



Providing Fault-Tolerant Ad hoc Routing Service in Adversarial Environments

YUAN XUE and KLARA NAHRSTEDT

*Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, Illinois 61801, U.S.A.
E-mail: xue@cs.uiuc.edu, klara@cs.uiuc.edu*

Abstract. Most existing designs of ad hoc networks are based on the assumption of non-adversarial environments, where each node in the network is cooperative and well-behaved. When misbehaving nodes exist in the network, the performance of current routing protocols degrades significantly. Since ad hoc networks, consisting of autonomous nodes, are open and distributed in nature, maintaining a fault-free network environment is extremely difficult and expensive.

In this paper, we propose a new routing service named best-effort fault-tolerant routing (BFTR). The design goal of BFTR is to provide packet routing service with high delivery ratio and low overhead in presence of misbehaving nodes. Instead of judging whether a path is good or bad, i.e., whether it contains any misbehaving node, BFTR evaluates the routing feasibility of a path by its end-to-end performance (e.g. packet delivery ratio and delay). By continuously observing the routing performance, BFTR dynamically routes packets via the most feasible path. BFTR provides an efficient and uniform solution for a broad range of node misbehaviors with very few security assumptions. The BFTR algorithm is evaluated through both analysis and extensive simulations. The results show that BFTR greatly improves the ad hoc routing performance in the presence of misbehaving nodes.

Keywords: ad hoc networks, routing protocol, misbehaving node

1. Introduction

Mobile ad hoc networks are self-organized wireless networks formed by mobile nodes. To provide end-to-end communication throughout the network, peer hosts cooperate with each other to handle network functions, such as packet routing. These networks could be rapidly deployed without the support of fixed infrastructure.

1.1. PROBLEM DESCRIPTION

Most existing designs of ad hoc networks are based on the assumption of non-adversarial environments [1], i.e., each node in the network is cooperative and well behaved. However, in adversarial environment, misbehaving nodes always exist, and may significantly degrade the routing performance. For example, if a misbehaving node participating in the routing operation drops data packets, then a large number of packets will be lost. Simulation result shows that the average packet delivery ratio of dynamic source routing (DSR) [2] degrades by 30%, when 20% nodes are misbehaving. Misbehaving nodes exist for the following reasons; (a) mobile hosts lack adequate physical protection, making them prone to be captured, compromised and to become misbehaving nodes; (b) wireless ad hoc networks have open medium and dynamic topology, making them vulnerable to malicious attacks. Attacked devices may

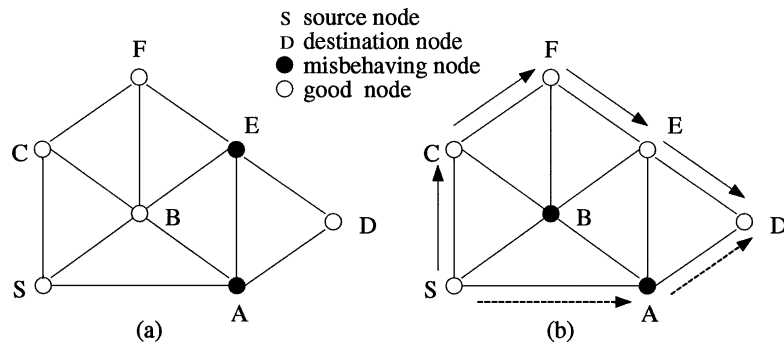


Figure 1. Bad path scenarios: (a) No good path exists between S and D ; (b) The shortest path $S \rightarrow A \rightarrow D$ from S to D is bad; $S \rightarrow C \rightarrow F \rightarrow E \rightarrow D$ is a good path.

behave maliciously; (c) Mobile hosts in ad hoc networks are severely resource constrained computing devices. Overloaded hosts lack the CPU cycles, buffer space or available bandwidth to forward packets. Selfish nodes are unwilling to spend their precious resources cooperatively for operations that do not directly benefit them.

