Fault-Tolerant Routing in Mobile *Ad Hoc* Networks

B. John Oommen[1,2] and Luis Rueda[3]
[1]*School of Computer Science, Carleton University, Ottawa;*
[2]*University of Agder, in Grimstad,*
[3]*School of Computer Science, University of Windsor,*
401 Sunset Avenue, Windsor, Ontario, N9B 3P4,
[1,3]*Canada*
[2]*Norway*

1. Introduction

Mobile *Ad Hoc* Networks (MANETs) are characterized by the cooperative engagement of mobile nodes that constitute networks possessing continuously-changing infrastructures, the absence of centralized network managers, access points, fixed base stations, a backbone network for controlling the network management functions, and the absence of designated routers for making routing decisions. All the nodes in MANETs participate in the routing process by acting as routers for one another. However, for the transmission of data from one node to another, such networks normally require several hops because of the limited wireless transmission range associated with the operation of the mobile nodes [2,7,9].

The above-mentioned characteristics of MANETs, particularly those arising due to the mobility of the nodes, and the continuously-changing network infrastructure, pose several challenges. Due to the continuously changing infrastructure, the routes that were once considered to be the "best" may no longer remain as the "best" at a later time instant. Therefore, one needs to continuously re-compute the routes, implying that in such networks, there is no permanent convergence to a fixed set of routes. Thus, any routing protocol that needs to operate in MANET network environments should take these issues into consideration [2].

Designing routing protocols poses further challenges when one needs to design routing schemes in the presence of adversarial environments in MANET networks. This is the primary focus of this chapter. More specifically, we discuss fault-tolerant routing schemes when the network contains malfunctioning nodes. To motivate this, we observe that most existing MANET protocols were postulated considering scenarios in which all the mobile nodes in the *ad hoc* network function properly and in an idealistic manner. However, adversarial environments are common in MANET environments, and misbehaving nodes degrade the performance of these routing protocols [11]. The need for fault-tolerant routing protocols was identified to address routing in adversarial environments in the presence of faulty nodes by exploring redundancies in the networks [10,11].

Despite the challenges that we mention above, it is worthwhile to note a few applications of MANETs which have made them popular. One of the popular application domains of

MANETs is communications in moving battlefields [7]. Other applications may be found in rural regions where building up fixed wired or wireless infrastructures can be costly and/or difficult.
Although our primary discussion centers around fault-tolerant routing in MANETs, since this chapter is intended to be of a survey nature, we shall first briefly include an overview of the field and the corresponding routing protocols.

2. Routing protocols for MANETs

Routing in MANETs is currently a challenging and interesting problem studied by the community primarily due to the dynamic nature of the infrastructure present in MANETs, e.g., due to nodes joining and leaving the network. For routing, the transmission of data from one node to another is *direct,* if the source and destination nodes are neighbors, i.e., if they are within the wireless range of each other. On the other hand, the transmission is *indirect*, if the source and destination nodes are not within their range of operation [7]. In such a case, routing is achieved through a series of multiple hops, with intermediate nodes between the source and the destination nodes serving the purpose of routers for relaying the information in between. The dynamic nature of the topology of MANETs due to the constant migration of nodes renders routing considerations difficult. The following characteristics of MANETs make their routing further challenging [7]:

1. The terrain in which the mobile nodes operate in MANETs may pose to be hostile with hazardous conditions that can lead to the frequent failure of the nodes and their mutual links.
2. The medium of transmission of information in MANETs is wireless. Wireless media are relatively unreliable, insecure, and quite susceptible to different kinds of errors and unwanted noise.
3. MANETs operate with battery-powered nodes, which are normally low powered, and resource constrained. If the region of operation of the nodes is in a hostile terrain, the frequent recharging of the nodes may not always be feasible. Consequently, all routing algorithms should be energy-efficient, of low complexity, and should be capable of operating under limited bandwidth.

The different types of errors that can occur in MANETs are the following [7]:

1. Transmission errors
2. Node failures
3. Link failures
4. Route breakages
5. Packet loss due to congested nodes/links.

The currently available MANET routing protocols can be classified into two categories [7]: (i) Unipath routing protocols, and (ii) Multipath routing protocols, explained below.

2.1 Unipath routing protocols

In unipath routing protocols, the transmission of messages between a source-destination pair of nodes takes places through a unique path. All the unipath routing protocols may be classified to be either table-based or on-demand [7]. Table-based protocols are characterized by their ability to maintain routing tables that store information about routes from one node in the network to the others. Obviously, this requires that the nodes in the network maintain the table up-to-date by exchanging routing information between the participating nodes. Although, in general, table-based protocols may be easy to implement, the major limitation

associated with these protocols is that due to the highly-mobile and dynamic nature of *ad hoc* networks, maintaining the routing information in these tables is a very challenging task [7].
On-demand routing protocols, on the other hand, alleviate the above problems, and make routing more scalable to highly dynamic and large networks. As the name suggests, on-demand routing protocols are characterized by the computation of routes on an "as-required" basis. In on-demand routing protocols, there is initially a *route discovery* phase in which a route is found between two nodes. The *route discovery* phase is normally followed by a *route maintenance* phase in which a broken link in a route is repaired, or a new route is found [7,9].
Various unipath routing protocols have been proposed in the literature (e.g., [5,9]). Of these, the *Ad Hoc On-Demand Distance Vector* (*AODV*) routing protocol [9], and the *Dynamic Source Routing* (*DSR*) protocol [5] are the most popular ones. In the interest of completeness, we briefly discuss these protocols below, with sufficient details so as to introduce the context for the fault-tolerant routing problem discussed later in this chapter.

2.1.1 The AODV routing protocol

As the name suggests, AODV is classified as a unipath on-demand distance vector routing protocol. It, therefore, functions by using both a *route discovery* phase and a *route maintenance* phase by incorporating multihop routing in the intermediate nodes between the source and destination. In the AODV, every mobile node functions as a specialized router. Routing tables are maintained in the intermediate nodes, with routing information being obtained on an "as-required" basis with no (or little) assumption on the presence of periodic advertisements by the nodes [7,9]. The AODV has been shown to be scalable with the increase in the number of mobile nodes in a MANET. It is characterized by its ability to provide loop-free route information in which broken links are resolved by repairing existing links or introducing new ones. Since there is no assumption on the presence of periodic advertisements by the nodes, there is little requirement on the amount of bandwidth that should be available to the mobile nodes as compared to protocols that require the presence of advertisements. Finally, it is worth mentioning that the AODV works under the assumption that the links are symmetric, and that the communication can be synchronous, implying that both nodes on either side of a link are capable of talking to each other [9].
Perkins and Royer [9] observed that, normally, there are nodes and paths in a network that are not frequently active. Not only do those nodes seldom maintain any routing information, but rather, they also seldom participate in the periodic advertisements of routing information. Furthermore, one should observe that two nodes need to share routing information only when they need to communicate with each other, or whenever one of them is acting as an intermediate node to relay information destined to reach *another* node in the network. Determining the local connectivity between the mobile nodes can be achieved in a number of ways. One of the most common of these is by transmitting local, and not system-wide, so-called "*Hello*" messages. This will assist the routing tables maintained by the nodes in the neighborhood to be updated quickly, and the response time to be optimized for local movements, thereby providing fast responses to establish new routes.
AODV has primarily two phases of operation: (1) the *route discovery* phase, and (2) the *route maintenance* phase [9]. When one node needs to communicate with another node for which there is no routing information in *its* table, the *route discovery* phase is triggered. The source specifies the destination node to which information needs to be transmitted, and floods the

network with a so-called *Route Request* (RREQ) packet. The latter contains the information about the source address, the source sequence number, the broadcast identification number (which is incremented every time the source node starts a new *route discovery* request), the destination address, the destination sequence number, and the hop count. Any of the nodes that receives the request checks to see if it is identified as the destination node by the RREQ packet, or if it can serve as an intermediate node to transmit information to another node in the network. If that is the case, that node generates a unicast *Route Reply Packet* (RREP) that is sent back along the reverse path in which the RREQ packet was originally sent by the source node. Once the source receives the RREP packet, it then knows where and how to transmit the packet. If none of the above cases hold true, i.e., the node that received the packet is neither the destination node, nor can it serve as an intermediate node to the destination node, it broadcasts the RREQ packet again. Obviously, by doing so, multiple copies of a RREQ packet may be received by the nodes in the network, and any such superfluous multiple copies are discarded [7,9].

The *route maintenance* phase is triggered whenever a broken link is detected by any node, and when that node attempts to forward a packet to the next hop. In the *route maintenance* phase, once the next hop is found to be unreachable, the upstream node sends an unsolicited RREP packet possessing a new sequence number that is greater than the previously-known sequence number by unity. It also sends a hop count of "∞" to all the neighboring upstream nodes, which, in turn, replay that information to their active neighbors, until all active source nodes are notified [9].

