Experimental Analysis of a Transport Protocol for Ad hoc Networks (TPA)

Giuseppe Anastasi, Emilio Ancillotti
Dept. of Information Engineering,
University of Pisa, Italy
{firstname.lastname@iet.unipi.it}

Marco Conti, Andrea Passarella
CNR-IIT
National Research Council, Italy
{firstname.lastname@iit.cnr.it}

ABSTRACT
Many previous papers have pointed out that TCP performance in multi-hop ad hoc networks is not optimal. This is due to several TCP design principles that reflect the characteristics of wired networks dominant at the time when TCP was designed, but are not adequate for multi-hop ad hoc networks. For example, congestion phenomena in multi-hop networks are very different than in traditional wired networks, and route failures and route changes may be frequent events. To overcome these problems, in a previous work we presented a novel transport protocol – named TPA – specifically tailored to multi-hop ad hoc networks. In this paper we perform an experimental analysis of TPA in static multi-hop scenarios. Specifically, we compare TPA and TCP performance in a chain topology with different number of hops and traffic patterns. We also consider the effect of the routing protocol. Our experimental results show that TPA protocol outperforms TCP significantly both in terms of throughput and energy consumption.

Categories and Subject Descriptors
C.2.2 [**Computer-Communication Networks**]: Network Protocols; C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design.

General Terms
Measurement, Performance, Design, Algorithms.

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

1. INTRODUCTION
TCP (Transmission Control Protocol) is the *de facto* standard for reliable connection-oriented transport protocols, and is normally used over the Internet Protocol (IP) to provide reliable communication to Internet applications. However, in the last years many papers have pointed out that TCP performance in multi-hop ad hoc networks is not optimal. This is because some assumptions in its design are clearly inspired from the characteristics of wired networks dominant at the time when it was conceived. More specifically, TCP implicitly assumes that packet 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 multi-hop ad hoc networks.

In multi-hop ad hoc networks, packet losses due to interference and link-layer contentions are largely predominant, while packet losses due to buffer overflows at intermediate nodes are rare events. The TCP protocol reacts to packet losses originated by link-layer contentions by activating the window-based congestion-control mechanism. This may lead to throughput degradation and instability [1], [11], [8], [12]. In addiction, in multi-hop ad hoc networks nodes may be mobile. This may further degrade the TCP performance [15]. However, in this paper we not consider node mobility.

To overcome the above problems several proposals have been presented. Most of them are TCP enhancements aimed at improving TCP performance either by means of lower layer mechanisms that hide the multi-hop network characteristics from TCP, or by modifying TCP mechanisms. A comprehensive survey of such TCP variants can be found in [15]. In addition, some novel transport protocols, designed from scratch and specifically tailored to multi-hop ad hoc networks, have been also proposed. In a previous paper [3] we argued that the characteristics of multi-hop ad hoc networks are very different from those of wired networks for which TCP was originally conceived. Thus, a TCP variant may not be the best choice for multi-hop ad hoc networks. Therefore, we designed a new transport protocol named TPA (Transport protocol for Ad Hoc networks). TPA provides a reliable, connection-oriented type of service and includes mechanisms for managing route failures and route changes efficiently. In addition, it has a novel congestion control mechanism that takes into account problems introduced by link-layer contentions. Finally, it implements a novel retransmission policy to reduce the number of useless retransmissions and, hence, power consumption.

The authors in [17] also took an approach similar to ours and defined the Ad hoc Transport Protocol (ATP). Even if novel in many respects, TPA conserves some TCP characteristics adapting them to the new environment (for example, it still uses a window-

PE-WASUN'06, October 6, 2006, Torremolinos, Malaga, Spain.

based transmission scheme but with very small maximum congestion window size). On the other side, ATP is completely antithetic to TCP as it relies upon rate-based transmissions, network-supported congestion detection and control, no retransmission timeout, decoupled congestion control and reliability, etc. [17]. Specifically, it requires assistance from the underlying network layer (and more precisely from intermediate nodes along the connection) for performing its tasks. This may make it unsuitable for those environments where the network layer protocol does not provide such a support.

