results of the experiments in the Cross Topology scenario when the adaptive

[3] In the simulative analysis, we set *alpha* to 0.7 to compute $\overline{FHD}$.

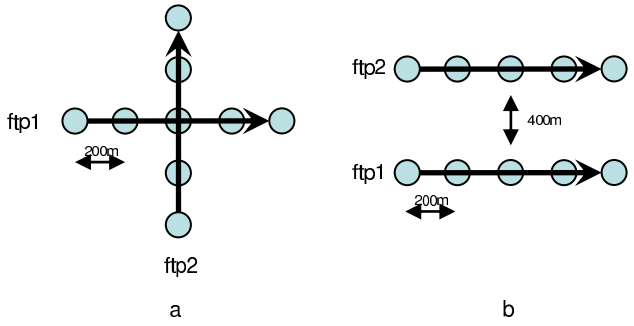

FIGURE 20. Cross and Parallel topology.

TABLE 4. Cross Topology

	Agg. Thr (kbps)	Fair. (%)	Ist. Fair. (%)	Mean Rtx (%)
TCP-2	292 ±1.4	94.6 ±3	63.6 ±1.7	5.13 ±0.49
TPA-2	301.7 ±1.3	98.9 ±0.7	69.6 ±1.3	5.23 ±0.24
TCP-3	282.5 ±2	95 ±4	69.9 ±1.9	8.3 ±0.6
TPA-3	297 ±1.6	99.7 ±0.2	72.4 ±1.25	7.35 ±0.26
TCP-uc	280.7 ±1.7	99.3 ±0.4	70.8 ±1.13	11.4 ±0.43
TPAst	309.5 ±1.3	99.3 ±0.36	73.3 ±1.24	7.22 ±0.3

TABLE 5. Cross Topology with adaptive pacing enabled

	Agg. Thr. (kbps)	Fair. (%)	Ist. Fair (%)	Mean Rtx (%)
TCP-AP	153 ±2.5	99.7 ±0.14	92.3 ±0.1	6.34 ±0.36
TPA-AP-2	254.7 ±2.3	99.3 ±0.7	87.3 ±2.2	2.46 ±0.14
TPA-AP-3	250.6 ±2.7	99.1 ±0.8	86.8 ±1.9	3.3 ±0.16
TPA*-AP-3	250 ±1.4	98.2 ±0.98	88.4 ±1.5	5.8 ±0.28

pacing mechanism is used. Adaptive pacing improves the *instantaneous fairness* of both protocols (+30% for TCP, +21% for TPA). This is due to sender rate limitation, which is paid by TCP with a drastic reduction of the aggregate throughput (-48%), and an increase of the retransmission index (+24%). This is clearly not the case for TPA: the throughput reduction is only -17%, and the retransmission index is *reduced* by 53%. The direct comparison between TCP-AP and TPA-AP tells that TPA-AP achieves significantly greater throughput (+66%) and lower retransmission index (-61%), at the cost of an acceptable reduction of fairness (less than 1%) and instantaneous fairness (-4%).

There is actually a trade-off between the performance in terms of throughput and the instantaneous fairness, which is controlled by the RTT estimate in Equation (1). It can be shown that TCP tends to greatly overestimate RTT, thus drastically reducing the sender's transmission rate. Clearly, this improves the fairness, but dramatically impacts on the throughput. A TPA sender is more aggressive because the RTT is more accurate (this has been noted by inspecting the simulation traces). It thus achieves greater throughput, at the cost of a limited reduction of the instantaneous fairness. By slightly modifying Equation (1) as in Equation (3) (i.e., scaling RTT by an integer number K), it is possible to tune the adaptive pacing so as to slightly reduce the TPA throughput, and achieve the same fairness of TCP.

TABLE 6. Cross Topology. Impact of the K parameter on TPA.

	$K=1$	$K=2$	$K=3$
Agg. Thr. (kbps)	254.7 ±2.3	208.7 ±3.8	171.2 ±1.1
Mean Rtx (%)	2.46 ±0.14	1.83 ±0.14	1.05 ±0.11
Fair. (%)	99.3 ±0.7	99.7 ±0.18	99.8 ±0.08
Ist. Fair. (%)	87.3 ±2.2	93.7 ±1.09	96.2 ±0.3

TABLE 7. Parallel Topology

	Agg. Thr. (kbps)	Fair. (%)	Ist. Fair (%)	Mean Rtx (%)
TCP-2	292.2 ±1.5	83.1 ±11	54.9 ±1.5	2.9 ±0.5
TPA-2	298.9 ±2.1	97 ±2	58.1 ±1.03	2.4 ±0.15
TCP-3	283.3 ±1.6	95.7 ±3	57.9 ±1.44	4.5 ±0.5
TPA-3	294.7 ±2.1	98.1 ±1.6	60 ±1	3.4 ±0.19
TCP-uc	279.5 ±2.5	82.3 ±13	56 ±2.4	6.4 ±0.7
TPAst	308.2 ±1.9	97.7 ±1	58.8 ±0.62	3.14 ±0.16

$$cov_{RTT} = \frac{\sqrt{\frac{1}{N-1} \times \sum_{i=1}^{N} \left(\mathbf{K} \times RTT_i - \overline{RTT}\right)^2}}{\overline{RTT}}, \quad (3)$$

As an example, Table 6 shows the performance of TPA for K equal to 1, 2 and 3. For simplicity, only the results obtained with a *cwnd* equal to 2 are reported (the other TPA-AP configurations achieve similar results). When K is equal to 2, TPA-AP improves the *instantaneous fairness* of TCP-AP of about 1.5%, maintaining the throughput 36% higher, and retransmitting 71% less segments.