Once the notification of a broken link is received, the source node could initiate a so-called *discovery process*. The latter is initiated only by that node which determines that there is a need for the identification of a route to the destination node. The source node then makes a decision about whether or not it wants to rebuild an alternative route to the destination node (by virtue of the broken link). If it does, a RREQ packet is sent out with a destination sequence number that is greater than the previously-known sequence number by unity [7,9].

To summarize, the AODV scheme sends broadcast discovery messages only when required, distinguishes between neighborhood detection and general topology maintenance, and selectively disseminates information about changes to local connectivity only to those nodes that might need the topology/connectivity change information [9].

2.1.2 The DSR protocol

Like the AODV, the DSR is a unicast dynamic on-demand routing protocol. It is a source routing protocol, where the source explicitly provides a packet with the complete information of the route to follow, which is subsequently used by the intermediate nodes to forward the packet to the correct destination node [7].

The DSR only routes packets between hosts that want to communicate with one another. Like the AODV, the DSR also has a *route discovery* phase and a *route maintenance* phase. When two nodes need to communicate with each other, the sender node determines a route. This is done based on the information stored in its cache, or based on the results of a route discovery phase, depending on whether or not the information about the destination node is already available to the source node [5].

In all brevity, the transmission of a packet from a source node to a destination node obeys the following mechanism. The DSR requires that the sender determines and stores in the packet's header the *source route*, where the address of each host in the network is explicitly provided until it can reach the intended destination node. The source finds out the complete

route to the destination from a *route cache* that stores the routing information to different nodes in the network. If such an entry is found, the sender uses this route to send the packet. On the other hand, if such an entry is not found, a *route discovery* exercise, similar to the one discussed for the AODV protocol is initiated by the source route. After the next destination is successfully identified, the packet is then sent to the first hop in the identified sequence of nodes by the source. The first hop node first determines whether it is the final destination. If it is, the packet is considered to be delivered. If not, the next hop is scanned from the sequence of identified nodes to the destination, and the packet is forwarded to the next identified hop. The process continues until the packet is considered to be delivered [5].
As in the AODV, a *route maintenance* exercise may be initiated whenever a broken link is detected. This is a scenario that could occur because any of the nodes along a route fails or is powered down. In such a case, an error message is relayed back to the source node with the information associated with the particular link that failed. Each of the intermediate nodes (including the source node) that receives *this* error message deletes all the routes containing that link from its route cache. A *route discovery* phase may then be initiated subsequently to find new routes [5,7].
The DSR is characterized by its ability to quickly adapt itself to routing changes in environments in which there are frequent and rapidly-occurring host movements. One of the important aspects of the DSR is that there is no requirement for periodic route advertisements, as is frequently required in many routing protocols. This reduces the overall overhead on the network bandwidth, especially because most mobile nodes in *ad hoc* networks are operated over battery power, and there are often situations in *such* networks when there are no periodic routing advertisements taking place [5]. The DSR has hence become popular as a suitable protocol for *ad hoc* networks.

2.2 Multipath routing protocols

Multipath routing protocols proposed in the literature (see, for example, [6,8,16]) are of different types, some of which are based on the foundational principles behind the AODV and DSR protocols. However, all multipath routing protocols share a common characteristic, i.e., they discover *multiple* routes between a pair of source-destination nodes. Multipath routing protocols take advantage of the inherent redundancy observed in networks to find multiple routes from one source node to a destination node. This becomes advantageous for *ad hoc* networks because they are characterized to be very dynamic, and unpredictable in nature [7].
In multipath routing, multiple redundant packets are sent along different paths between a pair of source-destination nodes. This redundancy increases the reliability in the transmission of the information [17], implying that there is a much greater chance (than in unipath routing) that at least one of the paths will be able to successfully deliver the packet. This further ensures its success as a fault-tolerant routing algorithm which provides route resilience when there are route failures in the network. However, the disadvantage of multipath routing is that when redundant packets are sent through different routes, they introduce an unnecessary overhead in the network's capacity [7,18]. This is disadvantageous especially when we take into account the fact that energy-efficiency is an important concern in wireless *ad hoc* networks [18], because most mobile nodes in such environments are battery powered, and are, thus, resource constrained.
Some of the multipath routing algorithms are also capable of providing load balancing in the network by carefully selecting a mechanism to split traffic along different routes to avoid

overloading any single route. This is often quite advantageous in wireless network environments because while, sometimes, it might be difficult to guarantee the reservation of a *large* portion of the bandwidth through a single path, it might be possible to reserve *small* portions of the bandwidth over multiple routes through many paths taken together [7].

The multipath routing algorithms, in general, involve three phases: *route discovery*, *route maintenance*, and *traffic allocation*. The overall *route discovery* and *route maintenance* strategies in multipath routing are similar to those in unipath routing, except that in a multipath routing protocol, multiple routes are discovered or maintained between a pair of source-destination nodes [7].

Two important issues arise in multipath routing, which are the number of paths that would be considered to be optimal, and the selection mechanism of the paths. Nelakuditi and Zhang [8] published an interesting paper that addresses these issues, because the performance of a multipath routing scheme is dependent on the number and the quality of the chosen multiple paths. They proposed a hybrid approach that uses the idea of exchanging link state metrics to identify a set of "good" paths. Without delving deeper into their approach, we review below some of the commonly-used approaches for the selection of multiple paths.

The multiple paths discovered in multipath routing may take different forms categorized as being *node disjoint*, *link disjoint*, or *non-disjoint* routes. In node disjoint routes, there are no overlapping nodes or links. In link disjoint routes, there are no overlapping links, while in non-disjoint routes one permits overlapping nodes or links. The advantage of having disjoint routes is that they provide greater fault-tolerance, in the sense that if one of the nodes/links fail, it is quite unlikely *that* the failure will affect any of the other routes. *Route maintenance* in multipath routing is similar to the one done in unipath routing, except that the protocol requires a decision to be made as to when a *route discovery* phase needs to be triggered, i.e., when a broken link is identified. This is because triggering a *route discovery* every time a failure is identified introduces more traffic, and results in a degraded network performance. On the other hand, if one waits for all the disjoint routes between a pair of source-destination nodes to fail before invoking a *route discovery*, it might result in an unreasonable amount of delay [7].

3. Fault-tolerant MANETs

Due to the mobility of the nodes and the associated rapidly-changing topologies, the reliability of the correct transmission of messages is an important concern for MANETs. Hence, we now consider strategies that would guarantee the delivery of packets in adversarial environments, and in the presence of node/link failures.

The well-known MANET routing algorithms listed above (e.g., DSR, multipath routing etc.) are unsuitable as fault-tolerant routing algorithms for MANETs. Since the DSR chooses the shortest path route for packet transmission in adversarial environments, it can be shown that it will achieve a low packet delivery rate. On the other hand, multipath routing algorithms are strong in their fault-tolerance ability, because they send multiple copies of packets through all possible (disjoint) routes between a pair of source-destination nodes. However, the disadvantage with multipath routing algorithms is that they introduce an unnecessary amount of overhead on the network. Without a mechanism that "tolerates" route failures due to malfunctioning nodes (while making routing decisions), the performance of *ad hoc* network protocols will necessarily be poor, and the routing decisions made by those protocols would be erroneous.

Xue and Nahrstedt [10,11] confirmed that devising a fault-tolerant routing algorithm for *ad hoc* networks is inherently hard. This is because the problem itself is NP-complete due to the unavailability of "correct" path information in these environments. In [10], they designed an efficient algorithm, called the *End-to-End Fault Tolerant Routing* (*E2FT*) *Algorithm*, which is capable of significantly lowering the packet overhead, while guaranteeing a certain packet delivery rate. Following the work of Xue and Nahrstedt [10,11], Oommen and Misra [15] proposed a weak-estimation learning based fault-tolerant routing protocol for MANETs. Very recently, Misra et al. [20] also proposed a low overhead ant-swarm inspired routing protocol for MANETs. This chapter is primarily based on the paper published by Oommen and Misra [15], and most of the discussions and results presented here can also be found in [15].

The algorithms that attempt to solve the fault-tolerant routing problem do so by:

1. Either "flooding" the network with multiple redundant packets along different paths between a pair of source-destination nodes (thus, increasing the probability of a successful transfer);
2. Following a dynamic on-demand routing protocol, where the source *explicitly* provides, *a priori*, the transmitted packet with the complete information of the route to be followed, and hence minimizing the number of multiple redundant packets being transmitted; or
3. Seeking a "happy" medium between the latter strategies, namely, by estimating the potential profitability of maintaining selected paths.