In [4] we performed a preliminary simulation analysis of TPA and compared its performance to that of TCP. The results obtained were very promising as they showed that TPA outperforms TCP in all scenarios taken into consideration, both in terms of throughput and energy consumption. However, previous experimental studies have shown that certain aspects of real multi-hop networks are often not effectively captured in simulation tools [5]. Furthermore, available software and hardware products often use parameter settings different from those commonly assumed in simulation tools. Finally, real operating conditions are often different from those modeled in simulation experiments. For example, interferences caused by WiFi hotspots or other devices in the proximity are inevitable in practice. For all the above reasons, we believe it is of great importance to compare TPA and TCP performance in a real environment. Therefore, we implemented the TPA protocol and performed the experimental analysis described in this paper. The TPA performance is thus compared with that of TCP NewReno (with the delayed-ACK mechanism enabled) available with Linux distributions. To perform our experiments, we used a testbed based on laptops running the Linux operating system. In addition, to investigate the effect of the routing protocol, we considered two very popular routing protocols, i.e., AODV [16], [2] and OLSR [1], [10] which take a different approach on building and maintaining routes (reactive vs. proactive). For the sake of simplicity our analysis is limited to static networks with a chain topology. The results obtained show that TPA outperforms TCP, both terms of throughput and energy consumption, especially when problems related to link-layer contentions became more evident. Specifically, with respect to the best TCP configuration, TPA increases the throughput up to 26%, and reduces the number of (re)transmissions up to 80%.

The rest of the paper is organized as follows. Section 2 describes the TPA protocol. Section 3 provides a brief description of TPA implementation in a Linux environment. Section 4 introduces the experimental environment used for performance evaluation. Section 5 discusses the results obtained from the experiments. Finally, Section 6 concludes the paper.

2. TPA PROTOCOL DESCRIPTION

TPA provides a reliable, connection-oriented type of service. In the following subsections we will provide a description of the main aspects of the protocol. The interested reader can refer to [3] and [4] for motivations behind TPA design, and protocol details, respectively.

2.1 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 [18], while the congestion control mechanism is described in Section 2.2.

TPA manages application data in blocks, with a block consisting of K segments. 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.

Segment transmissions are handled as follows. Whenever sending a segment, the source TPA sets a timer and waits for the related ACK 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. 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.

As soon as all segments belonging to the block have been transmitted, the sender performs a second round for retransmitting "timed-out" segments, which are said to form a "retransmission stream". 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. Only when all segments belonging to a block have been acknowledged, TPA takes care to manage the next block.

The proposed scheme has several advantages with respect to the transmission 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 out-of-order arrivals of segments.

2.2 Congestion Control Mechanism

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 neighboring nodes and, thus, it sends an ELFN (if available) back to the sender node. 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.

The TPA congestion control mechanism is window-based as in TCP. However, in TPA the maximum congestion window size ($cwnd_{max}$) is very small (in the order of 2-3 TPA segments) and, hence, the maximum and minimum values are very close. Therefore, the TPA congestion control algorithm is 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.

2.3 ACK Management

Many papers [1], [14], [1] have shown that the *Delayed ACK* mechanism can improve significantly the TCP performance. Based on these results we implemented a *Delayed ACK* mechanism in TPA as well. When using this mechanism, if the TPA sender is transmitting with the maximum value for the transmission window ($cwnd_{max}$), the TPA receiver sends back one acknowledgement *every other* segment received, or upon timer expiration. Otherwise, the TPA receiver sends back one acknowledgment for *each* segment received. The TPA sender uses the *txStatus* flag of the TPA segment header to announce the size of the transmission window to the receiver.

We also implemented the following modified version of *Delayed ACK*. To minimize the number of ACKs in transit in the network, the receiver can delay the ACK transmission up to $cwnd_{max}$ segments. In other words, when the TPA sender transmits segments with a window size equal to one, the receiver sends back one ACK for *each* segment received. Otherwise, if the TPA sender transmits segments with a window size equal to $cwnd_{max}$ segments the TPA receiver sends back one ACK *every* $cwnd_{max}$ segments received (e.g. every three segments received if $cwnd_{max}$ is set to three) or on timer expiration. Throughout the paper we will refer to TPA with modified version of the *Delayed ACK* as TPA^{*}.

In both TPA variants, the interval that triggers the ACK transmission is set to a constant value.

3. TPA PROTOCOL IMPLEMENTATION

In this section, we provide a brief description of our TPA implementation in the Linux environment. A more detailed description can be found in [6]. 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 [19]. 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 means of a user-level implementation. Since TPA only requires a datagram service it was implemented on top of the UDP/IP protocols that are accessed through the legacy socket-based interface. 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.

TPA was implemented (by using the C programming language) with distinct execution flows that interact according to the client/server and producer/consumer paradigms. Specifically, we structured the TPA software module by means of three processes: a *data-processing* process, a *sender* process, and a *receiver* process. 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 them according to TPA specifications. 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 each other by using FIFOs [1] and signals, while the *sender* and the *receiver* process use a UDP socket to transmit data and ACK packets.

4. EXPERIMENTAL ENVIRONMENT

4.1 Testbed Description

Our testbed consisted of IBM R-50 laptops equipped with integrated Intel Pro-Wireless 2200 wireless cards. All the 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 bit rate set to 2 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. The transmission power of the wireless cards was set to the minimum allowed value (-12db) so as to reduce the transmission range and make it possible to perform experiments in an indoor environment.