Table 7 shows the results obtained over the parallel topology when the adaptive pacing mechanism is not used. In this scenario TPA outperforms TCP in terms of throughput (+6%), retransmissions (-17%), fairness (+2.5%) and instantaneous fairness (+3.6%).

TABLE 8. Parallel Topology with adaptive pacing enabled

	Agg. Thr. (kbps)	Fair. (%)	Ist. Fair (%)	Mean Rtx (%)
TCP-AP	155.5 ±4.2	99.8 ±0.1	90.8 ±0.6	7.3 ±0.4
TPA-AP-2	215.3 ±0.9	99.5 ±0.35	87.8 ±1.2	1.7 ±0.16
TPA-AP-3	208.3 ±2.5	99.6 ±0.5	87.7 ±1.6	2 ±0.2
TPA*-AP-3	193.2 ±3.7	97.5 ±1.5	88.4 ±2.2	2.5 ±0.12

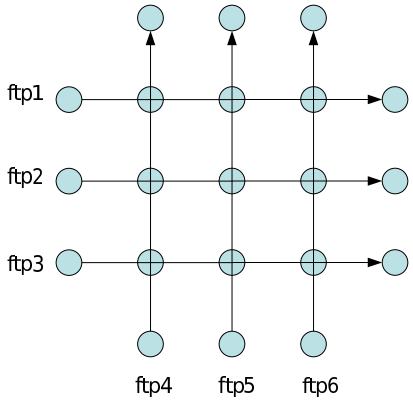

FIGURE 21. Grid topology.

TABLE 9. Grid Topology

	Agg. Thr. (kbps)	Fair. (%)	Ist. Fair. (%)	Mean. Rtx. (%)
TCP-2	270.1 ±3	62.6 ±5	35.4 ±2	12.3 ±1
TPA-2	280 ±2.9	89.2 ±1.8	47.5 ±0.4	15 ±0.34
TCP-3	263.9 ±4.2	66.5 ±8.6	35.9 ±2	13.7 ±0.8
TPA-3	278 ±2	89.8 ±1.73	48.1 ±0.6	18.9 ±0.2
TCP-uc	268 ±2.7	65.3 ±8	34.4 ±2.6	15.9 ±1.3
TPAst	283 ±2	88.2 ±2	47.5 ±0.6	16.4 ±0.4

TABLE 10. Grid Topology with adaptive pacing enabled

	Agg. Thr. (kbps)	Fair. (%)	Ist. Fair. (%)	Mean. Rtx. (%)
TCP-AP	202.8 ±2.6	98.6 ±0.9	86.1 ±1.5	28.6 ±1
TPA-AP-2	239 ±2.3	96.1 ±0.9	85.8 ±1	7.5 ±0.4
TPA-AP-3	241 ±1.1	96.5 ±1.7	84.9 ±1.3	8.4 ±0.26
TPA*-AP-3	233.7 ±1.8	95.3 ±1.4	85.5 ±1.6	7.2 ±0.4

Table 8 shows the results when the adaptive pacing mechanism is used. Again, the instantaneous fairness of both protocols increases (+56% for TCP, +47% for TPA), and TPA-AP outperforms TCP-AP in terms of throughput (+39%) and retransmissions (-77%), with a small reduction of the instantaneous fairness (-3%).

In general, the above results show that i) even without adaptive pacing, TPA outperforms TCP also in terms of fairness; ii) the adaptive pacing mechanism is required by both protocol to achieve higher fairness, and iii) TPA-AP greatly outperforms TCP-AP in terms of throughput, with a small reduction of the instantaneous fairness. These outcomes are confirmed also in the more complex topologies that we consider in the final part of the paper.

5.4. Grid topology

This section reports the results of the simulation analysis of TPA over the grid topology depicted in Figure 21. It is made up of 25 nodes in a 5 x 5 grid, where the distance between horizontal and vertical adjacent nodes is 200 meters. Over this topology, we set up 6 *ftp* flows, each of wich is 4-hops long. Each ftp flow starts at time 20 and lasts for 500 seconds.

Tables 9 and 10 show the performance of TCP and TPA with and without adaptive pacing. When adaptive pacing mechanism is not enabled, TPA outperforms TCP - as in the other configurations - in terms of aggregate throughput (+5%), fairness (+35%) and instantaneous fairness (+34%). However, this is paid with an increment of the retransmission index (+22%).