The strategy which is presented by Oommen and Misra [15] is a combination of all these three philosophies [15]. The rationale for this strategy can be catalogued as follows:

1. First of all, this strategy opts to retain *certain* multiple redundant paths, and hence follows the basic principles of the multipath families;
2. Secondly, the strategy simultaneously seeks a solution that minimizes the "flooding", and hence pursuing the *dynamic* source-routing philosophy;
3. Finally, the strategy is akin to the one proposed in [10,11], except that it attempts to explicitly consider the nature of the random variables encountered. Observe that since the nodes are *mobile*, these random variables are, by definition, non-stationary. Thus, rather than using traditional maximum likelihood estimates, we argue that it is expedient to utilize *weak* estimates, namely those that converge in *distribution* as opposed to those that converge *with probability one*. We achieve this by invoking novel *weak* estimation methods that are built on the principles of stochastic learning – as explained in [12,13].

To the best of our knowledge, a scheme which collectively uses all these principles is novel to the work of Oommen and Misra [15]. Indeed, more particularly, we are not aware of any reported method which utilizes non-traditional estimates to achieve the ranking of all possible paths. *These are the novel contributions of this chapter*.

4. Problem model

The problem model that was considered by Oommen and Misra [15] is similar to that used by Xue and Nahrstedt [10], with a few differences introduced in order to simulate more realistic MANET scenarios. Their study, however, considers non-stationary environments, as discussed later in this section. We consider a graph G = (V, E) consisting of |V| mobile nodes, and |E| bi-directional links connecting different nodes. If there are n mobile nodes in a path, the length of any path p is denoted by L(p), in which $p = \{v_1, v_2, ..., v_n\}$, where $v_1, v_2, ..., v_n \in V$, and where every pair $(v_i, v_{i+1}) \in E, i \in \{1,2,...,n\text{-}1\}$. The multipath routes between a pair of source-destination nodes is denoted by $\pi = \{p_1, p_2, ..., p_m\}$, where m is the number of paths between any pair of source-destination nodes. In such a model, $L(\pi) = \sum_{i=1}^{m} L(p_i)$ is used to represent the length of the multipath route.

The *packet delivery probability of a path* is represented as $\gamma(p) = \prod_{i=1}^{m} \gamma(v_i)$. If there are m paths in a multipath route between a pair of source-destination nodes, the packet *delivery probability of a multipath route*, $\gamma(\pi)$, determines the probability that when multiple copies of the packets are sent along all the m paths between the source-destination pair, at least one copy is received. Clearly, $\gamma(\pi)$ is calculated as $\gamma(\pi) = 1 - \prod_{i=1}^{m} (1 - \gamma(p_i))$.

The problem that is addressed in the subsequent portions of this chapter consists of determining a mechanism for fault-tolerant routing that would route packets through mobile nodes in the above environment (i.e., in the presence of faulty nodes) by providing a certain packet delivery rate guarantee, and at the same time, by attempting to route "the least" number of duplicate packets through multiple routes between a pair of source-destination nodes. The reader should note that "blind" multipath routing algorithms are capable of achieving a high packet delivery rate guarantee, because they utilize the benefits of network redundancy. However, their disadvantage is that they route duplicate packets through the multipath routes to provide such a high packet delivery guarantee. Therefore, a solution was sought that would provide a certain "optimum" packet delivery rate guarantee, and that would, simultaneously, reduce the "overhead" routing that could burden the network by virtue of the packet duplication mechanisms adapted by the existing "blind" multipath routing algorithms.

Another *objective* of the work was to propose an algorithm that would be efficient in *non-stationary* environments, i.e., environments in which the fault probability of a mobile node increases as it moves away from the center of the network in which it is supposed to operate. In other words, we would enforce the constraint that as a node moves away from the center of the region of operation, the likelihood of it dropping packets also increases. This is an enhancement of the work by Oommen and Misra [15] over the work by Xue and Nahrstedt [10].

In the interest of brevity, our present survey of the E2FT algorithm is necessarily brief. The algorithm involves two major phases: A *route estimation* phase and a *route selection* phase. The *route estimation* phase is used to estimate the packet delivery probability of all the routes at the disposal of the algorithm at any time instant. As opposed to this, the *route selection* phase is used to select those routes that are confirmed to have satisfied a certain optimization constraint, and to drop those routes from further consideration that are estimated to be unnecessary among all the available multipath routes between a pair of source-destination nodes.

In the *route estimation* phase, the number of packets sent depends on the level of accuracy desired as per the estimation process. Note that a superior estimation is achieved by sending a larger number of packets, compensated by a tradeoff of the overall high network overhead. The accuracy of the estimation is achieved progressively through iterations.

The *route selection* algorithm works as follows. At the beginning, since no estimation results are available, all paths between a pair of source-destination nodes are selected to route the packets. By using a suitable estimation criterion, when the associated estimates of the paths are guaranteed to be accurate enough, the paths are reviewed to either be *confirmed* as one of the routes that "wins" the selection process and be permanently used for routing all future requests, or be *dropped* from further routing considerations.

5. Weak estimation-based fault tolerant routing

As mentioned earlier, the *objective* of the weak-estimation based fault tolerant routing solution proposed in [15] was to minimize the overhead by sending the least possible number of redundant packets, while guaranteeing a certain rate for the delivery of packets. We again emphasize in this chapter as well that there is a tradeoff between the rate of delivery of packets and the overhead. It is possible to achieve a very high packet delivery rate if the number of packets sent is not a concern (e.g., by using the multipath routing scheme). On the other hand, it is possible to achieve a very low overhead, if we do not care about the number of packets that are successfully delivered (e.g., by using the DSR scheme). Thus, attempting to increase one will decrease another and *vice versa*. What is challenging is to see how we can achieve a "balance" between the two. In other words, we need an algorithm that will be able to minimize the overhead by guaranteeing a certain level of efficiency of the packet delivery process. To achieve our objective, we propose a stochastic learning-based weak estimation fault-tolerant routing scheme.

5.1 Weak estimation learning

In statistical problems involving random variables, the quality, reliability, and accuracy of the estimation are important considerations. Traditionally, there have been different estimation schemes proposed in the literature, which can broadly be classified as either belonging to the *Maximum Likelihood Estimator* (*MLE*) class of algorithms [3,4], or as belonging to the *Bayesian family* of algorithms [1,3]. Although the above estimation schemes have been proved to be quite efficient, they work under the premise that the underlying distribution in the environment is *stationary*, i.e., the estimated parameter does not vary with time. In this context, the first two authors of this chapter studied this problem [12,13], and proposed a novel estimation scheme for learning in non-stationary environments[1]. They considered the case when the parameter associated with Bernoulli trials, which lead to binomially distributed outcomes of random variables, changed with time.
In the fault-tolerant routing solution presented in [15], we had used this efficient procedure for the estimation of the packet delivery probability through available paths. It is called the *Stochastic Learning Weak Estimator* (*SLWE*) scheme[2] [12,13], and is based on the principles of the stochastic learning paradigm. It uses a learning parameter, λ, which does not influence the mean of the final estimate. On the other hand, the variance of the final distribution, and the speed of convergence decrease with the increase in the value of this learning parameter. We discuss below the weak estimation scheme.

[1] The theory of these estimates is presented here, briefly, and without the fine details of the respective proofs. They are found in [12].

[2] The term "weak" used in the SLWE estimator scheme refers to the weak convergence of the random variable with respect to the first and second moments only.

Let us consider a binomially distributed random variable, X, as follows:

$$X = \begin{cases} 0 & \text{with probability } s_0 \\ 1 & \text{with probability } s_1 \end{cases} \tag{1}$$

$$\text{such that } s_0 + s_1 = 1, \text{ where } S = [s_0, s_1]^T$$

At any time, t, let X assume the value x(t). In order to estimate s_0 and s_1, SLWE keeps track of the running estimate $p_i(t)$ of s_i at time t, where i = 0,1. In such a setting, the value of p_0 is updated using the following *multiplicative* scheme:

$$p_0(t+1) = \begin{cases} \lambda \times p_0(t) & \text{if } x(t)=1 \\ 1 - \lambda \times p_1(t) & \text{if } x(t) = 0 \end{cases} \tag{2}$$

where λ is a constant (0<λ<1), called the learning parameter, and $p_1(t+1) = 1 - p_0(t+1)$

We now present below some of the interesting results [12] concerning the SLWE.

Theorem 1: *Let* X *be a binomially distributed random variable, and* $P(n)$ *be the estimate of* S *at time '* n *'. Then,* $E\left[P(\infty)\right] = S$.