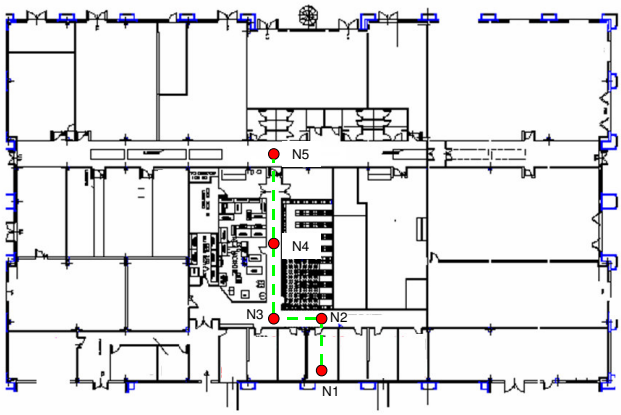

Figure 1. Indoor environment and network topology used in our experiments.

Figure 1 shows the indoor environment where the experiments were carried out. It is a real working environment with offices and labs. In particular, there are several WiFi Access Points in the proximity. Although this environment may influences our performance measures, we believe that it is important to test the TPA protocol performance in a real working environment. In our experiments we considered a chain topology with five nodes deployed as in Figure 1. In all the experiments node N1 was the sender, while the receiver (and the number of active nodes) depended on the specific scenario. Specifically, we considered four different scenarios with hop count ranging from 1 to 4. For

example, in the 3-hop scenario, node N4 is the receiver (node N5 is not active). The distance between nodes was chosen in such a way that only adjacent nodes were within the transmission range of each other.

We used *ftp-like* traffic, i.e., the sender node had always data ready to send. To this end, we developed a simple client/server application using Linux sockets operating with TCP protocol and we adapted this application to operate with TPA sockets as well. Several papers have shown the advantage of limiting the TCP window size in multi-hop ad hoc networks. To be fair, we compared TPA and TCP with limited window size. To limit the maximum value for the TCP *congestion window size* (*cwnd*), we used the *TCP_WINDOW_CLAMP* socket parameter that permits to bound the size of the TCP *advertised window*. The segment size was constant in all the experiments with the transport-layer payload size set to 1460 bytes. To capture TCP segment we used *tcpdump*, while to analyze the experiments results we used *tcpstat* and *tcptrace* (enhanced by our shell scripts). Since TPA is implemented in the user-space to capture TPA segments we used our TPA code.

As anticipated, we compared the performance of TPA and TCP by considering two different routing protocols, i.e., AODV and OLSR. AODV (Ad hoc On-demand Distance Vector, [16]) is a well-known reactive protocol that uses RREQ, RREP and RERR messages to discover and maintain routes to the destination. It can use two different mechanisms for neighbor discovery and local connectivity maintenance, i.e., link layer information provided by the underlying MAC protocol, or HELLO messages periodically exchanged between all nodes in the multi-hop network. In our testbed we used the AODV implementation for Linux by the Uppsala University [2] 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, [10]) is an optimization of the classical link state algorithm for mobile ad hoc networks (it is thus a proactive protocol). OLSR periodically floods the network with route information so that each node can build locally a routing table containing the complete information of routes to all possible destinations within the ad hoc network. Similarly to AODV, OLSR employs a neighbor discovery procedure based on HELLO messages. In our testbed we used the OLSR_UniK implementation for Linux version 0.4.10 [1]. We set all the parameters to their default values and we disabled the OLSR hysteresis mechanism, because it was found to degrade TCP throughput in an unacceptable way [8].

Table 1 shows the operational parameters for the TPA protocol. As far as TCP, the following kernel variables were disabled.

- `tcp_moderate_rcvbuf`. This variable enables the kernel to dynamically update receiver socket buffer size.
- `tcp_bic`. This variable enables the Binary Increase (BIC) congestion control algorithm.
- `tcp_vegas_cong_avoid`. This variable enables TCP-Vegas congestion avoidance algorithm.
- `tcp_westwood`. This variable enables TCP westwood (TCPW) congestion control algorithm.
- `tcp_sack`. This variables enables RFC 2018 TCP Selective Acknowledgments (SACK) mechanism.
- `tcp_window_scaling`. This variable enables RFC 1323 TCP window scaling mechanism.
- `tcp_timestamps`. This variable enables RFC 1323 TCP timestamps mechanism.