When adaptive pacing is enabled (Table 10) we see the usual pattern highlighted before. The instantaneous fairness of both protocols is increased (+140% for TCP, +78% for TPA). TPA-AP outperforms TCP-AP with respect to the throughput (+19%) and retransmission index (-75%), with a slight reduction of the fairness (-2%).

5.5. Static Random Topology

In order to test TPA performance in a less rigid scenario, we considered a random topology of 50 nodes randomly distributed in an area A=1000m x 1000m. We considered a variable number of connections simultaneously active in the network (3, 5, and 10). Each connection starts at time 20s, and lasts for 500s. For each number of concurrent connections, we generated 10 random nodes' placements, and simulations with each placement were replicated 10 times.

The communication end points of each connection were fixed for all nodes' placements (i.e. we chose randomly the source-destination pairs, and we used the same choice for all the considered scenario). Performance figures of experiments run over different placements are not identically distributed, and therefore it is not meaningful to compute confidence intervals. We, however, averaged the performance indices over

TABLE 11. Static Random Topology

		Number of connections		
		3	5	10
TCP-uc	A. Thr.	885.1	1126	1303
	I. Fair.	46.8	40.3	28.9
TPA*-3	A. Thr.	827.2	1108	1195
	I. Fair.	56.7	46.5	34
TCP-2	A. Thr.	831.3	1121	1226
	I. Fair.	52.98	42.6	31.3
TPA-2	A. Thr.	801	1080	1173
	I. Fair.	58.3	46.9	34.6

TABLE 12. Static Random Topology with the adaptive pacing enabled

		Number of connections		
		3	5	10
TCP-AP	A. Thr.	415.9	546.7	558.8
	I. Fair.	71.4	66.7	59
	M. Rtx	7.4	9.5	13.4
TPA-AP-2	A. Thr.	476.6	621.8	641.4
	I. Fair.	75.3	68.9	57.9
	M. Rtx	1.7	2.2	3.5

the 10 replicas for each placement, and over the 10 placements.

This methodology does not allow us to quantitatively compare performance figures with a sufficient statistical confidence. However, the results generally confirm the different behaviour of TPA and TCP highlighted in the previous sections, with special regard to the different effects of activating the Adaptive Pacing mechanism. The general indication is that, when Adaptive Pacing is used (Table 12) both protocols achieve a lower throughput and a higher instantaneous fairness with respect to the case when Adaptive Pacing is not used (Table 11, which only reports the results for the best configuration of both TCP and TPA i.e. *TCP-uc* and *TPA*-3* as far as the throughput, *TCP-2* and *TPA-2* as far as the fairness). However, the throughput decrease is lower for TPA, which also achieves performance comparable to that of TCP in terms of instantaneous fairness. In other words, the indication is that also in this case Adaptive Pacing applied to TPA results, in comparison with TCP, in equivalent performance in terms of fairness, with lower reduction in terms of throughput.

Another general indication we can draw is that in this case the instantaneous fairness with adaptive pacing is quite lower than in previous configurations for both protocols. This is due to the fact that in the previous topologies the connections spanned the same number of hops, and were thus homogenous. When connections span a different number of hops, even if adaptive pacing is used, shorter connections get a higher throughput with respect to longer ones.

5.6. Mobile Scenario

The considered mobile network consisted of 50 nodes moving over a 1000m x 1000m field. We utilized

TABLE 13. Mobile Scenario

	Agg. Thr. (kbps)	Fair. (%)	Ist. Fair. (%)	Mean. Rtx. (%)
TCP-uc	864.8	75	38.2	5.8
TCP-2	837.4	77	43.9	4.8
TPA-3*	871.7	78.5	45.3	3.6

TABLE 14. Mobile Scenario with the adaptive pacing enabled

	Agg. Thr. (kbps)	Fair. (%)	Ist. Fair. (%)	Mean. Rtx. (%)
TCP-AP	433.2	94	69.8	16
TPA-AP*-3	536.2	91.2	67.1	3.9
TPA-AP-2	523.7	92.8	68.4	3.4

50 different mobility patterns based on the random waypoint model, each one being generated according to the perfect simulation paradigm [61, 62]. To mimic high node mobility, nodes' speed was uniformly sampled in the interval between 1 to 9 m/s, and the pause time was 20s. After a 20s warm-up period, 5 TCP/TPA connections were established, and a *ftp-like* transfer was launched on each of them. File transfers stopped after 500s. We calculated our performance metrics as the mean over the 50 considered patterns.

As in the case of the Static Random Topology discussed in Section 5.5, we chose randomly the source-destination pairs for each connection, and we used the same pairs for all the considered mobility patterns. Also in this case, experiments run over different patterns are not identically distributed, and it is thus not meaningful to compute confidence intervals.Nevertheless, we hereafter present the average performance figures over the different mobility patterns, which are able to provide trends about the TPA and TCP behaviour also in this setup (see Tables 13 and 14). Once again, using the Adaptive Pacing mechanisms reduces the throughput and increases the instantaneous fairness. However, in TPA the throughput degradation is lower than in TCP, while the instantaneous fairness is equivalent.