Proof. Based on the updating scheme specified by Eq. (2), the conditional expected value of $p_1(n+1)$ given P can be seen to be:

$$E\left[p_1(n+1) \mid P\right] = \lambda s_2 p_1 + s_1 - \lambda s_1 + \lambda s_1 p_1 \tag{3}$$

$$= (1-\lambda)s_1 + \lambda p_1(s_1 + s_2) \tag{4}$$

$$= (1-\lambda)s_1 + \lambda p_1. \tag{5}$$

Taking expectations a second time, we can write (5) as:

$$E[p_1(n+1)] = (1-\lambda)s_1 + \lambda E[p_1(n)]. \tag{6}$$

As $n \to \infty$, $E[p_1(n)]$ converges to a limit because the coefficient of the linear difference equation is λ, where $0 < \lambda < 1$. Futhermore, if it converges to $E\left[p_1(\infty)\right]$, we can solve for $E\left[p_1(\infty)\right]$ from (6) as:

$$E[p_1(\infty)](1-\lambda) = (1-\lambda)s_1, \tag{7}$$

implying that $E\left[p_1(\infty)\right] = s_1$. Similarly, $E\left[p_2(\infty)\right] = s_2$, and the result follows. ■

The next results which we shall prove indicate that $E\left[P(n+1)\right]$ is related to $E\left[P(n)\right]$ by means of a stochastic matrix. We derive the explicit stochastic dependence, and allude to the resultant properties by virtue of the stochastic nature of the matrix. This leads us to two results, namely that of the mean of the limiting distribution of the vector $P(n)$, and that which concerns its rate of convergence. It turns out that while the former is independent of the learning parameter, λ, the latter is determined *only* by λ. The reader will observe that the results we have derived are asymptotic. In other words, the mean of $P(n)$ is shown to converge exactly to the mean of S. The implications of the "asymptotic" nature of the results will be clarified presently.

Theorem 2: *If the components of $P(n+1)$ are obtained from the components of $P(n)$ as per Eq. (2), $E\left[P(n+1)\right]=\mathbf{M}^T E\left[P(n)\right]$, where $\mathbf{M}$ is a stochastic matrix. Thus, the limiting value of the expectation of $P(.)$ converges to S, and the rate of convergence of P to S is fully determined by λ.*
Proof. Consider Eq. (6). Since $p_1 + p_2 = 1$, we can write:

$$E\left[p_1(n+1) \mid P\right] = (1-\lambda)s_1(p_1+p_2) + \lambda E\left[p_1(n)\right] \tag{8}$$

$$E\left[p_2(n+1) \mid P\right] = (1-\lambda)s_2(p_1+p_2) + \lambda E\left[p_2(n)\right]. \tag{9}$$

Substituting the above equalities, simplifying and taking expectations again leads to the following vectorial form:

$$E\left[P(n+1)\right] = \mathbf{M}^T E\left[P(n)\right], \tag{10}$$

where

$$\mathbf{M} = \begin{bmatrix} (1-\lambda)s_1 + \lambda & (1-\lambda)s_2 \\ (1-\lambda)s_1 & (1-\lambda)s_2 + \lambda \end{bmatrix} = (1-\lambda)\begin{bmatrix} s_1 & s_2 \\ s_1 & s_2 \end{bmatrix} + \lambda\mathbf{I}, \tag{11}$$

is a stochastic matrix. Since, as $n \to \infty$, both $E\left[P(n+1)\right]$ and $E\left[P(n)\right]$ converge to $E\left[P(\infty)\right]$, it follows that:

$$E\left[P(\infty)\right] = \mathbf{M}^T E\left[P(\infty)\right]. \tag{12}$$

Using Eq. (11), we now show that:

$$E\left[P(\infty)\right] = S, \tag{13}$$

as follows:

$$E\left[p_1(\infty)\right] = (1-\lambda)s_1\left\{E\left[p_1(\infty)\right] + E\left[p_2(\infty)\right]\right\} + \lambda E\left[p_1(\infty)\right] \tag{14}$$

$$= (1-\lambda)s_1 + \lambda E\left[p_1(\infty)\right] \tag{15}$$

$$\Rightarrow E\left[p_1(\infty)\right](1-\lambda) = s_1(1-\lambda). \tag{16}$$

which implies that $E\left[p_1(\infty)\right] = s_1$.
An exact parallel argument leads to the result that $E\left[p_2(\infty)\right] = s_2$, whence the first result of the theorem is proved. Observing that $(\mathbf{M} - \lambda\mathbf{I})$ has the common factor $(1-\lambda)$, it follows that the convergence of P to S, which, in general, is determined by the eigenvalues of $\mathbf{M}$, is *fully determined* by λ. Hence the theorem. ■
From the analysis given above, we can derive the explicit expression for the asymptotic variance of the SLWE. We show that a small value of λ leads to fast convergence and a large variance. As opposed to this, a large value of λ implies slow convergence and a small variance.
Theorem 3: *Let X be a binomially distributed random variable governed by the distribution S, and $P(n)$ be the estimate of S at time 'n' obtained by Eq. (2). Then, the algebraic expression for the variance of $P(\infty)$ is fully determined by λ.*
Proof. Using the same notation as above, the square of p_1 at time '$n+1$' is given by:

$$p_1^2(n+1) = \lambda^2 p_1^2 \qquad w.p. s_2 \tag{17}$$

$$= 1 - 2\lambda(1-p_1) + \lambda^2(1-p_1)^2 \qquad w.p. s_1 \tag{18}$$

$$= 1 - 2\lambda + 2\lambda p_1 + \lambda^2(1 - 2p_1 + p_1^2) \tag{19}$$

$$= 1 - 2\lambda + 2\lambda p_1 + \lambda^2 - 2\lambda^2 p_1 + \lambda^2 p_1^2. \tag{20}$$

Using Eq. (20), we can write $E\left[p_1^2(n+1) \mid P(n) = P\right]$ as:

$$E\left[p_1^2(n+1) \mid P(n) = P\right] = \lambda^2 p_1^2 s_2 + (1 - 2\lambda + \lambda^2)s_1 + 2\lambda(1-\lambda)p_1 s_1 + \lambda^2 p_1^2 s_1 \tag{21}$$

$$= \lambda^2 p_1^2 + 2\lambda(1-\lambda)p_1 s_1 + (1-\lambda)^2 s_1. \tag{22}$$

From Eq. (22), we observe that as $n \to \infty$, both $E\left[p_1^2(n)\right]$ and $E\left[p_1^2(n+1)\right]$ converge to $E\left[p_1^2(\infty)\right]$. Thus, by gathering terms involving $E\left[p_1^2(n)\right]$, Eq. (22) can be written as:

$$E\left[p_1^2(\infty)\right] (1-\lambda^2) = 2\lambda(1-\lambda)E\left[p_1(\infty)\right]s_1 + (1-\lambda)^2 s_1, \tag{23}$$

which can also be expressed as:

$$E\left[p_1^2(\infty)\right] (1+\lambda) = 2\lambda E\left[p_1(\infty)\right]s_1 + (1-\lambda)s_1 \tag{24}$$

$$= 2\lambda s_1^2 + (1-\lambda)s_1, \tag{25}$$

where the last equalities hold since $E\left[p_1(\infty)\right] = s_1$. Thus, we have:

$$E\left[p_1^2(\infty)\right] = \frac{2\lambda s_1^2 + (1-\lambda)s_1}{1+\lambda}. \tag{26}$$

We finally compute the variance of $p_1(\infty)$ as below:

$$Var[p_1(\infty)] = E[p_1^2(\infty)] - E[p_1(\infty)]^2 \tag{27}$$

$$= \frac{(1-\lambda)s_1 s_2}{1+\lambda}, \tag{28}$$

and since $s_2 = 1 - s_1$, the theorem is proved. ■

When $\lambda \to 1$, the variance tends to zero, implying mean square convergence. The *maximum* value of the variance is attained when $\lambda = 0$, and the *minimum* value of the variance is achieved when $\lambda = 1$.

Our result seems to be contradictory to our initial goal. When we motivated our problem, we were working with the notion that the environment was non-stationary. However, the results we have derived are asymptotic, and thus, are valid only as $n \to \infty$. While this could prove to be a handicap, realistically, and for all practical purposes, the convergence takes place after a relatively small value of n. As we will see later, in practice, choosing a value of λ in the interval $[0.9, 0.99]$ yields quite good results. Thus, if λ is even as "small" as 0.9, after 50 iterations, the variation from the asymptotic value will be of the order of 10^{-50}, because λ also determines the rate of convergence, and this occurs in a geometric manner [19]. In other words, even if the environment switches its Bernoulli parameter after 50 steps, the SLWE will be able to track this change. Observe too that we do not need to consider the use of a "sliding window".

5.2 The WEFTR algorithm

In [15], Oommen and Misra used the above-mentioned weak-estimation learning scheme to propose a new fault-tolerant routing algorithm, named the *Weak-Estimation-Based Fault Tolerant Routing* (*WEFTR*) *Algorithm*, which is capable of efficiently estimating the probability of the delivery of packets through the paths available at any moment. Like the E2FT algorithm [10], the WEFTR algorithm involves, among other steps, a *route estimation* phase and a *route selection* phase. The *route estimation* phase is used to estimate the packet delivery probability of all the routes at the disposal at any time instant, whereas the *route selection* phase is used to select those routes that are confirmed to have satisfied a certain optimization constraint, and to drop the unnecessary multipath routes between a pair of source-destination nodes.