To address this problem, there are two types of traditional design philosophies. The first one is *prevention*. By placing extra protection on the routers, malicious attack can be effectively prevented. Also by setting certain rules to enforce cooperation in routing, one can regulate nodes to be unselfish. This approach works well in the wireline network whose routing is supported by organizational infrastructure. However, in infrastructure-less network such as ad hoc network, routing is achieved by cooperation of autonomous nodes in the form of self-organization. In this case, any effort to mandatorily regulate the behavior of each autonomous node would incur significant overhead. The second philosophy is *detection*. Here, nodes with abnormal behaviors are detected and marked, thus avoided by well-behaved nodes during packet routing. However, without a centralized monitoring point and network administrative authority, it becomes extremely difficult and expensive to effectively detect faulty nodes. In summary, the goal of both designs is to acquire a fault-free network by preventing its nodes from selfishness or malicious attacks, or purging nodes with such behaviors. This goal is too expensive to reach (if not at all impossible) in ad hoc network, whose environment is open and distributed, and consists of autonomous nodes.

1.2. EXPLORING NETWORK REDUNDANCY

Before presenting our design philosophy, we first discuss an intrinsic nature of ad hoc network that can support our idea: network redundancy. Figure 1 shows a simple network containing misbehaving nodes. S is the source, and D is the destination. There are two cases. In Figure 1(a), no good path exists, since all routes from S to D go through misbehaving nodes. No routing protocol can resolve this problem. In Figure 1(b), both good and bad paths exist in the network. In this case, it is possible to deliver packets correctly. However, popular routing protocols such as DSR¹ may still fail to achieve so due to the following reasons: a) It does not discover all the paths between the source and the destination in its route discovery. DSR floods route

¹ DSR is one of the most well-known ad hoc routing protocols. DSR selects a single path, preferably a short one, for packet delivery. For the detailed description of DSR, readers are referred to Section IV.A.

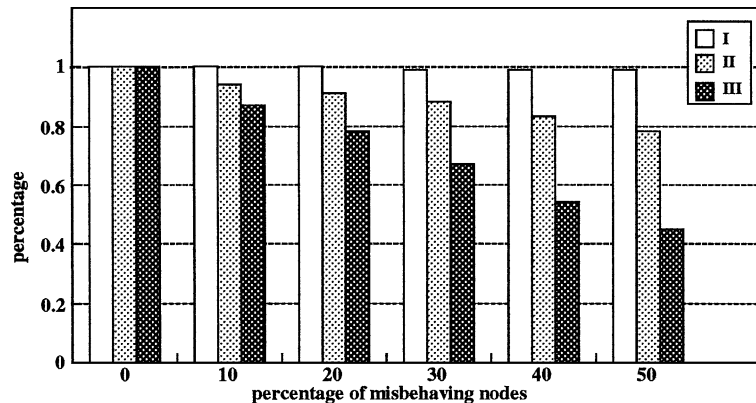


Figure 2. I – the percentage of scenarios where the network has a good path; II – the percentage of scenarios where DSR discovers a good path through flooding; III – the percentage of scenarios where DSR uses a good path.

request packets to discover a path. In the current flooding algorithm, if a node receives the same route request twice, then the second request will be discarded. Although efficiency is achieved through this method, DSR only discovers a subset of paths between the source and the destination, which may miss the good paths; b) Without any clue of which path is good, the route selection of DSR does not consider the packet delivery behavior of paths. Thus, DSR may use a bad path (the one that contains a misbehaving node, such as $S \rightarrow A \rightarrow D$) for packet delivery, even if good paths (such as $S \rightarrow C \rightarrow F \rightarrow E \rightarrow D$) are discovered.

To further study the relationship between these two causes, we simulate DSR on 100 different network topologies. Depending on the cause of packet loss, the network topologies are classified into three categories: a) scenarios where the network has a good path; b) scenarios where DSR discovers a good path through flooding; c) scenarios where DSR uses a good path. The percentage of these scenarios are plotted respectively in Figure 2. It shows that the percentage of scenarios where DSR uses a good path decreases significantly as the percentage of misbehaving nodes increases, which means that reason b) is the critical factor that affects the routing performance. Meanwhile, it also shows that even when misbehaving nodes are densely populated, good paths still exist with high probability, which implies that the routing performance can still be largely maintained.