Note that also in this case (as in the static random topology), the instantaneous fairness provided by adaptive pacing is lower than in configurations in which the path length of the connections is homogeneous.

6. SUMMARY AND CONCLUSIONS

In this paper we have presented and evaluated TPA, which is a new transport protocol designed to cope with the main inefficiencies of TCP over multi-hop ad hoc networks. TPA includes by design a number of features that have shown to be required to adapt TCP to ad hoc networks. The original approach of TPA is blending together these features since the design stage, rather than proposing and evaluating single modifications to the original TCP in isolation.

In this work we have provided a thorough evaluation and comparison of TCP and TPA in a real ad hoc network testbed. We have investigated the impact of different protocol parameters on the throughput and the number of segments used to sustain the throughput. We have run experiments on different topologies, different routing protocols, and also in mobile scenarios. In all the cases we have investigated TPA is able to improve the performance of TCP. Specifically, TPA delivers greater throughput with respect to TCP (up to 20% increase), while reducing *at the same time* the number of transmissions (up to 98% reduction). To complement results from the testbed, we have also presented a detailed simulation analysis to investigate the TPA performance in terms of fairness, and its scalability properties. We have considered different topologies, both in static and mobile configurations. We have found that, when the adaptive pacing algorithm is included both in TPA and in TCP, TPA achieves the same TCP fairness, while providing a drastic improvement in terms of average throughput (up to +66%), and a drastic reduction in terms of retransmissions (up to -77%).

The results presented in this paper are tailored to multi-hop ad hoc networks in which groups of users set up a stand-alone network and exchange data in a p2p fashion. The results we have obtained motivate us to further investigate the TPA performance also in different setups. A first interesting area for further studies is a thorough analysis of TPA and TCP interoperability and integration. First of all, it will be interesting to highlight the performance of TCP and TPA connections running concurrently over the same network. Secondly, ways should be identified to make TPA-enabled nodes work with legacy TCP nodes. From this standpoint, a straightforward solution could exploit the fact that all major operating systems come with a complete TCP implementation. Therefore the choice between TCP and TPA could be done while opening a connection. Specifically, a TPA node opening a connection (via the `tpa_connect()` function, see Appendix A) can send a TCP SYN segment announcing TPA availability (e.g., by using a reserved bit in the TCP header). If the other endpoint does not implement TPA, the sender will revert to a standard TCP connection. More refined strategies trying to use TPA features even when just a single endpoint implements TPA are under investigation. Another area of future studies is understanding how TPA works in mixed wireless/wired scenarios. In these cases two options could be compared, namely rely on slight TPA modifications to make it work with unmodified TCP endpoints, or using an Indirect-TCP approach, and thus envisioning a first TPA trunk between the wireless node and the gateway to the wired network, and a standard TCP trunk in the wired network. This naturally leads to investigate the viability of TPA in mesh network environments.

REFERENCES

[1] Anastasi, G., Borgia, E., Conti, M., Gregori, E., and Passarella, A. (2005) Understanding the Real Behavior of Mote and 802.11 Ad hoc Networks: an Experimental Approach. *Pervasive and Mobile Computing*, **1**, 237–256. Special Issue on Performance Evaluation of Wireless Networks.

[2] Kotz, D., nad R. Gray nad J. Liu, C. N., Yuan, Y., and Elliot, C. (2004) Experimental Evaluation of Wireless Simulation Assumptions. *Proceedings of the ACM/IEEE International Symposium on Modeling, Analysis and Simulation of Wireless and Mobile System*, Venice (Italy), October 4–6, pp. 221–231. ACM Press.

[3] Lundgren, H., Nordstrom, E., and Tschudin, C. (2002) Coping with communication gray zones in ieee 802.11b based ad hoc networks. *WOWMOM '02: Proceedings of the 5th ACM international workshop on Wireless mobile multimedia*, New York, NY, USA, pp. 49–55. ACM Press.

[4] Kurkowski, S., Camp, T., and Colagrosso, M. (2005) MANET Simulation Studies: The Incredibles. *ACM Mobile Computing and Communication Review*, New York, NY, USA, October, pp. 49–55. ACM Press.

[5] Conti, M. and Giordano, S. (2005) Multihop Ad Hoc Networking: The Theory. *IEEE Communications Magazine*, **45**, 78–86.

[6] Conti, M. and Giordano, S. (2005) Multihop Ad Hoc Networking: The Reality. *IEEE Communications Magazine*, **45**, 88–95.

[7] Gunningberg, P., Lundgren, H., Nordstroem, E., and Tschudin, C. (2005) Lessons from Experimental MANET Research. *Ad Hoc Networks Journal*, **3**, 221–233.

[8] ElRakabawy, S. M., Klemm, A., and Lindemann, C. (2005) Tcp with adaptive pacing for multihop wireless networks. *MobiHoc '05: Proceedings of the 6th ACM international symposium on Mobile ad hoc networking and computing*, New York, NY, USA, May, pp. 288–299. ACM Press.