In the *route estimation* phase, N packets are sent along a path p. The source node estimates the fraction of packets delivered, $\hat{\gamma}(p)$ from the number of packets, N′, received along that path[3].

In our strategy, the estimate of the packet delivery probability is refined with the increase in the number of iterations. At every iteration, a set of packets is transmitted through each of the multipath routes between a pair of source-destination nodes. We can have two possible scenarios for any path: The nodes in a path either forward the packets correctly, or they do not. Consequently, we can use a binomial estimation scheme (based on the above SLWE) as follows:

$$\hat{\gamma}_0(p) = \begin{cases} \lambda \times \hat{\gamma}_0(p) & \text{if the path does not forward the packet correctly} \\ 1 - \lambda \times \hat{\gamma}_1(p) & \text{if the path forwards the packet correctly} \end{cases} \tag{29}$$

where λ is the learning parameter, such that $0<\lambda<1$, and $\hat{\gamma}_1(p) = 1 - \hat{\gamma}_0(p)$.

In our *route selection* algorithm, for a path to be confirmed, the following condition should be satisfied: $\hat{\gamma}_{WE}(p) \geq \gamma^*$, where γ^* is the minimum packet delivery probability required for a path to be confirmed, and $\hat{\gamma}_{WE}(p)$ is the packet delivery probability estimate using the SLWE scheme presented in Eq. (29). Once a path is confirmed, it is considered to be useful for routing future requests, and consequently, no further estimation is carried out on that path.

[3] Traditionally, this is estimated as: $\hat{\gamma}(p) = \frac{N'}{N}$.

The dropping algorithm selects a path, p_{min}, from all the available paths, π, with the minimum packet delivery estimation value, where the latter is examined to see if the following dropping condition is satisfied [10]:

$$\hat{\gamma}_{WE^{1/m}}(\pi') \geq \gamma^{*} \\ \text{where } \hat{\gamma}_{WE^{1/m}}(\pi') = 1 - \prod_{p \in \pi'} (1 - \hat{\gamma}_{WE^{1/m}}(p)), \text{ and } \pi' = \pi - \{p_{min}\} \tag{30}$$

With the above as a background, we present below a high level sketch of the WEFTR algorithm [15].

Algorithm WEFTR

Input

- A graph (network) with a set of nodes, and a set of links connecting the nodes.
- The nodes are mobile, and links connecting them can be reset with the change in the position of the nodes.
- Some of the nodes in the network are faulty with a certain packet delivery rate dependent on the distance of the node from the center of the area of mobility of the mobile nodes, which, for the purpose of this study, is the "simulation area".

Output

- All the incoming packets are delivered from the source node to the destination node, with the intention of maximizing the packet delivery rate, and minimizing the network overhead.

Algorithm

BEGIN

Step 0 (initialization) - Initialize a vector WEFTR_MP that stores all the paths in use, and WEFTR_Nodes that stores all the nodes in the graph, along with the information about their estimated packet delivery probabilities.

At each time unit, do the following:

```
Case 1: If the unit of time is a simulation pause then
        Step 1. Save the estimated packet delivery probability of each node in the vector
                WEFTR_Nodes.
        Step 2. Update the edges and probabilities in the graph to reflect the current
                position of the nodes, and calculate the new paths from the source to the
                destination.
        Step 3. Use the values stored in WEFTR_Nodes in order to calculate the estimated
                (using the SLWE) packet delivery probability of each path.
Case 2: At each unit of time
        Step 1. Try to confirm or drop paths. Paths dropped are removed from the
                WEFTR_MP vector.
        Step 2. Use all the paths in the WEFTR_MP vector to send the packets, and
                calculate the number of packets that are received for each path and the
                total number of non-duplicated packets that are received.
```

END

5.3 Experimental setup

In order to determine how the performance of the proposed algorithm compares with other competing algorithms[4], we simulated an *ad hoc* network with mobile nodes and dynamically changing topologies, and then ran our proposed algorithm along with the other benchmark algorithms (described in the next Section) in the simulated environment. The results, which appeared in [15], are presented here again.

5.3.1 Simulation environment

The simulated environment that we considered consisted of a flat square of length 500 meters. There were 50 nodes in the network, each having a different data delivery probability which decreases as they move away from the center of the square, and increases as they move closer to it. In other words, if we fix a node in the centre of the square, the reliability of data delivery to its peer nodes (and vice versa) decreases as those peer nodes move away from it. This can happen due to the diminishing signal strength between any pair of communicating wireless devices when they move away from each other. Furthermore, to assume that things are done in a systematic manner (i.e., as per the benchmark accepted "standards") we assumed that each node moves randomly, following the *random waypoint model*[5]. If after a random move as per Eq. (31) and (32) below, a node reaches the edge of the square, then the move is canceled and a new random move for this is done until it lands in a valid position. In our simulated *ad hoc* network, we assumed that the maximum speed with which the mobile nodes can travel is 20m/s. Observe that the nodes move at each time unit, but the links between them are only recalculated at a simulation pause[6]. The maximum speed of a node specified above (i.e., 20 m/s) is needed to calculate how much a node can move in a second. This is because the position of a node at the ith second is calculated as:

$$\mathrm{Xpos}(i) = \mathrm{Xpos}(i\text{-}1) + \mathrm{randnum} \tag{31}$$

$$\mathrm{Ypos}(i) = \mathrm{Ypos}(i\text{-}1) + \mathrm{randnum} \tag{32}$$

In the above, Xpos(i-1) and Ypos(i-1) denote the abscissa and ordinate of a node in the previous second (or time instant), and Xpos(i) and Ypos(i) denote the abscissa and ordinate, respectively, of the corresponding node at the current second (or time instant). If the

[4] There are currently quite a few algorithms (and their variants) reported in the literature that claim to solve the present problem. It is clearly impossible to compare any single algorithm with all of them. But we had opted to compare the WEFTR algorithm with individual schemes that represent the various "families" of strategies reported earlier. The rationale for choosing these was that we believed that it represents a reasonably fair comparison against the entire spectrum of philosophies motivating the algorithms. We are currently considering undertaking a more comprehensive comparison (including a testing on "real-life" network topologies).

[5] The details of this model can be found at http://www.netlab.tkk.fi/~esa/java/rwp/rwp-model.shtml

[6] Here, we assumed in [15] that the links between the nodes in the network do not get "torn down" with every movement of the nodes in the network. In other words, we assumed that the links in the network remain connected until a certain time (i.e., the Pause Time). The alternative could have been to re-compute the links in the network with a unit movement of nodes. The former, according to our view, although debatable, is more realistic. Additionally, re-computing the links with every movement of the nodes in the network would lead to a prohibitively large computational overhead.

maximum speed is 20m/s, the randnum shown above is a random number generated between -20 and +20.
The maximum distance that two nodes can have for which they are connected (i.e., that they can deliver packets to each other) is directly dependent on the simulation parameter referred to as the network's "*Sparsity*". The *Sparsity* of the network is an attribute that signifies how the nodes connect with one another, and is a coefficient whose value ranges between 0 and 1 as follows: A value of 1 signifies that no edges (100% sparse coefficient) connect with one another, whereas a value of 0 signifies that the maximum possible number of nodes connect with one another (i.e., a 0% sparse coefficient). The reader should observe that in the simulation there is no fixed number of links in the networks. The links are recalculated at each simulation pause. This is because two nodes are considered to have a link, if they are within a certain distance of each other. Thus, the *Sparsity* directly influences this distance.
Another parameter that was used in the simulations is the so-called *Pause Time*. It signifies how the algorithm is able to accommodate node mobility. This parameter defines the time interval after which the links are recomputed. Each of the simulations was run for 500 seconds. During the simulation period, random Constant Bit Rate (CBR) traffic was generated between a pair of nodes, where this random traffic had a rate of 10 KB/s. Also, during the simulations the SLWE's learning parameter was kept constant, although, as mentioned earlier, the learning parameter does not influence the mean of the final estimate.
We also determine how far a node is from the center of the square, by measuring its Euclidean distance from the center.

5.3.2 Benchmark algorithms

In order to assess how our algorithm performs when compared to the existing algorithms, we had selected three algorithms [15], all of which were executed together with our proposed algorithm in the simulated environment. The three benchmark algorithms that we chose were:
1. DSR Algorithm
2. Multipath Routing Algorithm
3. E2FT Algorithm[7]

Of these three algorithms, the E2FT represented the the state-of-the-art in the area of fault-tolerant routing in MANETs, and so, we reckoned that the performance comparison between our algorithm and the E2FT was crucial. However, since the DSR and the Multipath routing algorithms are currently widely used in deployed MANETs, they were also considered. Also, although the DSR is a simple routing algorithm, it is weak when it concerns routing information in the presence of malfunctioning nodes. On the other hand, multipath routing is, perhaps, a very strong routing algorithm when there are misbehaving nodes. But, as mentioned earlier, the most significant limitation of multipath routing is that it possesses a large network overhead, as it "loads" all the relevant routes between a pair of source-destination nodes with redundant packets so as to ensure that the destination node receives at least one correct copy of the packet sent from the source.

