Improving the performability of data transfer in mobile ad hoc networks

Marco Conti, Enrico Gregori, Gaia Maselli
IIT Institute, CNR, Via G. Moruzzi 1, 56124 Pisa, Italy
{*marco.conti, enrico.gregori, gaia.maselli*}*@iit.cnr.it*

***Abstract*— Data transfer in ad hoc environments shows poor network performance due to frequent link breakages and route failures. Selfish and malicious nodes may further deteriorate nodes communication, having a strong impact on transport layer protocols such as TCP, which are highly sensitive to packet losses. Although these misbehaviors have similar effects on the network functioning (i.e. packets are dropped), they are separately addressed by the research community.**

This paper provides a comprehensive method to improve the performance and reliability (performability) of nodes communication in presence of faults, selfish and malicious behavior. Specifically, we propose and evaluate a novel forwarding policy that is based on multi-path routing and considers nodes reliability and routes length in forwarding decisions. We investigate through simulations how this mechanism improves the performability of TCP data transfers. In particular, we show that the simultaneous use of multiple paths yields higher throughput and continuous network connectivity when compared to single path forwarding. This has been verified in case of both fault conditions and intentional nodes misbehavior.

I. Introduction

Mobile ad hoc networking represents a new frontier for wireless communications. Its intrinsic extensibility, auto-configuration, ease of maintenance, and infrastructure-independence capabilities make it a prime candidate for becoming the stalwart technology for the future information society. An essential factor in the deployment of this powerful networking technology is the performability of nodes communication, which may be affected by two important aspects, typical of such environment. On the one hand, the wireless nature of ad hoc nodes exposes the network to temporary fault conditions, such as congestion, route breakages and link failures, that may severely degrade data transfer among nodes. On the other hand, the necessity of distributing basic network functions, such as routing and forwarding, to all participating nodes, combined with energy constraint issues, poses tough challenges from a cooperation point of view. Selfish and/or malicious nodes may misbehave by not adhering to the cooperative paradigm, causing poor performance and low reliability on nodes communication. In the worst case scenario, the network may become partitioned.

Currently, different research areas deal with the complexity typical of ad hoc scenarios. Network faults caused by uncontrollable events are generally addressed at the TCP level, with specific protocol enhancements [1][2][3]. Intentional misbehavior characterized by malice are targeted by the security area [4], while selfishness is the main subject of cooperation enforcing mechanisms [5][6]. Each proposed solution copes well with the target issue. However, the different kinds of misbehavior affect the network functioning in a similar way, producing the same effects (i.e. packet dropping) and yielding degradation of network performance and reliability. Furthermore, in some cases they may interfere with each other. For example, the solution of a cooperation problem can be the cause of a congestion event, since the avoidance of misbehaving nodes in packet delivery can lead to overloading well-behaving nodes.

This paper copes with performability issues of nodes communication, independently from the causes behind service degradation. For the concept of performability we point at the definition given in [7], where performance refers to how *effectively* (i.e. throughput), or *efficiently* (i.e. resource utilization) a system delivers a specified service, presuming it is delivered correctly. On the other hand, reliability reflects the *dependability* (i.e. continuity) of service delivery. We aim at optimizing both performance and reliability measures by improving

- the throughput of data transfer (i.e. service effectiveness) through a lightweight mechanism (i.e. system efficiency);
- the quality of data transfer, so as to provide continuous network connectivity (i.e. service dependability).

To this end, we adopt a simple forwarding scheme, based on multi-path routing, which estimates neighbors' reliability and forwards traffic on most reliable routes. The basic mechanism, called REEF (REliable and Efficient Forwarding) [8], is composed by a reliability estimator and a forwarding policy. Every node keeps a reliability index for each neighbor. This measure is affected by all paths rooted at the pointed neighbor and is updated every time the node sends a packet through it. The updating is positive whenever the packet delivery is successful, negative otherwise. In order to understand whether packets get delivered, we use end-to-end acknowledgments. If data packets are sent relying on the UDP protocol, REEF requires the introduction of a notification system that entails the destination node to send acknowledgments. In case data transfer relies on the TCP protocol (as considered in this paper), REEF uses TCP ACKs as delivery notifications. Although REEF works

This work was partially funded by the Information Society Technologies programme of the European Commission, Future and Emerging Technologies under the IST-2001-38113 MobileMAN project, and by the Italian Ministry for Education and Scientific Research in the framework of the FIRB-VICOM project.

at the network level, it can efficiently retrieve transport layer acknowledgments through a cross-layer interaction with the transport agent. Specifically, REEF is supported by the cross-layer architecture described in [9][10], which allows protocols to exchange information beyond that provided by standard interfaces, and maintaining a clean architectural modularity. After sending a packet, the sender node waits for an ack from the destination node, and then updates the neighbor's reliability. With a set of routes at hand, REEF can select the best route to forward a packet, according to reliability estimates, which reflect the behavior so far observed.

This mechanism has shown significant improvement on network throughput when a simplified transport protocol with acknowledgment is used [8]. However, the impact on transport protocols with congestion control mechanisms, such as TCP, has not been investigated. This is an important aspect because multi-path routing is not always convenient in the ad hoc environment. As studied in [3], and also confirmed in this paper, the plain use of multiple routes may degrade the TCP performance. This is due to intrinsic mechanisms of the TCP protocol. When a TCP sender does not get acknowledgment of sent packets, it reduces the congestion window, causing the retransmission timeout to progressively enlarge, leading to high restart latency and very poor efficiency. Another problem is that the round trip time estimation is not accurate under multi-path routing. TCP senders may prematurely timeout packets which happen to take the longest path. Packets going through different paths may arrive at destination out of order and trigger duplicate ACKs, which in turn may trigger unnecessary TCP congestion window reductions.

The first contribution of this paper is a novel forwarding policy for the REEF mechanism, which improves the network performability, taking into account the several causes of packet dropping, as well as the above mentioned TCP limitations. The main idea is to combine the reliability of a route with its length, so as to keep the advantages of multi-path forwarding, necessary to tolerate network faults or intentional misbehavior, and limit, at the same time, the drawbacks of using TCP over multiple routes.

In addition, this paper provides an accurate performability analysis, which has been carried out with the $ns2$ Network Simulator, to demonstrate the effectiveness of the forwarding mechanism, in realistic environments. To this end, we implemented a multi-path routing protocol, and modified the forwarding agent at the network layer to include the REEF mechanism. Furthermore, we realized a cross-layer interaction between the forwarding and the transport agents to make them exchange information on TCP acknowledgments. We simulated practical situations where nodes execute file transfers (e.g. FTP) over TCP connections. Besides totally cooperative environments, where only network faults, such as congestion and lossy links, may affect the network performance, the mechanism has proved effective also in presence of a variable percentage of misbehaving nodes. Furthermore, nodes mobility is nicely tolerated.

The rest of the paper is organized as follows. Section II gives an overview of the REEF mechanism and describes the new forwarding policy . Section III presents the simulation framework used to carry out the performability evaluation. Section IV details the simulation results on TCP performance, obtained by comparing the standard single path forwarding with our multi-path policy. Finally, Section V draws some conclusions.