In our experimental analysis we compared the performance of TCP and TPA using different values for the maximum congestion window size. Previous simulation studies [8], [11] suggest that in our scenarios (chain topology whit hop count ranging from 3 to 4) the optimal value for the maximum *cwnd* size is 2. However, experimental measurements from a real testbed [7] show that 3 is the optimal value for the maximum *cwnd* size in a real environment. Therefore, we considered a maximum *cwnd* of size 2 and 3 for both TCP and TPA. We also considered an *unclamped* (uc) value for the maximum *cwnd* size of TCP. Moreover, we evaluated the performance of TPA with the modified version of the delayed ACK mechanism. In the following we will refer to TCP with maximum *cwnd* size of W as *TCP-W,* and we will refer to TPA with $cwnd_{max}$ set to W as *TPA-W* (or *TPA*-W*).

Table 1. TPA Operational Parameters.

Parameter	Value
th_{RC} (TPA)	3 segments
n_{RC} (TPA)	3 segments
G	0.125
H	0.25
g1	0.25
h1	0.5
Th_{CONG} (TPA)	1 segment
th_{ACK} (TPA)	1 segment
Block Size (TPA)	12 segments

4.2 Performance measures

In our analysis we considered 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) was obtained as:

$$rtx = \frac{\#\ of\ \ packets\ \ retransmitted\ by\ the\ source}{\#\ of\ non-duplicated\ packets\ \ successfully\ received\ by\ the\ destination}$$

The re-transmission index allows us to evaluate the ability of TPA/TCP to handle transmission 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-)transmission efficiently.

Therefore, a small value for the re-transmission index is highly desirable.

4.3 Methodology

When dealing with real testbeds one of the main difficulties is that experiments cannot be repeated exactly in the same way since external conditions may vary from time to time -- sometimes during the same experiment -- and there is definitely no control on them. Therefore, successive experiments carried out under the same operating conditions may provide outcomes that differ significantly from each other. In addition, comparison of performance measurements obtained in different scenarios or operating conditions becomes hard or even impossible.

To achieve more statistical accuracy, we replicated each experiment 5 times, and averaged the performance measures over the entire set of 5 replicas. In addition, experiments with similar parameter values (e.g., with different *maximum cwnd* size but all other equal parameter values) were performed in an interleaved way. For instance, with reference to the 3-hop topology, we ran a first replica of *TCP-2*, *TCP-3, TCP-uc*, *TPA-2*, *TPA-3* and *TPA*-3*. Then, we performed a second replica, a third replica, and so on.

Each replica was 120s long, and consisted of a file transfer. To perform multiple replicas the whole process of experimentation (data generation, logging and archiving) was automated using shell scripts.

5. EXPERIMENTAL RESULTS

In this section we describe the results obtained from our experiments carried out in different scenarios. Specifically, we considered four chain topologies with increasing number of hops (from 1 to 4). In addition, for each scenario we performed the experiments using both AODV and OLSR routing protocol. In all the experiments only a single TPA/TCP connection was active in the network. The purpose was to investigate the influence of the link-layer contention on TPA and TCP in a very simple network topology. Then, we extended our analysis by considering the effects of interfering traffic as well.

This section is organized as follows. Section 5.1 and 5.2 show the results of the experiments with AODV and OLSR, respectively. Section 5.3 describes the different behavior of TPA and TCP in the presence of *ACK inhibition* by means of a detailed analysis of TPA traces.

5.1 Analysis with AODV routing protocol

In this section we describe the experimental results obtained with the AODV routing protocol. Figure 2 through Figure 6 show the throughput and retransmission index of both TCP and TPA in all the scenarios we considered.

Figure 2 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 IEEE 802.11 MAC protocol. Specifically, in this simple scenario all nodes are within the transmission range of each other and can thus coordinate efficiently their transmissions. This produces a retransmission index equal to zero for all values of the maximum *cwnd* size parameter. Figure 2 also shows that TCP with an unclamped congestion window slightly outperform both TPA and TCP with clamped window. Specifically, *TCP-uc* throughput is about 2% higher that *TPA-3* throughput. However, we can observe that *TPA*-3* achieves about the same throughput of *TCP-uc*. It may be worthwhile to recall here that TPA is implemented in the user space and, thus, experiences a higher overhead with respect to *TCP-uc* running in the kernel space.

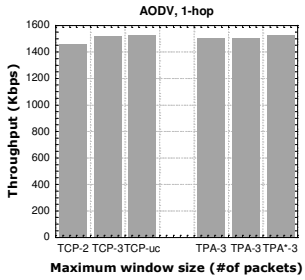

Figure 2. Throughput (left) and percentage of retransmitted segments (right) vs. window size in the 1-hop scenario.

Figure 3 shows that in the 2-hop scenario TPA outperforms TCP with clamped window both in terms of throughput and retransmission index. Specifically, *TPA-2* provides approximately the same throughput as *TCP-2* but retransmits about 79% less segments. *TPA-3* provides a throughput about 9% higher than that of *TCP-3* but it reduces the number of retransmitted segments of about 13%.