[9] Hanbali, A. A., Altman, E., and Nain, P. (2005) A survey of tcp over ad hoc networks. *Communications Surveys & Tutorials, IEEE*, **7**, 22–36.

[10] Holland, G. and Vaidya, N. (2002) Analysis of tcp performance over mobile ad hoc networks. *Wireless Networks*, **8**, 275–288.

[11] Anantharaman, V., Park, S.-J., Sundaresan, K., and Sivakumar, R. (2004) TCP Performance over Mobile Ad-hoc Networks: A Quantitative Study. *Wireless Communications and Mobile Computing Journal (WCMC)*, **4**, 203–222. Special Issue on Performance Evaluation of Wireless Networks.

[12] Dyer, T. D. and Boppana, R. V. (2001) A comparison of tcp performance over three routing protocols for mobile ad hoc networks. *MobiHoc '01: Proceedings of the 2nd ACM international symposium on Mobile ad hoc networking & computing*, New York, NY, USA, pp. 56–66. ACM Press.

[13] Ahuja, A., Agarwal, S., Sing, J., and Shorey, R. (2000) Performance of TCP over Different Routing Protocols in Mobile Ad Hoc Networks. *Proceedings of the IEEE Vehicular Technology Conference (VTC 2000)*, Tokyo,

Japan, May, pp. 2315–2319. IEEE Computer Society Press.

[14] Chandran, K., Raghunathan, S., Venkatesan, S., and Prakash, R. (2001) A Feedback Based Scheme for Improving TCP Performance in Ad Hoc Wireless Networks. *IEEE Personal Communication Magazine*, **8**, 34–39. Special Issue on Ad Hoc Networks.

[15] Fu, Z., Meng, X., and Lu, S. (2002) How Bad TCP Can Perform in Mobile Ad Hoc Networks. *Proceedings of the IEEE Symposium on Computers and Communications (ISCC 2002)*, Taormina-Giardini Naxos (Italy), July, pp. 298–303. IEEE Computer Society Press.

[16] Kim, D., Toh, C., and Choi, Y. (June 2001) TCP-BuS: Improving TCP Performance in Wireless Ad Hoc Networks. *J. Commun. and Net.*, **3**, 175–186.

[17] Liu, J. and Singh, S. (2001) ATCP: TCP for Mobile Ad Hoc Networks. *IEEE Journal on Selected Areas in Communications*, **19**, 1300–1315.

[18] Ramakrishnan, K., Floyd, S., and Black, D. (2001). The Addition of Explicit Congestion Notification (ECN) to IP. RFC 3168.

[19] Wang, F. and Zhang, Y. (2002) Improving tcp performance over mobile ad-hoc networks with out-of-order detection and response. *MobiHoc '02: Proceedings of the 3rd ACM international symposium on Mobile ad hoc networking & computing*, New York, NY, USA, June, pp. 217–225. ACM Press.

[20] Fu, Z., Greenstein, B., Meng, X., and Lu, S. (2002) Design and implementation of a tcp-friendly transport protocol for ad hoc wireless networks. *ICNP '02: Proceedings of the 10th IEEE International Conference on Network Protocols*, Washington, DC, USA, November, pp. 216–225. IEEE Computer Society.

[21] Goff, T., Abu-Ghazaleh, N., Phatak, D., and Kahvecioglu, R. (2003) Preemptive routing in ad hoc networks. *J. Parallel Distrib. Comput.*, **63**, 123–140.

[22] Klemm, F., Ye, Z., Krishnamurthy, S. V., and Tripathi, S. K. (2005) Improving tcp performance in ad hoc networks using signal strength based link management. *Ad Hoc Networks*, **3**, 175–191.

[23] Kopparty, S., Krishnamurthy, S. V., Faloutsos, M., and Tripathi, S. K. (2002) Split tcp for mobile ad hoc networks. *Global Telecommunications Conference, 2002. GLOBECOM '02.*, Taipei, Taiwan, November, pp. 138–142 vol.1. IEEE Press.

[24] He, Q., Cai, L., Shen, X. S., and Ho, P. (2006) Improving tcp performance over wireless ad hoc networks with busy tone assisted scheme. *EURASIP Journal on Wireless Communications and Networking*, **2006**, Article ID 51610, 11 pages. doi:10.1155/WCN/2006/51610.

[25] Lim, H., Xu, K., and Gerla, M. (2003) TCP Performance over Multipath Routing in Mobile Ad hoc Networks. *in Proceedings of the IEEE International Conference on Communications (ICC'03)*, May 11–15, pp. 1064–1068. IEEE Press.

[26] Ng, P. C. and Liew, S. C. (2005) Re-routing Instability in IEEE 802.11 Multi-hop Ad-hoc Networks. *Ad Hoc and Sensor Wireless Networks*, **1**, 01–25.

[27] Xu, S. and Saadawi, T. (2002) Revealing the problems with 802.11 medium access control protocol in multi-hop wireless ad hoc networks. *Computer Networks*, **38**, 531–548.

[28] Xu, S. and Saadawi, T. (2001) Performance evaluation of tcp algorithms in multi-hop wireless packet networks. *Wireless Communications and Mobile Computing*, **2**, 85–100.