5.3.3 Performance metrics

Two metrics were used in [15] for evaluating the performance of the algorithms invoked in the experiments:

[7] In our study [15], to be fair to the competition, we had considered the optimized version of E2FT that provides an optimization methodology – namely the one that takes the mobility of the nodes into account.

1. *Percentage of packets delivered*: This represents the rate of the successful delivery of packets to the destination, and is calculated as follows: At each second, the packet delivery probability of all the paths in use is calculated. Then, for each packet sent at that time unit, a random number between 0 and 1 is generated. If the number is lower than the packet delivery probability, the packet is considered as having been delivered. Thereafter, after all the iterations, the percentage of delivered packets is calculated as follows:

$$\text{percentage delivered packets} = \frac{\text{total number of delivered packets}}{\text{total number of sent packets}}.$$

2. *Overhead*: This represents the overall number of packets sent. This *Overhead* index is calculated as the product of the total length of all the paths in use, and the number of packets sent per second (time unit).

5.3.4 Experimental results

Several experiments were conducted [15] to assess the performance of WEFTR (the proposed algorithm) with respect to the benchmark algorithms. The results of the following three sets of experiments are presented below (also available in [15]):

- Variation in *Pause Time*
- Variation in *Sparsity*
- Variation in the faultiness of nodes

Variation in Pause Time: As noted earlier, the *Pause Time* is a parameter specific to the simulation, which indicates how much an algorithm is capable of accommodating the mobility of the nodes. The results of the simulation for this scenario are given in Figure 1. From this figure, we notice that with respect to the *Overhead*, while the *blind* multipath routing is the worst, the DSR is the best, and the metric for the E2FT lies somewhere in between the DSR and the multipath curves. This is, of course, understandable. Our proposed algorithm further improves on the performance of the E2FT scheme by decreasing the *Overhead* by 25-50%. For example, when the *Pause Time* is 250 seconds, the *Overhead* for the multipath routing is 19,790, that for E2FT is 8,740, while that for the WEFTR is 7,225. On the other hand, from Figure 2, we observe that the WEFTR achieves an almost similar order of performance when compared to the E2FT. However, by examining Figures 1 and 2 together, one can infer that our proposed algorithm (WEFTR) is capable of *significantly* reducing the *Overhead* of the best fault tolerant routing algorithm (E2FT) currently available, while achieving a performance packet delivery guarantee of at least 80%. Thus, if one considers *both* these issues simultaneously, it is clear that our algorithm always performs much better than both the DSR and the blind multipath routing schemes.

Variation in Sparsity: In the second set of experiments, we intended to study how the algorithms compared with respect to each other with the variation in the *Sparsity* of the nodes in the network. As mentioned earlier, the value of *Sparsity* ranges between 0 and 1, where 0 represents the smallest percentage of *Sparsity*, and 1 represents the largest percentage of *Sparsity*. Since the nodes are mobile, the question of how often they connect with each other depends on how close they can get to one another, and, clearly, this is directly related to the *Sparsity*. The different *Sparsity* values used in our experiments indicate the relative number of edges between the nodes in the network.

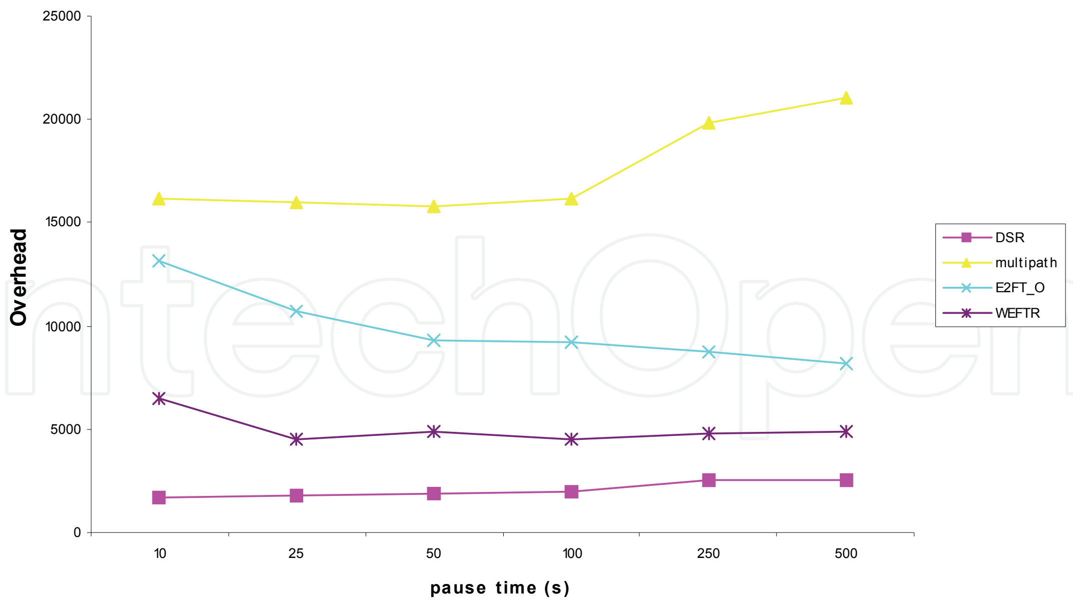

Fig. 1. Plot of the *Overhead* versus the *Pause Time* for the various algorithms tested.

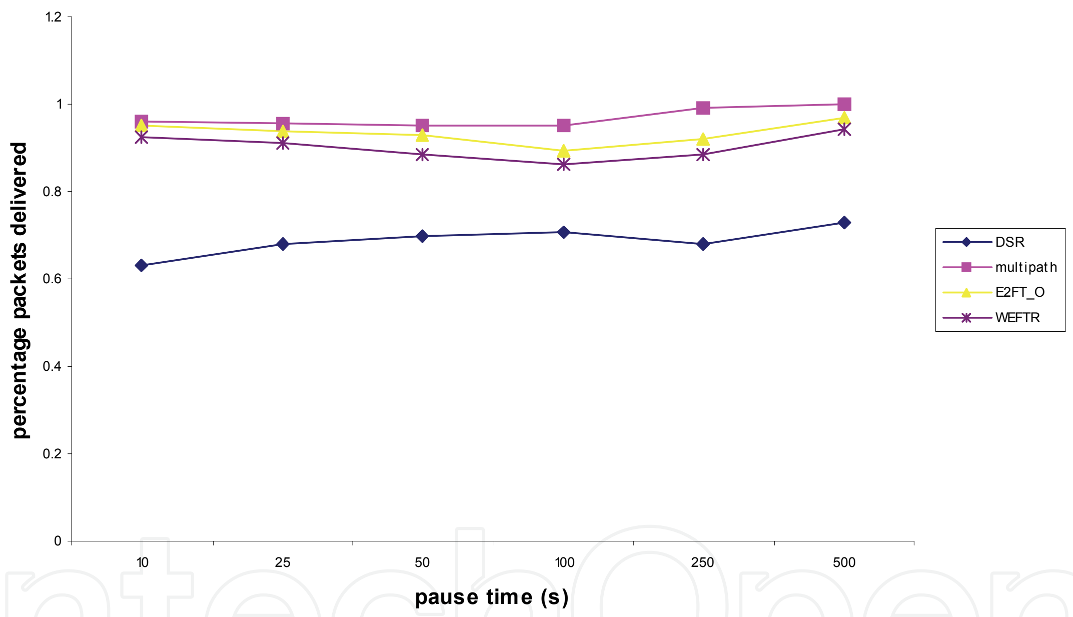

Fig. 2. Plot of percentage delivered packets versus *Pause Time* for the various algorithms tested.