II. Overview of REEF

The forwarding mechanism is composed by a reliability estimator and a policy to forward traffic. In the following, we first give a brief overview of the reliability estimator. For a detailed description we point the reader at [8][11]. Then, we describe the new forwarding policy, which is later evaluated with an extensive simulation study.

A. Reliability estimator

Each node keeps track of neighbors' reliability according to its "personal" experience while transferring data. Whenever a node communicates with another node in the network, it estimates the reliability of the neighbor node involved in relaying its packets. Specifically, it maintains a table of sent packets, storing also the identity of the next hop that has been charged with forwarding the packet toward the destination. Then its reliability is estimated according to the delivery result. If the source node receives a TCP acknowledgment, then all the intermediate nodes have correctly forwarded the packet, and hence the reliability of the neighbor node is positively updated. Otherwise, some node on the path misbehaved, and the neighbor's reliability decreases. This means that the reliability of a neighbor node may be affected by a packet loss that was caused by another node on the path, and quantifies the cooperation/performance/reliability so far observed for paths rooted at that neighbor. The simplest way to estimate the reliability of a neighbor j is

$$R_j \leftarrow \alpha R_j + (1-\alpha)M \tag{1}$$

where α, $0 \leq \alpha \leq 1$, is the percentage of the previous estimate that we consider in the current update, and M represents the present delivery outcome and may assume the following values:

$$M = \begin{cases} 0 & \text{if } s \text{ does not receive ack from } d \\ 1 & \text{if } s \text{ receive ack from } d \end{cases} \tag{2}$$

As previously stated, the reliability estimator works at the network layer and uses information coming from the transport layer. This is possible through a cross-layer interaction between the two protocols, which bases on an innovative architecture that standardizes vertical communication between protocols, and allows for several performance optimizations [9][10]. In this specific case, the TCP notifies packet acknowledgments to the forwarding agent through *cross-layer events*, which in turn trigger the update of the relative reliability index (see Section III-B). In this way, the network layer can easily understand the packet delivery status without producing any additional overhead (packet sniffing would result heavy).

B. Forwarding policy

Reliability estimates are useful to choose the best route for packet forwarding. Whenever multiple paths are available, the route with the highest success probability is desired. However, a forwarding policy must be defined keeping in mind also the limitations of TCP over multi-path routing. Sending packets on routes that highly differ for the number of hops toward the destination has proved to be detrimental for network throughput [3]. For this reason, we propose to select paths according to reliability indexes as well as distances to the destination. In order to combine this two factors, we estimate the average number of transmissions, in terms of hops number, to successfully deliver a packet to its destination. Let *n* be the number of hops between the source and destination nodes on a given route, and R_I the reliability index of the first node *I* on that route. Let us approximate $p = R_I$ the probability to succeed while sending a packet through node *I*, and suppose to obtain a successful delivery (represented by the p factor) after $k-1$ failures (see Figure 1). Then, we estimate the average number of times the sender transmits a packet to successfully reach the destination node as:

$$E[N_T] = \sum_{k=1}^{\infty} kp(1-p)^{k-1} \tag{3}$$

As the packet delivery may fail at any hop on the path, we consider the distance n between the source and the destination nodes as an upper bound for the number of transmissions along the path, and estimate the average cost $E[C]$ of transmitting through a specified neighbor as:

$$E[C] \leq n \sum_{k=1}^{\infty} kp(1-p)^{k-1} \tag{4}$$

If we consider that the mean of the geometric distribution is $\frac{1}{p}$, we can re-write the average cost as,

$$E[C] \quad \simeq \quad \frac{n}{p} \tag{5}$$

Thus, we define a new forwarding policy, namely *performability-route* (*p-route*), in the following way. Given $p_i = R_i$ for each neighbor i, and n_i the number of hops to reach the destination through neighbor i, then we choose the route with the minimum cost, in terms of number of transmissions, $\min_i \{\frac{n_i}{p_i}\}$, and if multiple routes have the same value, we randomly choose one of them. This choice is made on a per-packet base (as provided for the route selection by standard forwarding agents), and is repeated by each intermediate hop between the source and the destination nodes, whenever the routing protocols provides only the next hop toward the destination. Furthermore, with this policy, we consider routes that require the minimum (estimated) number of transmissions to successfully reach the destination, and avoids next-hops that are far from the destination.

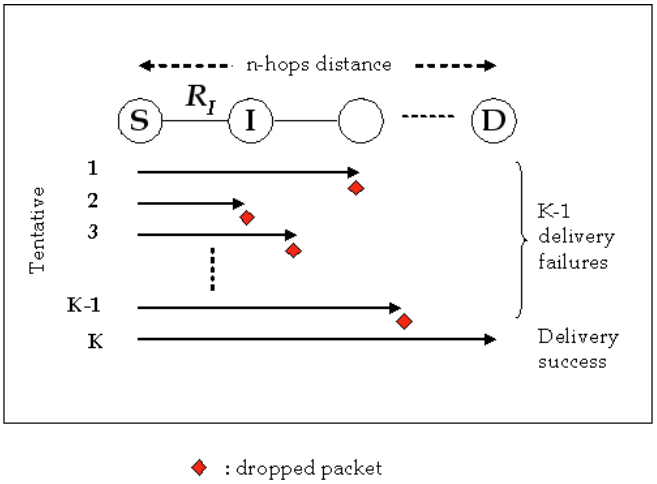


Fig. 1. Example of a packet delivery success between nodes S and D, after k-1 failures. At each tentative the packet may be dropped at any hop between the source and destination nodes.

C. Performability requirements and goals

As previously stated, performability of data transfer involves service efficiency and effectiveness for measuring performance, and service dependability or continuity for judging the level of reliability. We identify the efficiency as a design requirement for the forwarding mechanism, while service effectiveness and dependability are goals to achieve.

Service *efficiency* focuses on resource utilization and asks for computational lightness. Our forwarding mechanism is based on a cross-layer architecture [9][10] that allows to easily retrieve transport layer acknowledgments, without any additional effort (i.e. packet sniffing or watchdog mechanisms would results heavy [5]). Furthermore, resources employed to keep and update reliability indexes are minimal: a single value is stored for each neighbor. However, it is worth noting that the hypothesis on multi-path routing brings some overhead in terms of network traffic. Hence, in order to satisfy the need for service efficiency, an evaluation of resource utilization (in terms of network resources) is necessary. To this end, we perform a simulation study (see Section IV-A), showing that the overhead produced by multi-path and single path routing protocols are comparable.