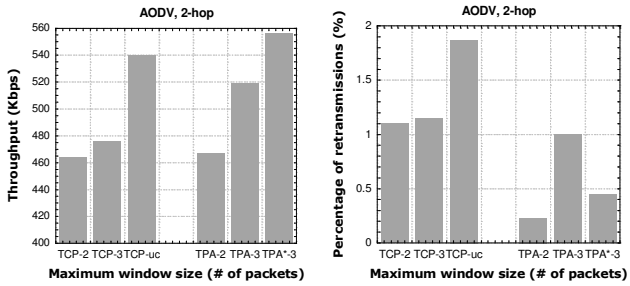

Figure 3.Throughput (left) and percentage of retransmitted segments (right) vs. window size in the 2-hop scenario.

The main reason for the difference between TCP and TPA performance in this scenario, where there is no interference (all nodes are in the carrier sensing of each other), resides on HELLO messages used by AODV to maintain local connectivity [7]. When using HELLO messages (with default parameter values), it's sufficient to loose two consecutive HELLO messages to detect a link failure. Since HELLO messages are broadcast packets, and broadcast packets are neither acknowledged nor retransmitted by the MAC layer, they are vulnerable to collisions and channel errors. This results in a loss of local connectivity even in networks where all nodes are within the same carrier sensing range. This loss of local connectivity results in frequent route failures which turns out in timeouts and retransmissions at the sender side. However, as expected TPA is much more efficient than TCP in managing such events as highlighted by the above results (Figure 3 right).

Figure 3 also shows that, as in the 2-hop scenario, TCP achieves optimal throughput when it uses an unclamped transmission

window [1]. Specifically, *TCP-uc* throughput is about 13% higher than *TCP-3* throughput. This happen because in a scenario where all nodes are within the same carries sensing range, the IEEE 802.11 MAC protocol works well. However, *TPA*-3* provides a higher throughput than *TCP-uc* (about 3%) while retransmits about 70% less segments. This is because *TPA*-3* reduces the number of ACKs and, hence, the amount traffic on the channel.

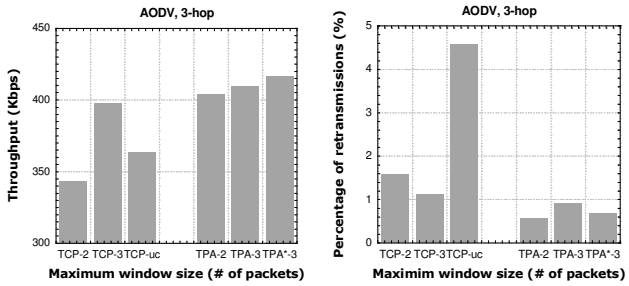

Figure 4. Throughput (left) and percentage of retransmitted segments (right) vs. window size in the 3-hop scenario.

The 3-hop scenario is the first scenario where problems related to link layer contentions become evident [1]. Figure 4 shows that the optimal value for the TCP maximum *cwnd* size is 3 segments. This result is apparently in contrast with previous simulation results showing that in the 3-hop scenario the TCP optimal window size is 2 segments. This discrepancy has been found to reside in the different behaviour between the TCP version implemented in the Linux kernel used in our testbed, and the TCP implementation available in the ns-2 simulator [13]. In [7], by a detailed analysis of traces, we found that when a maximum *cwnd* size of to 2 segments is used, 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. Figure 4 shows that TPA outperforms TCP with optimal window size both in terms of throughput and retransmission index (with all congestion window size). Specifically, *TPA-2* increases the throughput of about 17% with respect to *TCP-2*, retransmitting about 62% less segments. *TPA-3* provides a throughput 3% higher than *TCP-3*, while retransmitting about 19% less segments. Figure 4 also shows that *TPA*-3* is the best configuration for TPA, providing a throughput increase of about 5% with respect to *TCP-3* and retransmitting about 38% less segments.

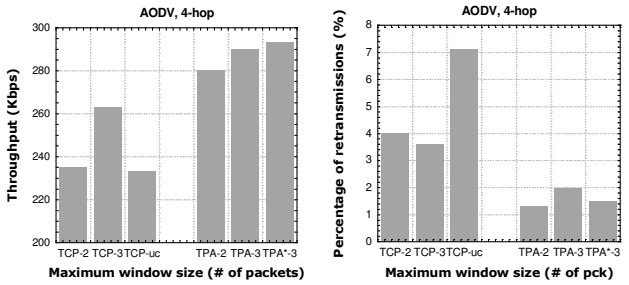

Figure 5. Throughput (left) and percentage of retransmitted packets (right) vs. window size in the 4-hop scenario.