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

[30] Fu, Z., Zerfos, P., Luo, H., Lu, S., Zhang, L., and Gerla, M. (2003) The Impact of Multi-hop Wireless Channel on TCP Throughput and Loss. *Proceedings of IEEE INFOCOM 2003*, San Francisco (California), March 30–April 3, pp. 1744–1753. IEEE Computer Society Press.

[31] Chen, K., Xue, Y., Shah, S., and Nahrstedt, K. (2004) Understanding Bandwidth-Delay Product in Mobile Ad Hoc Networks. *Computer Communications*, **27**, 923–934. Special Issue on Performance Evaluation of Wireless Networks.

[32] Papanastasiou, S. and Ould-Khaoua, M. (2004) TCP Congestion Window Evolution and Spatial Reuse in MANETs. *Journal of Wireless Communications and Mobile Computing*, **4**, 669–682.

[33] Nahm, K., Helmy, A., and Kuo, C.-C. (2005) TCP over Multi-hop 802.11 Networks: issues and Performance Enhancement. *Proceedings of ACM MobiHoc*, Urbana-Champaign, IL, USA, June, pp. 277–287. ACM Press.

[34] Altman, E. and Jimenez, T. (2003) Novel Delayed ACK Techniques for improving TCP Performance in Multi-hop Wireless Networks. *Proceedings of the IFIP International Conference on Personal Wireless Communications (PWC 2003)*, Venice, Italy, September 23–25, pp. 237–250. LNCS.

[35] de Oliveira, R. and Braun, T. (2005) A Dynamic Adaptive Acknowledgment Strategy for TCP over Multi-hop Wireless Networks. *Proceedings of IEEE Infocom 2005*, Miami, USA, March, pp. 1863–1874. IEEE Computer Society Press.

[36] Cordeiro, C. D. A., Das, S. R., and Agrawal, D. P. (2002) Copas: dynamic contention-balancing to enhance the performance of tcp over multi-hop wireless networks. *Computer Communications and Networks, 2002. In Proceedings of IC3N'02*, Miami, FL, USA, Oct. 14–16, pp. 382–387.

[37] Floyd, S. and Jacobson, V. (1993) Random early detection gateways for congestion avoidance. *IEEE/ACM Trans. Netw.*, **1**, 397–413.

[38] Tang, K. and Gerla, M. (1999) Fair Sharing of MAC under TCP in Wireless Ad hoc Networks. *Proceedings of IEEE MMT'99*, Venice, Italy, Oct.

[39] Xu, K. and Gerla, M. (2005) TCP Unfairness in Ad Hoc Wireless Networks and a Neighborhood RED Solution. *Wireless Networks*, **11**, 383–399.

[40] Xu, K., Bae, S., Lee, S., and Gerla, M. (2002) TCP behavior across multihop wireless networks and the wired internet. *WoWMoM '02: Proceedings of the 5th ACM international workshop on Wireless mobile multimedia*, New York, NY, USA, pp. 41–48. ACM Press.

[41] Yang, L., Seah, W. K., and Yin, Q. (2003) Improving fairness among tcp flows crossing wireless ad hoc and wired networks. *MobiHoc '03: Proceedings of the*

4th ACM international symposium on Mobile ad hoc networking & computing, New York, NY, USA, pp. 57–63. ACM Press.

[42] Jiang, L. B. and Liew, S. C. (2005) Proportional fairness in wireless LANs and ad hoc networks. *Wireless Communications and Networking Conference, 2005 IEEE*, Long Beach, CA, USA, 13-17 March, pp. 1551–1556. IEEE Computer Society Press.

[43] Sundaresan, K., Hsieh, V. A. H., and Sivakumar, R. (2005) ATP: A Reliable Transport Protocol for Ad Hoc Networks. *IEEE transactions on mobile computing*, **4**, 588–603.

[44] Anastasi, G. and Passarella, A. (2003) Towards a Novel Transport Protocol for Ad Hoc Networks. *Proc. IFIP Int. Conference on Personal Wireless Communications (PWC 2003)*, Venice, Italy, September 23–25, pp. 805–810. LNCS.

[45] Anastasi, G., Conti, M., Ancillotti, E., and Passarella, A. (2005) TPA: A Transport Protocol for Ad Hoc Networks. *ISCC '05: Proceedings of the 10th IEEE Symposium on Computers and Communications (ISCC'05)*, Murcia, Cartagena,Spain, June 27–30, pp. 51–56. IEEE Computer Society.

[46] Anastasi, G., Ancillotti, E., Conti, M., and Passarella, A. (2006) Experimental analysis of a transport protocol for ad hoc networks (TPA). *PE-WASUN '06: Proceedings of the 3rd ACM international workshop on Performance evaluation of wireless ad hoc, sensor and ubiquitous networks*, New York, NY, USA, september, pp. 9–16. ACM Press.

[47] Stevens, W. (1994) TCP/IP Illustrated. Addison Wesley.