Service *effectiveness* and *dependability* are achieved through the multi-path forwarding policy that distributes traffic among most successful routes and avoids misbehaving nodes. The idea is to choose the route with a low number of hops and high reliability, so as to to minimize the number of transmissions needed to reach the destination. The policy is such that as soon as the reliability of a path decreases, because the sender observes packets losses, and the ratio of its distance to the destination over the reliability of the next node is no more the most convenient, a different (more reliable) route is used, even if it is longer. This allows to go round misbehaving points, providing continuous network connectivity. In fact, in case of nodes that are congested, selfish, or malicious,

packet dropping may lasts for a while, paralyzing the data transfer. Furthermore, spreading traffic among routes, with the same level of success, operates a load balancing that, in turn, reduces the possibility of congestion, as well as the motivations for selfish behavior. As a consequence, the throughput and the quality of data transfer are increased (see Section IV), thanks to a fairly distribution of traffic on different paths, and the avoidance of misbehaving nodes that cause service interruption.

III. Evaluation framework

To realize a simulation framework, suitable for a complete evaluation of the reliable forwarding components, we used the Network Simulator $ns2$ (v. 2.27) [12], and a library of objects and abstractions provided by the Naval Research Laboratory (i.e., ProtoLib) [13], which includes an implementation of the Optimized Link-State Routing protocol (OLSR) [14]. OLSR is a well-established proactive protocol of the Manet IETF working group, which suites our cross-layer architecture and supports several cross-layer optimizations [9][10]. An example is given in [15], where the performance of the Gnutella protocol is enhanced through a closer cooperation with the routing agent.

Starting from the $ns2$+ProtoLib basement, we were able to first introduce the cross-layering concepts described in [10] and, therefore, use them to develop a reliable forwarding agent at the network layer. As detailed in the following section, we extended OLSR in order to have a multi-path version of it, and also re-programmed agents at transport layer (i.e., TCP agents) so as to send cross-layer events for positive and negative acknowledgments. We used TCP-Reno with Delayed Ack. Furthermore, we modified the forwarding agent at the network layer in order to:

1) catch cross-layer events coming from transport agents;
2) maintain reliability tables according to the notified events, and the routes discovered by the OLSR routing agent;
3) implement the REEF forwarding policy on the reliability tables.

A. Multi-Path OLSR

Performance of packet forwarding is dependent on the ability of the REEF mechanism to utilize alternative routes when it detects non-operational ones. The availability of redundant routes is usually not provided by common routing protocols that, typically, build shortest-path routing tables. For this reason, we implemented a multi-path extension of the OLSR protocol provided by the ProtoLib package.

OLSR is a proactive routing protocol, based on a link state algorithm and optimized for mobile ad hoc networks. It minimizes the overhead from flooding of control traffic by using only selected nodes, called Multipoint Relays (MPRs), to send and retransmit topology control (TC) messages. When a node broadcasts a packet, only its MPR set rebroadcasts the packet, while other neighbors simply process the packet. TC messages are sent by a node in the network to declare a set of links, called advertised link set, which must include at least the links to all nodes of its MPR Selector set (i.e. the neighbors which have selected the sender node as a MPR). This is sufficient information to ensure the computation of a routing table based on a shortest path algorithm. By increasing the amount of information included in the TC messages, and the number of node sending them, it is possible to build a multi-path routing table. In particular, the requirements for multi-path routing are: i) the advertised link set of the node is the full neighbor link set; ii) besides MPRs each node having at least one neighbor must send TC messages. In practice, these two requirements are easily satisfied by the "all links" feature implemented in the ProtoLib's OLSR agent.

Our contribution to realize multi-path routing was to enhance the OLSR agent implementation with a new procedure, $MakeNewMultiPathRoutingTable$, that is based on a breadth-first logic. The first step is to add to the routing table all symmetric neighbors (with hop distance h=1) as destination nodes. Then, for each added route, we go through the topology set to build all routes of length 2 (h=2). Again, starting from this new set of routes, all routes 3-hops long are built, and so on. The procedure is repeated starting from the set of routes just added (h=n) and building the set of routes 1-hop longer (h=n+1), until there are no more routes to build. Obviously, in case of multiple routes to a destination passing through the same neighbor we consider only the shortest one.

B. Introducing cross-layer interactions

After patching the network simulator with the ProtoLib and the multi-path OLSR implementation, we introduced also a set of primitives to allow cross-layer interactions. Specifically, the realization of the forwarding mechanism in our evaluation framework involves the introduction of a class of cross-layer events of type *Recv TCP-ack/nack*, to which the forwarding agent subscribes for notifications coming from a local TCP agent. These events notify the forwarding agent about delivery outcomes of packets related to connections between the local host and a foreign party. In particular, the TCP agent sends a TCP-ack event to the forwarding agent whenever it receives a valid acknowledgment. Instead, TCP-nack events are caused by packets retransmissions and generated when: 1) a packet timeout expires; 2) three duplicate acknowledgments on the same packet are received. An event notification causes the forwarding agent to update the reliability index associated to the neighbor through which the packet passed. The update is positive for TCP-ack and negative for TCP-nack.

In order to easily relate notified events with neighbor nodes, the forwarding agent keeps a transmission list containing for each sent packet, the TCP sequence number and flow identity, plus the neighbor through which the packet was sent. When an acknowledgment is notified, the forwarding agent looks for the corresponding packet in the transmission list, in order to retrieve the neighbor that relayed it. Then, it updates the reliability index of such neighbor according to the entity of the event. Packets are stored in the transmission list before being sent, and removed after the reception of a

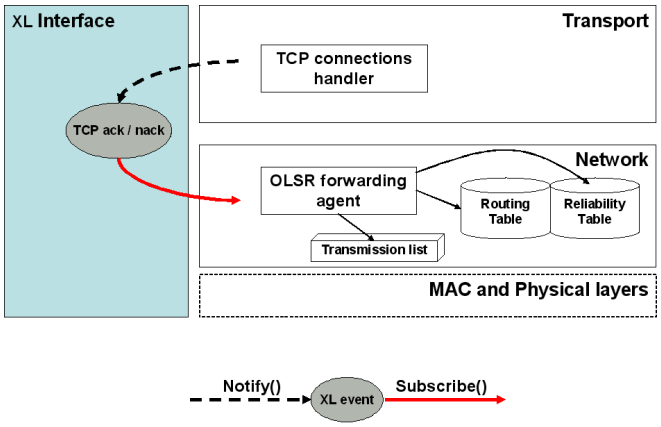


Fig. 2. Cross-layer interaction between the network and the transport layers.

Recv-ack/-nack event. If the received ack is cumulative (i.e., it acknowledges the reception of multiple consecutive packets), then the forwarding agent makes an update for each entry in the transmission list with the sequence number lower or equal to the received ack.

Reliability indexes are maintained in a table containing an entry for each neighbor. In order to keep the reliability table constantly up-to-date, we trigger an update every time there is a change in the neighbor list. Figure 2 shows the resulting system architecture, with the cross-layer interaction between the forwarding and TCP agents.

IV. Performability evaluation

To evaluate the performability of data transfer on top of REEF, we consider the following metrics.

Overhead Since the cost of internal computation in terms of space and energy consumption is negligible compared to the cost of transmission, we look at the overhead caused by extra routing messages, measured in Bytes/sec. Hence, routing overhead represents a measure of service efficiency.