In the 4-hop scenario, the link layer contentions increases and, thus the difference in performance between the two protocols is expected to become even more evident. Figure 5 shows that, as in the 3-hop scenario, the optimal value for the TCP maximum *cwnd* size is 3 segments. Throughput with *TPA-2* is 19% higher than with *TCP-2*, while retransmissions are cut by 68%. *TPA-3* increases the throughput of about 10% with respect to *TCP-3*, and retransmits 45% less segments. Also in this scenario *TPA*-3* is the best configuration for TPA, providing an increase of about 11% in terms of throughput with respect to *TCP-3* and a decrease of about 60% in terms of retransmitted packets.

Finally, we also evaluated 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 to it. This CBR session has N3 as its source node, and N2 as its recipient node (Figure 1), and uses UDP as the transport protocol. It inject a periodic traffic pattern in the network with a bit rate equal to 192 Kbps, which corresponds to the bit rate of an MP3 stream. The packet size of UDP traffic was set to 1460 bytes. The results obtained, summarized in Figure 6, shows that in this scenario there is no 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. With respect to *TCP-2*, *TPA-2* exhibits an increment in throughput of about 24% and a decrement of retransmission index of about 70%. *TPA-3* provides a throughput of about 8% higher than TCP while retransmitting about 52% less segments. We can observe that also in this scenario *TPA*-3* is the best configuration for TPA, providing an increment in throughput of about 17% with respect to *TCP-3* and retransmitting about 60% less segments.

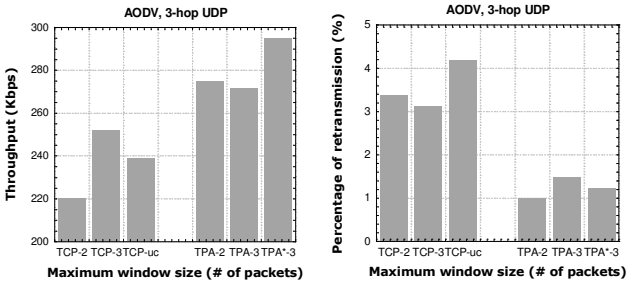

Figure 6. Throughput (left) and percentage of retransmitted segments (right) vs. window size in the 3-hop-UDP scenario.

5.2 Analysis with OLSR routing protocol

To conclude our analysis, in this section we show the experimental results obtained in the same above scenarios using OLSR instead of AODV as routing protocol. Table 2 and Table 3 summarize the throughput and retransmission index, respectively. We can see that the results obtained with OSLR are similar 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. We can observe that the performance improvement of TPA with respect to TCP is much more evident when OLSR is used instead of AODV as routing protocol. For example, in the 4-hop scenario TPA-2 increases the throughput of about 19% with respect to TCP-2 while retransmitting about 74% less segments. Throughput with TPA-3 is 17% higher than with TCP-3, and retransmissions are 56% less. Finally, TPA*-3 provides a throughput about 25% higher that TCP-3 (that is the optimal configuration for TCP) and retransmits about 60% less segments. Table 2 and Table 3 also

show that in the 2-hop scenario, in contrast with the results obtained with AODV, there are not significant differences between TPA and TCP performance. One possible reason for this behavior is the different parameter values used by AODV and OLSR to manage HELLO messages. Specifically, by considering the defaults parameter values, OLSR considers a link as broken if it fails to receive *three* consecutive HELLO messages form its neighbor while AODV assumes a link failure when if fails to receive *two* consecutive HELLO messages. This make OLSR more robust to false link failures [7], and results in a retransmission index closer to zero in the 2-hop scenario.

Table 2. Throughput (in Kbps) vs. maximum *cwnd* size with OLSR.

Scenario	TCP-2	TPA-2	TCP-3	TPA-3	TPA*-3	TCP-uc
1-hop	1369	1464	1456	1461	1495	1524
2-hop	603	679	657	650	687	675
3-hop	283	356	302	354	339	308
4-hop	175	208	173	203	215	163
3-hop-UDP	273	343	279	300	314	288

Table 3. Retransmission index vs. maximum *cwnd* size with OLSR.

Scenario	TCP-2	TPA-2	TCP-3	TPA-3	TPA*-3	TCP-uc
1-hop	0	0	0	0	0	0
2-hop	0,02	0	0,01	0,07	0	0
3-hop	1,13	0,37	1,5	0,67	0,56	1,22
4-hop	3,55	0,93	3,2	1,41	1,3	4,8
3-hop-UDP	1,49	0,34	1,42	0,97	0,35	1,62

5.3 Trace Analysis

The aim of this section is to emphasize 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 can help us to understand why TPA can outperform TCP. To this end we refer to a portion of the TPA trace file obtained in one replica of the OLSR experiment in the 3-hop scenario.