[48] Park, V. D. and Corson, M. S. (1997) A highly adaptive distributed routing algorithm for mobile wireless networks. *INFOCOM '97: Proceedings of the INFOCOM '97. Sixteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Driving the Information Revolution*, Kobe (Japan), April 09–11 1405. IEEE Computer Society.

[49] Sun, D. and Man, H. (2001) ENIC - An Improved Reliable Transport Scheme for Mobile Ad Hoc Networks. *IEEE Globecom Conference*, San Antonio, TX, November, pp. 2852–2856. IEEE Computer Society Press.

[50] Perkins, C., Belding-Royer, E., and Das, S. (2003). Ad hoc On-Demand Distance Vector (AODV) Routing. RFC 3561.

[51] AODV-UU, AODV Linux Implementation, University of Uppsala.

[52] Clausen, T. and Jaquet, P. (2003). Optimized Link State Routing Protocol (OLSR). RFC 3626.

[53] Tønnesen, A. (2004). Implementation of the OLSR specification (OLSR_UniK). Version 0.4.8.

[54] Ancillotti, E., Bruno, R., Conti, M., Gregori, E., and Pinizzotto, A. (2006) A Layer-2 Architecture for Interconnecting Multi-hop Hybrid Ad Hoc Networks to the Internet. *in Proceedings of WONS 2006*, Les Menuires, France, January, 18–20, pp. 87–96.

[55] Ancillotti, E., Anastasi, G., Conti, M., and Passarella, A. *Experimental Analysis of TCP Performance in Static Multi-hop Ad Hoc Networks*. M. Conti, J. Crowcroft, and A. Passarella. Chapter 6 in Mobile Ad Hoc Networks: from Theory to Reality, Nova Science Publisher.

[56] Xu, K., Gerla, M., and Bae, S. (2003) Effectiveness of RTS/CTS Handshake in IEEE 802.11 Based Ad Hoc Networks. *Ad Hoc Networks Journal*, **1**, 107–123.

[57] The Network Simulator - ns-2 (version 2.30).

[58] Papanastasiou, S., Mackenzie, L. M., Ould-Khaoua, M., and Charissis, V. (2006) On the interaction of TCP and Routing Protocols in MANETs. *AICT-ICIW '06: Proceedings of the Advanced Int'l Conference on Telecommunications and Int'l Conference on Internet and Web Applications and Services*, uadeloupe, French Caribbean, February 19–22, pp. 62–69. IEEE Computer Society.

[59] Borgia, E. and Delmastro, F. (2007) Effects of unstable links on aodv performance in real testbeds. *EURASIP Journal on Wireless Communications and Networking*, **2007**, Article ID 19375, 14 pages. doi:10.1155/2007/19375.

[60] Jain, R., Chiu, D., and Hawe, W. (1984). A Quantitative Measure of Fairness and Discrimination for Resource Allocation in Shared Systems. DEC Technical Report DEC-TR-301.

[61] S. PalChaudhuri, R. U. ns-2 Code for Random Trip Mobility Model.

[62] PalChaudhuri, S., Boudec, J.-Y. L., and Vojnovic, M. (2005) Perfect simulations for random trip mobility models. *ANSS '05: Proceedings of the 38th annual Symposium on Simulation*, Washington, DC, USA, April 04–06, pp. 72–79. IEEE Computer Society.

[63] Ancillotti, E., Anastasi, G., Conti, M., and Passarella, A. *Design, Implementation and Measurements of a Transport Protocol for Ad Hoc Networks*. M. Conti. Chapter in MobileMAN. Springer. To appear.

[64] Thekkath, C., Nguyen, T., Moy, E., and Lazowska, E. (1993) Implementing network protocols at user level. *IEEE/ACM Transactions on Networking*, **1**, 554–565.

[65] Stevens, W. (1999) *UNIX Network Programming Volume 2, Interprocess Communications*, 2nd edition edition. Prentice Hall PTR.

APPENDIX A: TPA IMPLEMENTATION

In this appendix we provide a brief description of the TPA implementation in our testbed. A more detailed description can be found in [63]. Network protocols are usually implemented in the kernel space and can be accessed by applications through an interface consisting of a set of system calls (e.g., the socket-based interface). Security and performance are the main motivations behind this approach. However, there are several factors that can motivate an implementation of network protocols out of the kernel space [64]. The most obvious of these factors is ease of prototyping, debugging and maintenance. Another factor may be an *improved system stability*. When developing protocols in a user-level environment, an unstable stack affects only the application using it and does not cause a system crash.

Therefore, we decided to prototype TPA by a user-level implementation. Since TPA only requires a datagram service it was implemented on top of the

TABLE 15. Functions provided by the TPA library.

Function name	Meaning
`tpa_socket()`	Creates a TPA socket
`tpa_connect()`	Connects the socket to the specified address
`tpa_bind()`	Gives to the socket the local address
`tpa_listen()`	Specifies a willingness to accept incoming connections and a queue limit for incoming connections
`tpa_accept()`	Listen for TPA connections requests
`tpa_close()`	Closes the TPA connection
`tpa_send()`	Sends data over a TPA connection
`tpa_recv()`	Receives data over TPA connection

UDP/IP protocols that are accessed through the socket-based mechanism. Therefore, the implemented TPA header does not include the ports and the checksum fields (see Section 3.1) that are already included in the UDP header. As the (implemented) TPA header is 12 bytes long, the overheads of TCP and TPA-over-UDP are exactly the same.