TCP sequence number The sequence number of TCP packets acknowledged by the destination, as function of time, is a measure of both performance and reliability. In the first case, a comparison of TCP sequence numbers between different forwarding policies allows to understand which one performs better (i.e. effectiveness). In the second case, a constant increase of TCP sequence numbers shows continuous network connectivity, while a flat line on the plot indicates an interruption of packet delivery (i.e. reliability).

Throughput We refer to the TCP throughput as the amount of the data correctly received by TCP destination nodes, in a time unit. This metric is useful to quantify the effectiveness of the forwarding mechanism in presence of TCP data transfer.

In the following, we first evaluate the routing overhead produced by our multi-path OLSR. This study is important to demonstrate that it is possible to improve TCP performance at the expense of very low routing overhead. After demonstrating that multi-path routing adds a reasonable amount of overhead, we investigate the impact of REEF on TCP traffic. Performability improvements are analyzed from two perspectives. With a transient analysis, we observe TCP connections on time intervals, in order to check whether our policy improves the quality of TCP data transfer. On the other hand, with a steady-state analysis, we show statistic results on general scenarios, which measure, for example, the total network throughput by varying the percentage of misbehaving nodes, and the nodes mobility. In order to simulate nodes misbehavior, we used the `SelectErrorModel` class, provided by $ns2$, that allows to selectively discard packets, by indicating the packet type (e.g., tcp) and the dropping frequency. Simulations are based on TCP-Reno agents with Delayed Ack.

A. Single vs. multi-path routing

Instructing the routing agent to calculate multiple routes may cause additional overhead, depending on the nature (proactive or reactive) of the protocol. For example, simulation studies on *reactive* protocols show that there are significant advantages with multi-path routing [16][17]. Their findings demonstrate that the number of route discoveries and hence of routing load decreases, even though end-to-end delay of data packets slightly increases. Further results show that multi-path routing allows to achieve faster and efficient recovery from route failures in highly dynamic networks.

In this paper, we consider OLSR, a *proactive* routing protocol, and evaluate the additional overhead induced by a multi-path version of it. In the implementation of the multi-path OLSR we identified the following as the main requirements: i) the advertised link set of the node must be the full neighbor link set; ii) besides MPRs each node having at least one neighbor must send TC messages. Consequently, the additional overhead produced by the multi-path version is mainly determined by the amount of increased topology information traveling through the network, and the number of generated TC messages. Hereafter, we go through a measurement study to quantify this overhead.

To evaluate the performance of multi-path OLSR with respect to its legacy version, we simulated a range of network scenarios. Figure 3 shows the mean overhead (Bytes/sec.) produced by the two protocols, varying the network size. This metric is measured as the mean total number of bytes sent by all network nodes. In particular, we evaluated routing overhead for different network sizes, respectively 10, 20, and 40 nodes. For each network size, we created three different scenarios, with increasing routes length. To this end, nodes are randomly placed in an area of different shapes: from a square to thin rectangles. Results are averaged on the three scenarios. Simulation time is 900 seconds, and Hello and TC message intervals are respectively 2 and 5 seconds. Figure 3 shows that in a static environment, the additional overhead produced by multi-path OLSR is almost independent of the number of nodes, and is around 15%. Considering that in the case of 40 nodes we have less than 2000 Bytes/sec of added routing load,

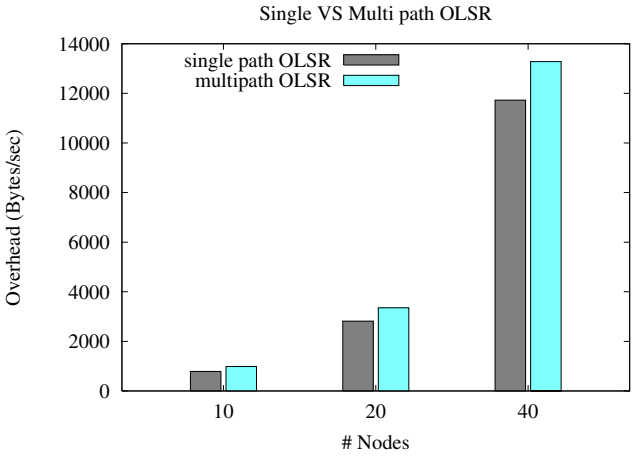


Fig. 3. Routing overhead in a static network.

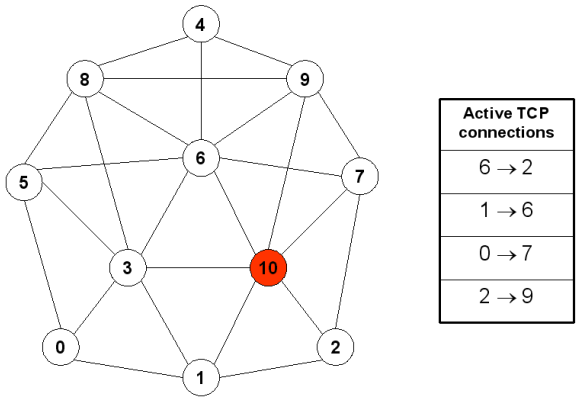


Fig. 4. Simulated network.

we can state that multi-path routing adds a reasonable amount of overhead, and hence we can have a good trade-off between costs and benefits of our multi-path forwarding.

B. The impact of REEF on TCP: A transient analysis

The objective of this evaluation is to find out whether REEF may provide higher throughput and better network connectivity to TCP traffic, with respect to the conventional case in which packet forwarding is based on single path routing (OLSR), where the shortest route is always chosen. In particular, we focus on networks affected by fault conditions and misbehaving nodes that forward traffic in an intermittent fashion.

1) Partially misbehaving network: With this study, we aim at evaluating REEF's effectiveness, investigating the quality of single TCP connections, on a time interval. To this end, we consider a small network, composed of a dozen of nodes, with routes that are 3-4 hops long. The small size of the network allowed us to Analise in details the behavior of single nodes and connections. Evaluations on a larger network have been conducted and are detailed in Section IV-C.

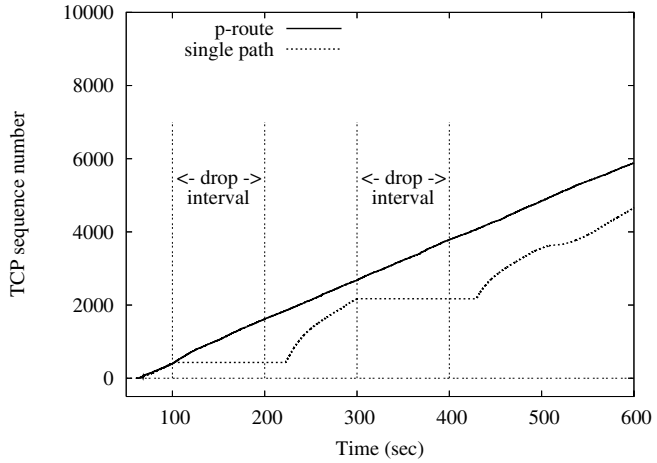
The simulated network is composed of 11 nodes, on a 600 by 600 square meters area (see Figure 4). One node in the network (i.e. node 10) behaves as on/off forwarder[1]: it does not relay traffic, from second 100 to 200, and from 300 to 400. This behavior can be typical of a selfish or a malicious node, as well as a fault condition. We configured 4 TCP connections with an FTP application on top of each as traffic generator. As FTP produces bulk data to send, it may cause situations of congestion. In the simulation, all FTP agents start around time 60 and last for the whole simulation run, which is 600 seconds. In particular, FTP transfers are active between nodes 6 and 2, 1 and 6, 0 and 7, and 2 and 9.