Figure 7-left shows the behaviour of TPA after about 39,557s since the beginning of the experiment. 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 reverse 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 receptions 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.

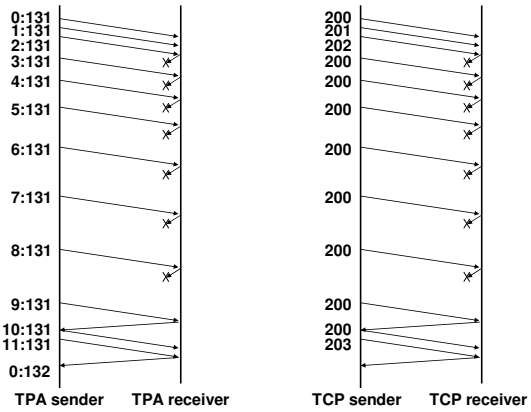

Figure 7. Impact of an *ACK inhibition*.

Figure 7-right shows the behaviour that TCP would have had in the same conditions. For the sake of simplicity we will refer to TCP sequence numbers in terms of packets instead of bytes. As we can see from the sequence shown in Figure 7-right, upon timer expiration for segment 200 the TCP sender retransmits the same segment and continues to do so upon recursive timeouts. This results in a performance degradation compared with TPA behaviour.

6. SUMMARY AND CONCLUSIONS

In this paper we have reported the results of an experimental analysis of the TPA protocol, a novel transport protocol specifically tailored to multi-hop ad hoc networks. We have implemented TPA in a real testbed and compared TPA and TCP performance in a chain topology with different number of hops and traffic patterns. From the results obtained we can draw the following conclusions. In the 1-hop and 2-hop scenarios, since all nodes are in the carrier sensing of each other, the major issues of running TCP over multi-hop ad hoc networks disappear. TCP with unclamped *cwnd* size exhibits a higher throughput than TCP with clamped window, even though the differences in terms of throughput are very small. Nevertheless, in the 2-hop scenario *TPA*-3* slightly outperforms TCP with unclamped transmission window, both in terms of throughput and retransmission index. This is a side effect of transmitting fewer ACK segments. As an example, when using AODV, TPA increases the throughput of about 3% and reduces the retransmission index of about 72%. In the 3-hop and 4-hop scenarios, the limitations of standard TCP over multi-hop networks become ever more evident. In such harsher configuration, TPA always outperforms TCP, either with AODV or with OLSR. For example, in the 4-hop scenario with AODV, *TPA-3* increases the throughput of about 10%, and reduces the retransmission index of about 45% in comparison with *TCP-3*. With OLSR, *TPA-3* increases the throughput of about 17% and reduces the retransmission index of about 56% with respect to *TCP-3*. Finally, when using *TPA*-3*, TPA further improves its performance. In the 4-hop scenario it increases the

throughput up to 26%, and reduces the retransmission index up to 80% with respect to *TCP-3*.

In conclusion, the results of the experimental analysis have shown that TPA always outperforms TCP, and highlights that the difference in performance increases as soon as the problems related to link-layer congestion becomes evident. Note that the testbed configuration presented in this paper slightly favors TCP in comparison with TPA. We have considered only static topologies, with fairly limited interferences from concurrent transport-layer flows. As soon as concurrent flows starts to appear, the advantage of using TPA becomes higher (see Figure 6). Therefore, we can speculate that in even harsher configurations TPA will further outperform TCP. Specifically, we are currently comparing the two protocols in cases of mobility, more complex topologies, and additional concurrent flows. Subjects for future work also include understanding the interactions between routing protocols' parameters and TCP/TPA performance.

7. ACKNOWLEDGMENTS

This work is partially funded by the Information Society Technologies program of the European Commission under the FET-SAC HAGGLE project.

8. REFERENCES

[1] E. Altman and T. Jimenez, "Novel Delayed ACK Techniques for improving TCP Performance in Multihop Wireless Networks," Proceedings of the *IFIP International Conference on Personal Wireless Communications (PWC 2003)*, Venice, Italy, September 23-25, 2003.

[2] AODV-UU, AODV Linux Implementation, University of Uppsala. Available at: http://core.it.uu.se/AdHoc/AodvUUImpl.

[3] G. Anastasi, A. Passarella, "Towards a Novel Transport Protocol for Ad Hoc Networks", Proc. *IFIP Int. Conference on Personal Wireless Communications (PWC 2003)*, Sept. 23-25, 2003, Venice (Italy), LNCS, N. 2775.