Figures 3 and 4 depict the performance comparison of all the examined algorithms with respect to the overall *Overhead*, and the percentage of packets successfully routed by the algorithms. From Figure 3, we can clearly observe that even at different values of *Sparsity*, the E2FT is capable of significantly reducing the overall *Overhead*. For example, when the value of the *Sparsity* is 0.25, the *Overhead* for the multipath routing is 32,320, while that of the E2FT scheme is 11,410. As opposed to this, the *Overhead* for our proposed algorithm is only 5,570. It should also be observed that the performance of E2FT is much better at lower *Sparsity* values than at the higher ones. On the other hand, if one considers Figure 4, one can observe that, in general, the percentage of packets delivered by both E2FT and WEFTR are almost identical. Thus, for this set of experiments, we observed that the WEFTR significantly reduces the *Overhead* when compared to both the E2FT and blind multipath routing algorithms. This

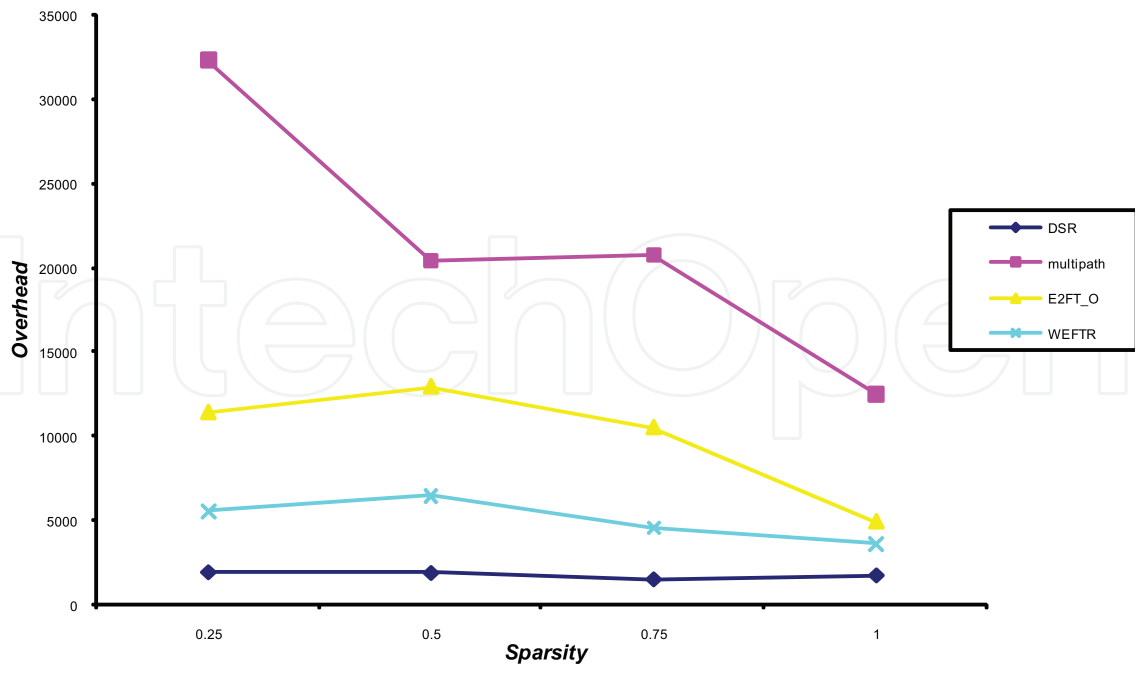

Fig. 3. Plot of the *Overhead* versus the *Sparsity* for the various algorithms tested.

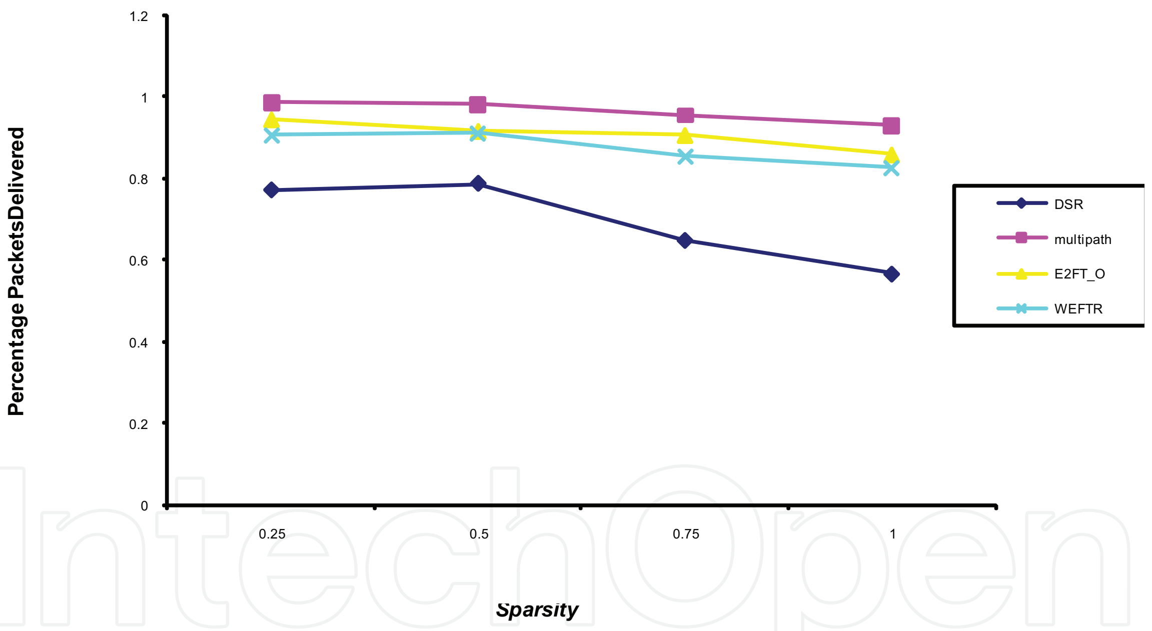

Fig. 4. Plot of the Percentage of Delivered Packets versus the *Sparsity* for the various algorithms tested.

was done while simultaneously achieving a performance comparable to that of the E2FT or the multipath schemes (and certainly yielding a performance noticeably superior to that of the DSR algorithm) with respect to the number packets successfully routed.

Variation in Faultiness: *Faultiness* is an internal simulation parameter that indicates how many nodes will be faulty[8] in a given environment. It influences the faultiness behavior of the nodes, given their distance from the center of the region of operation of the nodes. Figures 5 and 6

[8] In our simulations [15], we assumed that the faulty nodes do not deliver any packets at all.

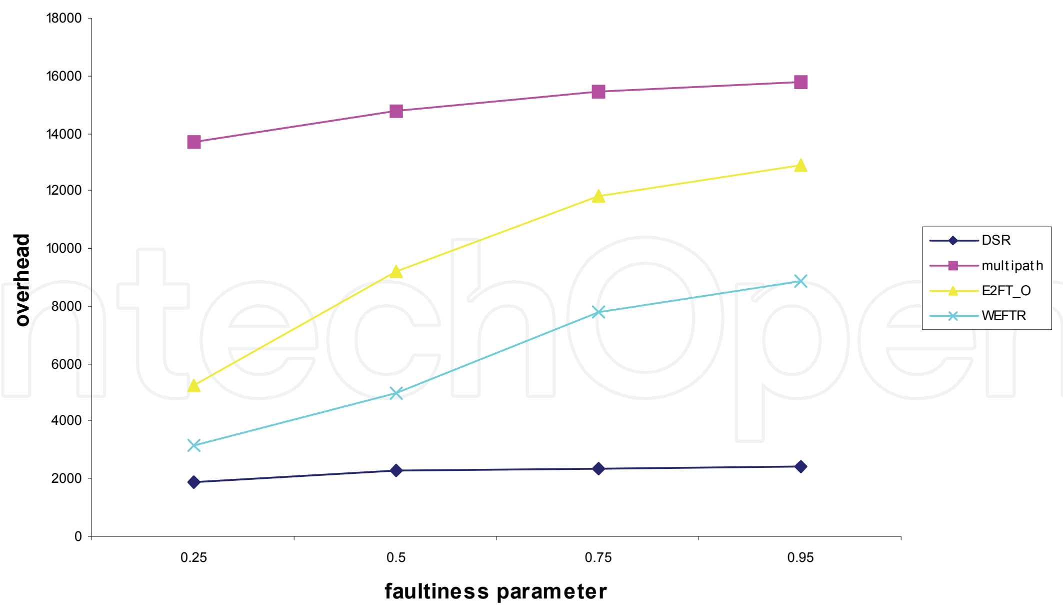

Fig. 5. Plot of the *Overhead* versus the *Faultiness* parameter for the various algorithms tested.