To check TCP behavior, we analyzed the sequence number of packets received by the destination nodes (i.e. FTP clients), as function of the application lifetime. In this way, it is possible to understand how the forwarding policies react to nodes misbehavior, and what their impact is on active connections.

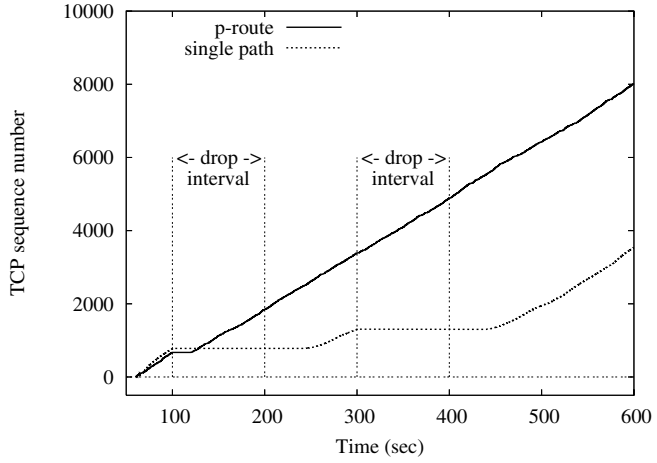
The results from this simulation study have validated the performance improvement achieved by the *p-route* policy. Besides Figure 6(a), which deserves to be discussed apart, Figures 5(a), 5(b), and 6(b) illustrates how the *p-route* policy outperforms single path forwarding. As shown by diagrams, not only *p-route* achieves the higher sequence number, increasing it up to 100% (Fig. 5(b) and 6(b)), but it also provides better service delivery. In fact, *p-route* offers continuous network connectivity regardless of nodes misbehavior. It uses alternative paths, and hence avoids the delivery interruption, while the other policy shows an evident discontinuous behavior when the selfish node discards packets: flat lines on misbehavior intervals (i.e. 100-200s and 300-400s) indicate a complete inability of nodes to exchange data.

Looking at the TCP connection between nodes 0 and 7 (see Figure 6(a)), it is evident that both policies obtain similar results, even if in different ways. Performance on this connection is very poor as the maximum sequence number obtained is around 600, while the other connections get up to 8000. The reason for this behavior does not directly depend on the applied forwarding policy. Instead, it is due to a combination of factors such as network topology, active connections, and nodes congestion. First of all, we remark that packet loss has detrimental effects on TCP performance, as the congestion window is reduced and the TCP retransmission timeout becomes progressively larger leading to high restart latency and very poor efficiency. Hence, the more packets get lost, the higher is the degradation of the involved TCP connections. Furthermore, the connection between nodes 0 and 7 relies on a route that is longer than the others. Packets have to traverse more intermediate hops, increasing chances of getting into congested nodes. In fact, most of nodes between 0 and 7 are overloaded because of the other active FTP transfers, and hence there is no way to go around the problem. With single path forwarding, the connection experiences long pauses that correspond to misbehavior intervals. With *p-route*, connections get the best service, without transfer interruptions. However,

[1]The node participates to the routing function but does not always forward packets on behalf of other nodes.

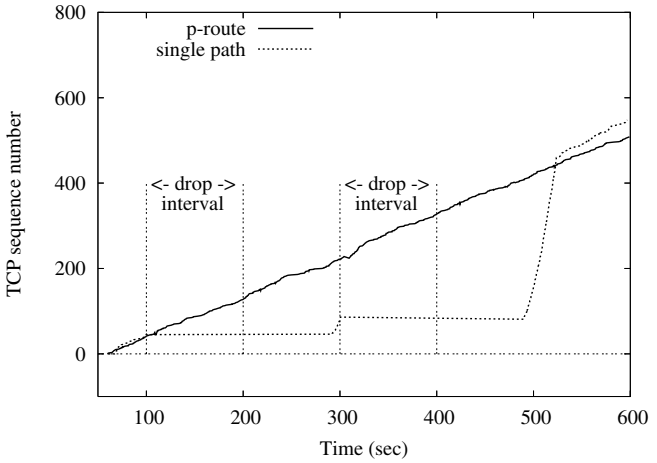


(a) Node 2 is FTP client of node 6.

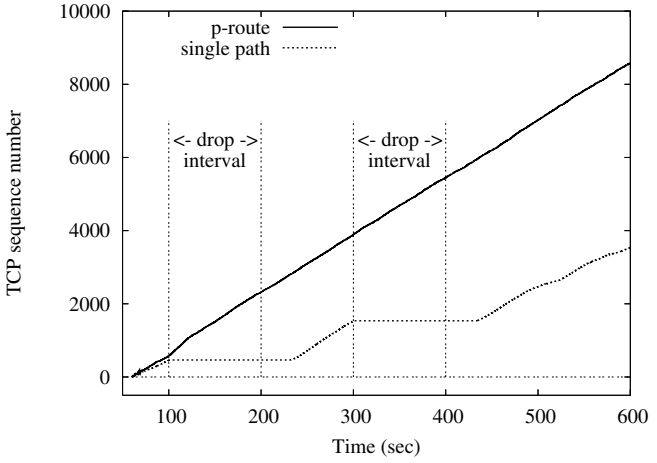


(b) Node 6 is FTP client of node 1.

Fig. 5. Sequence number of TCP packets received by FTP clients as function of time, in presence of a misbehaving node.



(a) Node 7 is FTP client of node 0.



(b) Node 9 is FTP client of node 2.

Fig. 6. Sequence number of TCP packets received by FTP clients as function of time, in presence of a misbehaving node.

the last TCP sequence number is quite low. From an analysis of trace files, we observed that the connection is affected mainly by congestion events. This causes poor performance because the neighbors of the sender node do not have knowledge of current network conditions. As they are not involved in end-to-end communications, they do not update their reliability, and hence unconditionally use shorter (but congested) routes. Analyzing more in details the TCP connection between node 0 and 7, the sender can communicate with the receiver through neighbor nodes 3, 5, and 1. In case the delivery relies on neighbor 3, this must choose one route toward the destination. As all neighbors of node 3 have the same reliability, node 3 chooses 6 or 10 as they provide a shorter route. Unfortunately, both of them are not reliable, because the former is overloaded and the latter is misbehaving. Node 6 would be chosen even in the case the communication occurs through neighbor 5. Packets find similar obstacles through neighbor 1, because node 10 is misbehaving and node 2 is involved in the other two connections. The only way to successfully deliver packets to the destination would be through the nodes on the border of the network (i.e. 5, 8, and 9), but this path is longer and intermediate nodes do not have knowledge in order to deviate traffic there. Hence, the protraction of intermittent losses (experienced on almost all used routes) slows down the TCP sender that stabilizes on low sending rate.