[4] G. Anastasi, E. Ancillotti, M. Conti, A. Passarella, "TPA: A Transport Protocol for Ad hoc Networks", Proceedings of the *IEEE Symposium on Computers and Communications (ISCC 2005)*, Cartagena (Spain), June 27-30, 2005.

[5] G. Anastasi, E. Borgia, M. Conti, E. Gregori, "Wi-Fi in Ad Hoc Mode: A Measurement Study", Proceedings of the *IEEE International Conference on Pervasive Computing and Communications (PerCom 2004)*, Orlando (Florida), March 14-17, 2004.

[6] G. Anastasi, E. Ancillotti, M. Conti, A. Passarella, "Design, Implementation and Measurements of a Transport Protocol for Ad Hoc Networks", chapter in *MobileMAN (M. Conti, Editor)*, Sprinter, to appear. Also available at http://www2.ing.unipi.it/~o1653499/papers.htm.

[7] G. Anastasi, E. Ancillotti, M. Conti, A. Passarella, "Experimental Analysis of TCP Performance in Static Multi-hop Ad Hoc Networks", chapter in *Mobile Ad Hoc Networks: from Theory to Reality, (M. Conti, J. Crowcroft, A. Passarella, Editors)*, Nova Science Publisher, to appear. Also available at http://www2.ing.unipi.it/~o1653499/papers.htm.

[8] E. Ancillotti, R. Bruno, M. Conti, E. Gregori, and A. Pinizzotto, "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 2006, pp. 87-96.

[9] K. Chen, Y. Xue, S. Shah, K. Nahrstedt, "Understanding Bandwidth-Delay Product in Mobile Ad Hoc Networks", *Computer Communications*, Vol. 27, pp. 923-934, 2004.

[10] T. Clausen and P. Jaquet, "Optimized Link State Routing Protocol (OLSR)," *RFC 3626*, October 2003. Available: http://www.ietf.org/rfc/rfc3626.txt.

[11] Z. Fu, P. Zerfos, H. Luo, S. Lu, L. Zhang and M. Gerla, "The Impact of Multihop Wireless Channel on TCP Throughput and Loss", Proceedings of *IEEE INFOCOM 2003*, San Francisco (California), March 30.April 3, 2003.

[12] K. Nahm, A. Helmy, C.-C.Jay Kuo, "TCP over Multihop 802.11 Networks: issues and Performance Enhancement", Proceedings of *ACM MobiHoc 2005*, Urbana-Champaign, IL, pp 277-287, June 2005.

[13] The Network Simulator - ns-2 (version 2.28). http://www.isi.edu/nsnam/ns/index.html.

[14] R. de Oliveira, T. Braun, "A Dynamic Adaptive Acknowledgment Strategy for TCP over Multihop Wireless Networks", Proceedings of *IEEE Infocom 2005*, Vol. 3, pp. 1863-1874, Miami, USA, March 13-17, 2005.

[15] S. Papanastasiou, M. Ould-Khaoua, L. MacKenzie, "TCP Developments in Mobile Ad Hoc Networks", Chapter 30 in *Handbook of Algorithms and Wireless Networking and Mobile Computing (A. Boucherckе editor)*.

[16] C. Perkins, E. Belding-Royer, S. Das, "Ad hoc On-Demand Distance Vector (AODV) Routing", *RFC 3561*, July 2003. Available at: http://www.ietf.org/rfc/rfc3561.txt

[17] K. Sundaresan, V. Anantharaman. H. Hsieh, R. Sivakumar, "ATP: A Reliable Transport Protocol for Ad Hoc Networks", Proceedings of *ACM MobiHoc 2003*, Annapolios (Maryland), June 1-3, 2003

[18] W.R. Stevens, "TCP/IP Illustrated", Vol. 1, Addison Wesley, 1994.

[19] W.R. Stevens, "UNIX Network Programming – Volume 2, Interprocess Communications", Prentice Hall PTR, 2nd Edition, 1999.

[20] C.A. Thekkath, T.D. Nguyen, E. Moy, E.D. Lazowska, "Implementing Network Protocols at User Level", *IEEE/ACM Transactions on Networking*, vol. 1(5), pp. 554-565, October 1993.

[21] A. Tønnesen, "Implementation of the OLSR specification (OLSR UniK)", Version 0.4.10, University of Oslo. Available at: http://www.olsr.org/.

[22] K. Xu, M. Gerla and S. Bae, "Effectiveness of RTS/CTS Handshake in IEEE 802.11 Based Ad Hoc Networks", Ad Hoc Networks Journal, vol. 1, no.1, pp. 107-123, July 2003.

[23] S. Xu, T. Saadawi, "Performance Evaluation of TCP Algorithms in Multi-hop Wireless Packet Networks", Wireless Communications and Mobile Computing, Vol. 2 (2001), N. 1, pp.85-100.