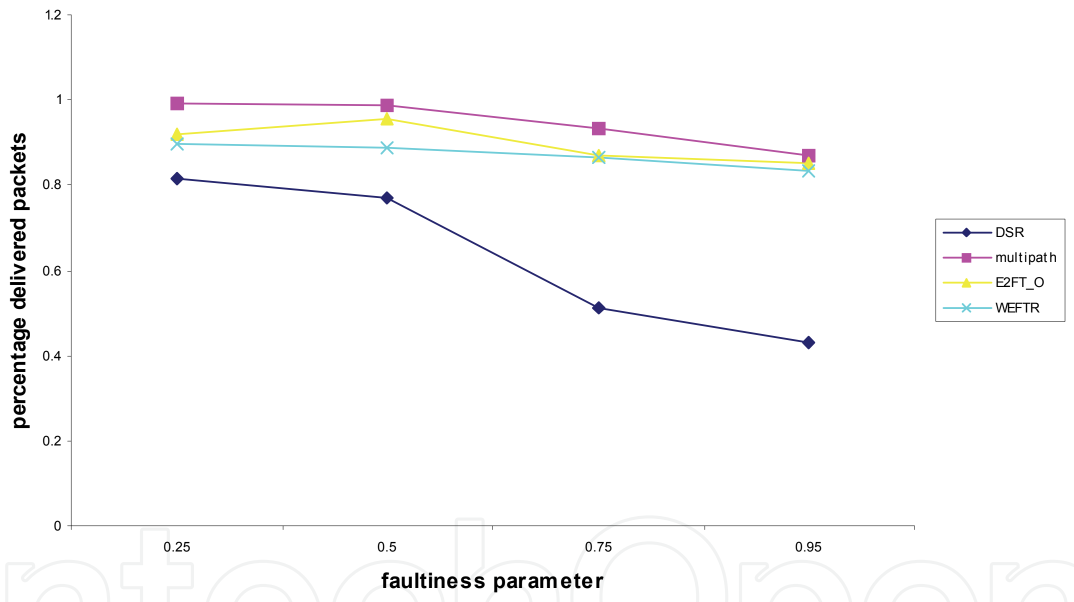

Fig. 6. Plot of percentage of delivered packets versus the *Faultiness* parameter for the various algorithms tested.

depict the variation in the *Overhead*, and the percentage of delivered packets, with the variation in the *Faultiness*. In our experiments, we had used the *Faultiness* parameter to vary from a very low value to a very high value (i.e., on a scale of 0 to 1). We observed that, even in this set of experiments, our proposed algorithm delivers much better performance, when compared to the other algorithms. For example, when the *Faultiness* parameter has a value of 0.25, the *Overhead* for the blind multipath routing is 13,690, for the E2FT is 5,240, while for the WEFTR, it is 3,150. Thus, in this case, our algorithm showed an improvement of about 62 % over multipath routing, and an improvement of about 40 % over the E2FT algorithm. All of these algorithms, however, in general, showed comparable performance with respect to the percentage of successfully delivered packets.

6. Conclusions

We have considered the problem of routing in MANETs, and reported the results of studying the interesting, yet challenging, problem of fault tolerant routing in MANETs, which also appeared in [15]. The problem is that of *efficiently* routing packets in MANETs in adversarial environments particularly, in the presence of misbehaving nodes. Apart from surveying the families of algorithms useful for non-fault tolerant schemes, we have considered state-of-the-art fault tolerant methods, and also devised an algorithm, which is able to successfully route packets by "tolerating" faults in the network. There are two principal metrics that characterize the quality of any fault tolerant routing algorithm designed for MANETs, namely: (1) The *Overhead*, and (2) The percentage of successfully delivered packets. The traditional algorithms, the DSR and the multipath routing, have the potential to attain two extremes of each of these metrics. While the multipath routing is a very strong algorithm for maximizing the number of successfully delivered packets, it introduces an extremely large *Overhead* into the network. On the other hand, the DSR has a low *Overhead*, but, simultaneously, is a very poor fault-tolerant routing algorithm, because it will drop packets if there are problems in the route identified by the algorithm. The E2FT algorithm, proposed by Xue and Nahrstedt [10], is capable of minimizing the *Overhead* when compared to the multipath routing algorithm, while achieving a similar order of performance (slightly inferior, to be more specific) with respect to the number of packets successfully delivered.

Since the nodes are mobile, it turns out that the random variables encountered are non-stationary, implying that estimation methods for stationary variables are inadequate. Consequently, in this chapter, we have also presented a fault-tolerant routing scheme [15] that invokes a stochastic learning-based weak estimation procedure to enhance a *route estimation* phase, which, in turn, is then incorporated in a *route selection* phase. Our algorithm significantly reduces the *Overhead* over the E2FT algorithm, while achieving a comparable performance when it concerns the number of successfully delivered packets. By rigorous simulations, we had shown in [15] that this new algorithm was successful in achieving the above goal.

In the future, we intend to test our proposed scheme on more realistic networks and topologies, and to also consider how *alternate* sequence-based estimates can be utilized advantageously to solve the same problem.

7. Acknowledgements

Most parts of this book chapter appeared in the *Telecommunication Systems* (*Springer*), Vol. 44, Nos. 1-2, June 2010, pp. 159-169. Some parts also appeared in *Pattern Recognition*, Vol. 39, 2006, pp. 328-341 and the Proceedings of *IEEE WiMob'08, pp.* 603-607.

8. References

[1] P. Bickel and K. Doksum, *Mathematical Statistics: Basic Ideas and Selected Topics*, Vol. 1, Prentice Hall, 2nd Edition, 2000.

[2] G. D. Caro, F. Ducatelle and L. M. Gambardella, "AntHocNet: An Ant-Based Hybrid Routing Algorithm for Mobile Ad Hoc Networks", Technical Report No. IDSIA-25-04-2004, Dalle Molle Institute for Artificial Intelligence, Switzerland, August 2004. (Also appeared in the *Proceedings of Parallel Problem Solving from Nature VIII*, LNCS 3242, Springer-Verlag, 2004, pp. 461-470).

[3] R. Duda, P. Hart and D. Stork, *Pattern Classification*, John Wiley and Sons, New York, 2nd Edition, 2000.

[4] R. Herbich, *Learning Kernel Classifiers: Theory and Algorithms*, MIT Press, Cambridge, MA, USA, 2001.

[5] D. B. Johnson and D. A. Maltz, "Dynamic Source Routing in Ad Hoc Wireless Networks", *Mobile Computing*, 1996, pp. 153-181.

[6] M. K. Marina and S. R. Das, "On-Demand Multipath Distance Vector Routing in Ad Hoc Networks", *Proceedings of the 9th International Conference on Network Protocols*, Riverside, California, 2001, pp. 14-23.

[7] S. Mueller R. P. Tsang and D. Ghosal, "Multipath Routing in Mobile Ad Hoc Networks: Issues and Challenges", In *Lecture Notes in Computer Science*, Vol. 2964, Maria Carla Calzarossa and Erol Gelenbe (Eds.), 2004.

[8] S. Nelakuditi and Z. –L. Zhang, "On Selection of Paths for Multipath Routing", *Proceedings of the 9th International Workshop on Quality of Service*, LNCS, Vol. 2092, Springer-Verlag, London, 2001.

[9] C. E. Perkins and E. M. Royer, "Ad-Hoc On-Demand Distance Vector Routing", *Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications*, New Orleans, Louisiana, 1999, pp. 207-218.

[10] Y. Xue and K. Nahrstedt, "Fault Tolerant Routing in Mobile Ad Hoc Networks", *Proceedings of the IEEE Wireless Communications and Networking Conference* (*WCNC*), New Orleans, Louisiania, March 2003, pp. 1174-1179.

[11] Y. Xue and K. Nahrstedt, "Providing Fault-Tolerant Ad Hoc Routing Service in Adversarial Environments", *Wireless Personal Communications*, Vol. 29, 2004, pp. 367-388.

[12] B. J. Oommen and L. Rueda, "Stochastic Learning-Based Weak Estimation of Multinomial Random Variables and Its Applications to Pattern Recognition in Non-stationary Environments", *Pattern Recognition*, Vol. 39, 2006, pp. 328-341.

[13] B. J. Oommen and L. Rueda, "A New Family of Weak Estimators for Training in Non-Stationary Distributions", *Proceedings of the 2004 International Symposium on Structural, Syntactic, and Statistical Pattern Recognition*, Lisbon, Portugal, August 2004, pp. 644-652.

[14] K. Wu and J. Harms, "On-Demand Multipath Routing for Mobile Ad Hoc Networks", *Proceedings of EMPCC*, Vienna, February 2001, pp. 1-7.

[15] B. J. Oommen and S. Misra, "Fault-Tolerant Routing In Adversarial Mobile Ad Hoc Networks: An Efficient Route Estimation Scheme For Non-Stationary Environments", *Telecommunication Systems Journal*, pp. 159-169, 2010.

[16] K. Wu and J. Harms, "On-Demand Multipath Routing for Mobile Ad Hoc Networks", *Proceedings of EMPCC*, Vienna, February 2001.

[17] Z. Ye, S. V. Krishnamurthy and S. K. Tripathi, "A Framework for Reliable Routing in Mobile Ad Hoc Networks", *Proceedings of IEEE INFOCOM*, San Francisco, 2003.

[18] V. Srinivasan, C. –F. Chiasserini, P. S. Nuggehalli and R. R. Rao, "Optimal Rate Allocation for Energy-Efficient Multipath Routing in Wireless Ad Hoc Networks", *IEEE Transactions on Wireless Communications*, Vol. 3, No. 3, 2004.

[19] K. Narendra and M. A. L. Thathachar,. *Learning Automata. An Introduction*. Prentice Hall, 1989.

[20] S. Misra, S. K. Dhurandher, M. S. Obaidat, K. Verma and P. Gupta, "A Low Overhead Fault-Tolerant Routing Algorithm for Mobile Ad-Hoc Networks Based on Ant Swarm Intelligence" *Simulation Modelling Practice and Theory* (*Elsevier*), Vol. 18, No. 5, 2010, pp. 637-649.