Figure 7 shows the total TCP throughput of FTP clients, as a function of time. In this case, we also report the results obtained with a *load-balancing* policy, which equally spreads traffic among all possible routes to the destination. We remark that this policy, as well as our *p-route* policy, makes a

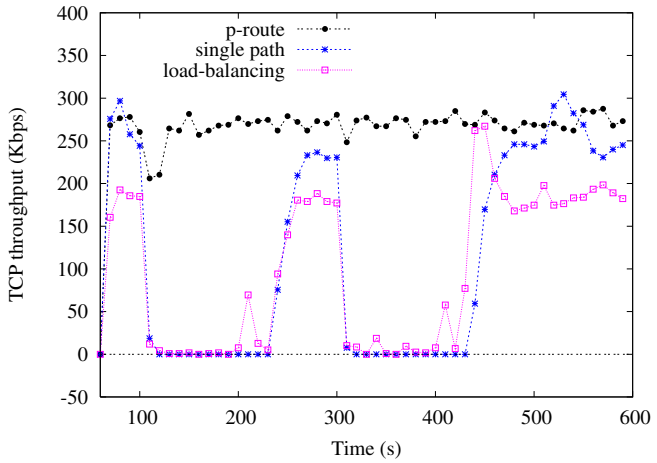
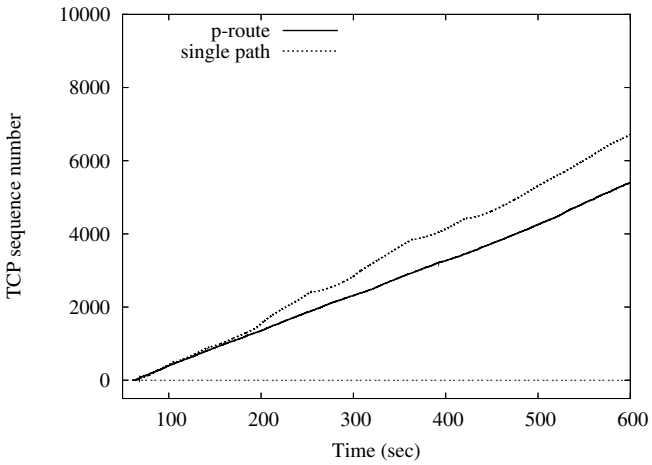
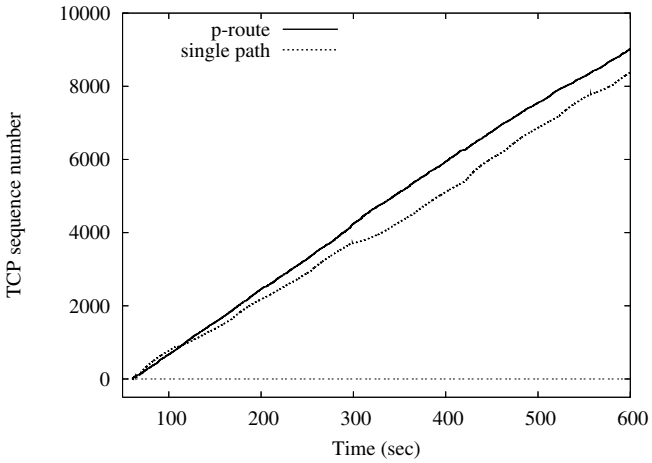


Fig. 7. TCP throughput on FTP clients obtained by applying the different forwarding policies, in presence of a misbehaving node.



(a) Node 2 is FTP client of node 6.



(b) Node 6 is FTP client of node 1.

Fig. 8. Sequence number of TCP packets received by FTP clients as function of time, in a network without misbehaving nodes.

per-packet and per-hop choice. *P-route* achieves the higher throughput and keeps it constant for the whole time, showing high tolerance to misbehaving nodes. Instead, the other forwarding policies are more sensitive to packets dropping; as soon as the misbehaving node starts discarding packets, both single-path and load-balancing experience a sharp drop of network throughput. This result confirms the ineffectiveness of a plain multi-path forwarding, as studied in [3].

In conclusion, in presence of misbehaving nodes, the *p-route* policy significantly improves the performance and reliability of TCP connections, achieving a twofold advantage: 1) the amount of packets successfully delivered at destination is increased up to 100%; 2) TCP connections benefit from a delivery service of higher quality, which provides continuous connectivity to the communicating end-points, hiding the effects of misbehaving nodes.

2) Totally cooperative network: To show that REEF yields better performance even in absence of misbehaving nodes, we repeated the simulation on the same network scenario (see Figure 4), with the difference that all nodes cooperate to packet forwarding. In this case, packet loss can be caused only by temporary fault conditions. Behavior of TCP connections is depicted in Figures 8 and 9. Plots show that the *p-route* policy is globally better than single path forwarding, as it almost always achieves the highest sequence number; the only exception is for the connection between nodes 6 and 2 (see Figure 8(a)).

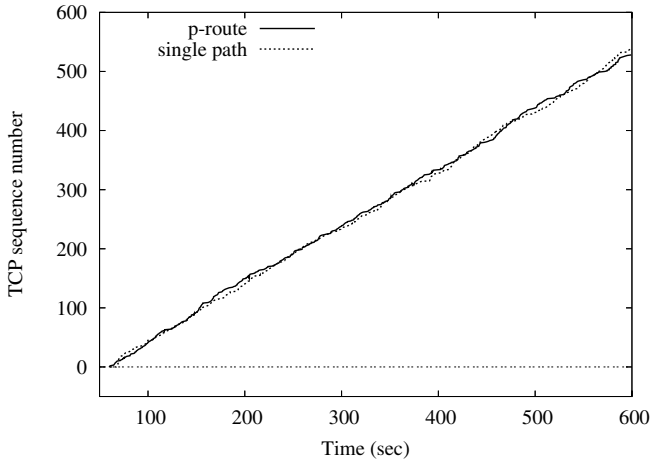
The connection between nodes 0 and 7 shows results similar to the case with the misbehaving node. The low performance for both policies confirms the motivations previously stated. This connection suffers from congestion events on intermediate nodes, and the higher distance between source and destination increases the negative effects.

Finally, Figure 10 shows the predominance of *p-route* over single path. The achieved throughput is on average higher and more uniform for *p-route*, while single path forwarding presents wide fluctuations, staying often below the *p-route* curve. The load-balancing policy again performs worse than single path.