To allow a simple re-use of legacy application written for TCP, we implemented for TPA a socket application programming interface (API) similar to that provided by TCP. TCP. Table 15 shows the list of functions provided by the user-level library implementing the TPA protocol.

TPA was implemented with distinct execution flows that interact according to the client/server and producer/consumer models. Specifically, we structured the TPA software module by means of three processes: a *data-processing* process, a *sender* process, and a *receiver* process (see Figure 22). The *data-processing* process collects data passed by the application process in a buffer to form blocks. Data blocks are then passed to the *sender* process that manages their transmission according to the TPA specification. Finally the *receiver* process is in charge of processing data coming from the network and sending ACKs back to the sender. In this model processes resident on the same machine communicate with each other by using FIFOs [65] and the signal mechanism, while the sender and the receiver process use a UDP socket to transmit data and ACK segments.

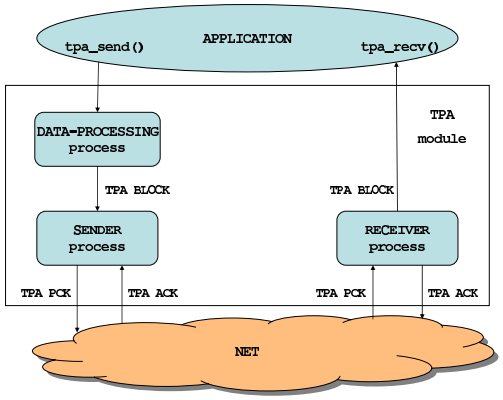

FIGURE 22. Inter-Process Communications.

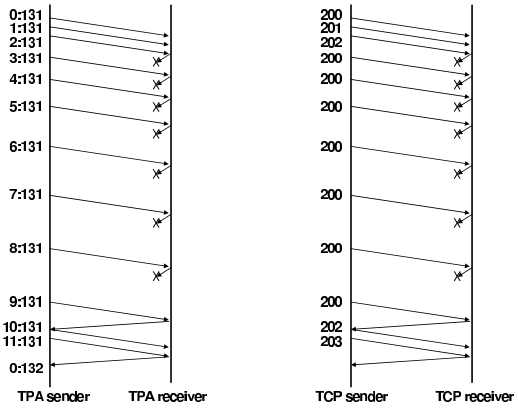

FIGURE 23. Impact of an ACK inhibition.

APPENDIX B: EXAMPLE OF A TRACE ANALYSIS

The aim of this appendix is to show the different behaviour of TPA and TCP in the presence of an ACK *inhibition*, i.e., when the route between receiver and sender is broken, while the route between sender and receiver is still available. This gives a tangible example of why TPA mechanisms results in higher performance with respect to TCP. To this end we refer to a portion of the TPA trace file obtained in one replica of the 3-hop OLSR experiment.

Figure 23-left shows the behaviour of TPA after about 39.557s from the start of the experiment. Numbers on the left-hand side should be read as $< position\_in\_the\_block : block\_number >$. At this time the TPA sender is transmitting segments belonging to the block 131. The route between node N4 (TPA receiver) and node N1 (TPA sender) is broken while the route between node N1 and N4 is active. This implies that TPA data segments can reach node N4 while ACKs are dropped by the routing protocol and never reach node N1. Upon timers expiration for segment 0, TPA

enters the *congested* state and starts sending segments with a $cwnd_{max}$ parameter set to one (segments 3 to 9 are sent at each timer expiration). Those segments are successfully received by the destination that continues to send ACKs to the sender node. However, ACKs are discarded by the routing protocol. At time 50.92s, the routing protocol recovers the route to node N1. At this point the TPA receiver, on reception of segments 9, sends back an ACK that reaches the TPA sender and notifies it that all segments belonging to block 131 have been successfully received by the destination. Upon reception of the above ACK, TPA leaves the *congested* state and sends segments 10 and 11. Then, upon reception of the ACK for segment 11, TPA transmits segments belonging to a new block.

Figure 23-right shows the behaviour that TCP would have had in the same conditions. In this case, numbers on the left-hand side should be read as $< segment\_sequence\_number >$. For the sake of simplicity we will refer to TCP sequence number in terms of segments instead of bytes. As we can see from the sequence shown in Figure 23-right, upon timer expiration for segment 200 the TCP sender retransmits the same segment and continues to do so upon recursive timeouts.

In such cases, in which data segment reach the destination, but ACK segments do not reach the sender, TPA is not stuck at retransmitting the same segment, as legacy TCP is. Therefore, the destination continues to receive new data segments. In other words, during ACK inhibitions, TPA is able to exploit the unidirectional route available between the sender and the destination, while legacy TCP is not. This results in a performance improvement of TPA, compared with TCP behaviour.