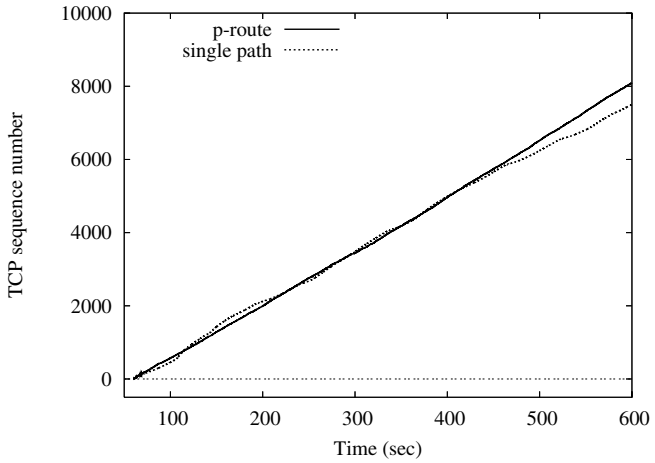
C. The impact of REEF on TCP: A steady state analysis

With a steady-state analysis, we show aggregate results on general scenarios, to evaluate the scalability of the REEF mechanism. Specifically, we measure the average total network throughput as a function of the percentage of misbehaving nodes, and mobility.

The experiments have been conducted in various network scenarios. We fixed the network size to 20 nodes, placed in a 1000x700 area, and configured 5 TCP connections. We used Telnet sessions as traffic generators. Packets inter-arrival times are chosen from an exponential distribution with average 0.2 seconds. All TCP connections are established at time 60 (to allow the routing table construction), and last for the whole



(a) Node 7 is FTP client of node 0.



(b) Node 9 is FTP client of node 2.

Fig. 9. Sequence number of TCP packets received by FTP clients as function of time, in a network without misbehaving nodes.

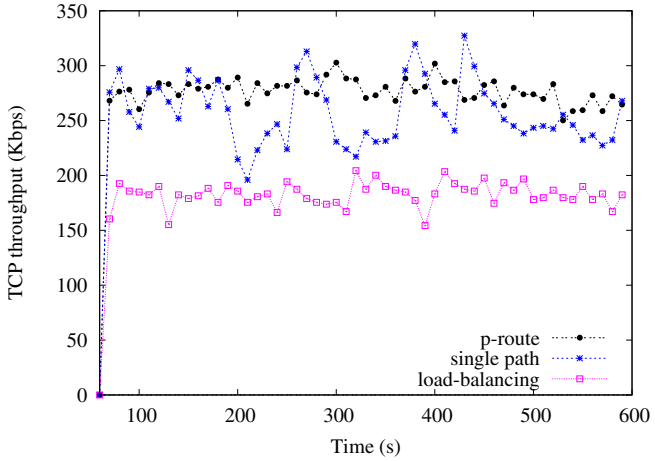


Fig. 10. TCP throughput on FTP clients obtained by applying *p-route* and single path forwarding, in absence on intentional nodes misbehavior.

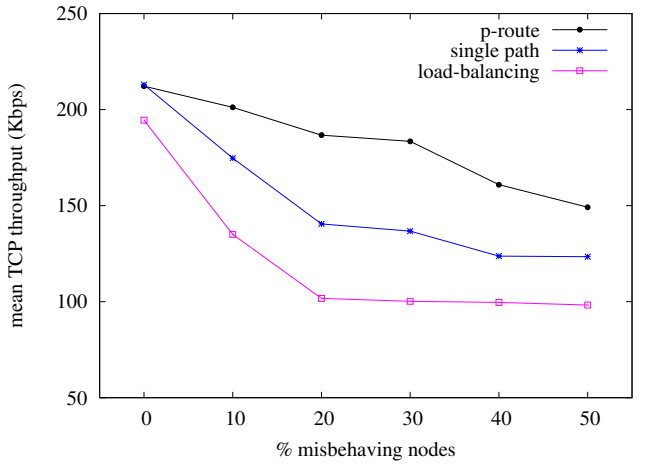


Fig. 11. Mean TCP throughput in a network of 20 nodes with 5 TCP connections as function of the percentage of misbehaving nodes.

simulation time that is 15 minutes (900 s). Connection endpoints are generated randomly and, for each scenario, 10 runs are performed to decrease the impact of randomness. We then introduced misbehaving nodes that cooperate to routing (and hence they appear in routing tables of the other nodes) but do not forward TCP traffic. The experiments were repeated for the different forwarding policies and the presented results are the average of the 10 runs. Hereafter, we present the performance analysis of our forwarding policy according to the aforementioned parameters.

Percentage of misbehaving nodes The first set of experiments aimed at evaluating the effect of an increasing percentage of misbehaving nodes on the network throughput. To this end, we produced random scenarios with increasing number of misbehaving nodes, which are not endpoints of TCP connections. We created 6 different scenarios, with respectively 0, 2, 4, 6, 8, 10 misbehaving nodes (i.e., 0% to 50%). Figure 11 shows that when TCP works on top of load-balancing, it always behaves worse than using single path, regardless of the percentage of misbehaving nodes. This negative result is caused by the combination of two factors: i) spreading traffic among multiple routes increases the possibility to run into unreliable routes (with consequent packet loss); ii) TCP sensitively reacts to packet loss, decreasing the transmission rate. Hence, some communications encounter misbehaving nodes, even if the shortest path between the endpoints is free of them. This causes TCP to slow down the sending rate, with consequent performance degradation. On the other hand, *p-route* outperforms single path forwarding whenever misbehaving nodes are present, while the two method are comparable in cooperative networks (i.e. the percentage of misbehaving nodes is 0). Specifically, the performance gain grows up to 50% in

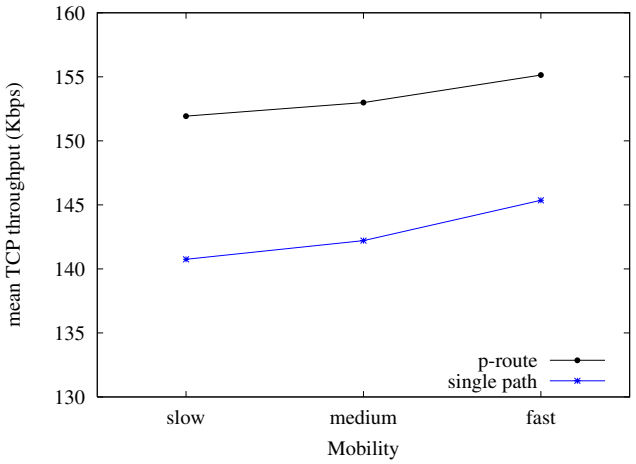


Fig. 12. Mean TCP throughput as function of nodes mobility in a network of 20 nodes with 5 TCP connections and 30% of misbehaving nodes.

the case of 20% and 30% of misbehaving nodes. This is a visible improvement.

Nodes mobility In the following set of experiments we wanted to study the effects of nodes mobility on the total network throughput. To this end, we generated random way-point mobility scenarios using the *set-dest* utility shipped with $ns2$. Considering a fixed population of 20 nodes moving over a rectangular area of 1000 by 700 square meters, with nodes speed uniformly ranging inside [1,5] m/s, we created three sets of mobility scenarios: 1) a *slow* scenario with pause times up to 10 seconds; 2) a *medium* scenario with pause times up to 5 seconds; 3) a *fast* scenario where nodes continuously move as the pause time is set to 0. The percentage of misbehaving nodes is set to 30% (i.e., 6 nodes discard TCP traffic).
Figure 12 shows that even in presence of nodes mobility the *p-route* policy globally achieves better performance than single path forwarding, with a constant increase in network throughput around 10%. This is a significant result because *p-route* is able to distribute traffic among multiple routes even a dynamic environment.
It is worth noting that throughput increases while mobility goes up. This effect is caused by the random way-point mobility model that tends to group nodes in the middle of the simulation area, making nodes closer and routes shorter. A more dense network allows nodes to easily reach each other, increasing their ability to communicate. Intuitively, we believe that *p-route* performance can be further improved by providing the REEF mechanism with a cache on reliability indexes. Instead of deleting a node from the reliability table as soon as it is not anymore a neighbor, the idea it to keep it for some time, so as to remember its reliability value in the case it appears again as neighbor node. Ongoing work is evaluating this caching mechanism on different mobility models, such as Group and Manhattan mobility [18].

V. Conclusions

The performability of data transfer in ad hoc environments is highly sensitive to packets loss, that may be caused by several factors, such as congestion and lossy links, as well as selfish and malicious nodes. When designing protocols for nodes communications, special care has to be taken to consider all the causes that may degrade the system performance. In the case of TCP traffic, even the use of multiple paths may have negative effects on the network throughput.

This paper proposes a new multi-path forwarding policy, *performability-route* (*p-route*), which address nodes misbehavior and network faults. Focusing on TCP traffic, we show how the *p-route* policy tolerates the negative effects that packet loss has on the protocol behavior, and overcomes the limitations of using multiple paths for TCP packets. To show such improvement, we carried on a simulation analysis in realistic environments, with a multi-path routing protocol, and TCP file transfers. We also simulated nodes misbehavior, so as to investigate the effectiveness of our forwarding policy in both cooperative and partially misbehaving networks. The performability achieved with the *p-route* policy is compared with the standard single path forwarding, which chooses always the shortest route. Simulation outcomes elect the *p-route* policy as the more efficient forwarding method, as it shows good tolerance to packets loss, maintaining a satisfactory level of connectivity among nodes, and avoiding delivery interruptions typical of single path forwarding. By increasing the percentage of misbehaving nodes (from 0% to 50%), *p-route* maintains a high level of efficiency, with a TCP throughput improvement up to 50%. The predominance of *p-route* is visible even in comparison with a plain multi-path forwarding, or in presence of nodes' mobility.

One may argue that the *p-route* policy may not make the correct decision in choosing alternative paths, as in some cases the one-link reliability does not correctly capture the reliability of the whole path from a sender to a specific destination. This is possible when some nodes do not communicate spontaneously (open TCP connections) with other parties, and hence have no mean for updating their reliability indexes. In REEF jargon, this is equivalent to have "blind" nodes, where reliability indexes are set to the initial value, and nodes are unable to distinguish the proximity of misbehaving nodes. This behavior has been observed in the simulation study (see Section IV-B). Obtained results show that choices made by blind intermediate nodes do not cause inefficiency, in comparison with single path forwarding, and result in a multi-path forwarding mechanism that favors shorter routes.

Future work will evaluate our policy in more dynamics environments (e.g. Manhattan and Group mobility models), and integrate in the simulation framework a scheme that assigns priority to packets in the forwarding queue, depending on the reliability of sender nodes, in order to enforce cooperation among nodes [8].

REFERENCES

[1] G. Holland and N. Vaidya, "Analysis of TCP performance over mobile ad hoc networks," *Wireless Networks*, vol. 8, no. 2/3, pp. 275–288, 2002.

[2] F. Wang and Y. Zhang, "Improving TCP Performance over Mobile Ad-Hoc Networks with Out-of-Order Detection and Response 2," in *Proceedings of The Third ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, Lausanne, June 2002.

[3] H. Lim, K. Xu, and M. Gerla, "TCP Performance over multipath routing in mobile ad hoc networks," in *Proceedings of the IEEE International Conference on Communications (ICC)*, May 2003.

[4] L. Buttyan and J. Hubaux, "Report on a Working Session on Security in Wireless Ad Hoc Networks," *Mobile Computing and Communications Review*, vol. 6, no. 4, 2002.

[5] S. Buchegger and J. Y. L. Boudec, "Performance analysis of the CONFIDANT protocol," in *Proceedings of The Third ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, Lausanne, June 2002.

[6] P. Michiardi and R. Molva, "Analysis of coalition formation and cooperation strategies in mobile ad hoc networks," *Ad Hoc Networks Journal*, vol. 3, no. 2, pp. 193–219, Mar. 2005.

[7] J. Meyer, "Performability evaluation: where it is and what lies ahead," in *IEEE Internation Computer Performance and Dependability Symposium (IPDS'95)*, Erlangen, Germany, 1995.

[8] M. Conti, E. Gregori, and G. Maselli, "Reliable and Efficient Forwarding in Ad Hoc Networks," *Ad Hoc Networks Journal, Elsevier*, to appear.

[9] M. Conti, G. Maselli, G. Turi, and S. Giordano, "Cross-Layering in Mobile Ad Hoc Network Design," *IEEE Computer, special issue on Ad Hoc Networks*, vol. 37, no. 2, pp. 48–51, 2004.

[10] M. Conti, J. Crowcroft, G. Maselli, and G. Turi, "A Modular Cross-Layer Architecture for Ad Hoc Networks," in *Handbook on Theoretical and Algorithmic Aspects of Sensor, Ad Hoc Wireless, and Peer-to-Peer Networks*, CRC, Ed., July 2005.

[11] M. Conti, E. Gregori, and G. Maselli, "Towards Reliable Forwarding for Ad Hoc Networks," in *Proceedings of the 8th IFIP-TC6 International Conference on Personal Wireless Communications (PWC 2003)*, Venice, Italy, Sept. 2003, pp. 790–804.

[12] "The Network Simulator - ns-2," http://www.isi.edu/nsnam/ns/.

[13] "PROTEAN Research Group," http://cs.itd.nrl.navy.mil/5522/.

[14] T. Clausen and P. Jacquet, "Optimized Link State Routing Protocol (OLSR)," RFC 3626, Oct. 2003.

[15] M. Conti, E. Gregori, and G. Turi, "A Cross-layer Optimization of Gnutella for Mobile Ad Hoc networks," in *Proceedings of The 6th ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, Urbana-Champaign, IL, USA, May 2005.

[16] A. Nasipuri, R. Castaneda, and S. Das, "Performance of Multipath Routing for On-Demand Protocols in Mobile Ad Hoc Networks," *ACM/Kluwer Mobile Networks and Applications (MONET)*, vol. 6, no. 4, pp. 339–349, 2001.

[17] M. Marina and S. Das, "On-demand multipath distance vector routing in ad hoc networks." in *Proceedings of IEEE International Conference on Network Protocols (ICNP)*, 2001, pp. 14–23.

[18] F. Bai, N. Sadagopan, and A. Helmy, "IMPORTANT: A framework to systematically analyze the Impact of Mobility on Performance of RouTing protocols for Adhoc NeTworks," in *Proceedings of INFOCOM 2003*, San Francisco, CA, USA.