1.3. DESIGN PHILOSOPHY

Given the above discussions on the infeasibility of traditional design philosophies (prevention and detection) and the possibility of utilizing network redundancy, our design philosophy naturally shifts from excluding or detecting misbehaving nodes to *tolerating* them to certain extent. Our goal also becomes to maximally maintain routing performance (e.g., high packet delivery ratio) in the presence of misbehaving nodes.

In this paper, we present a source-routing algorithm: best-effort fault-tolerant routing (BFTR). The basic idea of BFTR is as follows: although a bad node may exhibit various behaviors, the behavior of any good node is the same: delivering packets correctly with high delivery ratio. Consequently, a good path consisting of purely good nodes also exhibits the same behavior pattern from the end-to-end point of view. Any path which deviates from such a pattern is considered a bad path. In fact, our solution never attempts to conclude whether

the path, or any node along it, is good or bad based on its observed routing behavior. Instead, it utilizes the existing statistics to choose the *most feasible* path, i.e., the one with the highest packet delivery ratio in the immediate past. Likewise, a routing path is discarded when it becomes *infeasible*, i.e., its observed routing performance degrades. The salient features of BFTR are as follows.

1. It requires no security support from the intermediate nodes. Instead, the source node only relies on end-to-end performance observation to judge if a packet is successfully delivered. Thus, the algorithm only requires existences of a priori trust relationship between the communication pair nodes, and secure communication between them, which will be discussed in Section 2.
2. It works against various types of malicious attacks, as it does not differentiate on any of them. Simple attacks such as packet dropping, corruption, or misrouting can be easily detected from receiver's feedback. A malicious node may also exhibit inconsistent behavior, i.e., it first acts as a good node to get trust from other good ones, then starts to attack. This can also be quickly detected, since a sender constantly watches on its current path. On the other hand, those packets that have already been successfully delivered are still counted as goodputs, no matter the path they went through is good or bad. Such a design captures the essence of packet-switch routing: the performance of a routing service is supported by individual success of each packet delivery.

The rest of this paper is organized as follows: Section 2 presents our routing algorithm and its analysis. In Section 3, we discuss the implementation issues of the BFTR protocol. Section 4 studies the performance of BFTR through simulation. Section 5 presents the related work. Section 6 draws the conclusion.

2. Algorithm and Analysis

In this section, we present the BFTR algorithm and its analysis. Given the set of paths between the source and the destination, the algorithm is able to dynamically route packets via the most feasible path, i.e., the one with the highest packet delivery ratio based on end-to-end observation, with high probability.

2.1. SECURITY ASSUMPTIONS

We make the following assumptions on the security support of the network:

1. Both the source and the destination nodes of the connections that are considered in our study are well-behaved nodes. Routing service is not provided for misbehaving nodes.
2. There is a priori trust relationship between the source and the destination so that they can authenticate each other during data communication. In this way, their messages (data and acknowledgement) cannot be forged by anyone in the middle. Such trust relationship could be instantiated by the knowledge of the public key through an off-line a priori public key distribution such as PGP or via an on-line public key infrastructure service [3, 4]. The source and destination nodes can then negotiate a shared secret key, via Diffie-Hellman algorithm [5]. In the following data communication, the two end hosts can authenticate each other using this shared secret key.

Comparing with the existing work [6–8] on secure ad hoc routing protocols in the presence of misbehaving nodes, we make fewer assumptions on the network security support. Requiring no security support from intermediate nodes, our solution is more generic and efficient.

2.2. ALGORITHM OVERVIEW

BFTR is a source routing algorithm. The algorithm uses DSR flooding to retrieve a set of paths between source and destination nodes, whenever necessary. Then it chooses the shortest path from this set to send packets. If within a certain observation window (whose size is the number of packets sent), this path is identified as *feasible*, the algorithm continues to send packets along it. The window keeps shifting during the packet sending to observe the most recent behaviors of the same path. Anytime it is identified as an *infeasible* path, the algorithm discards it and chooses the next shortest path to repeat the same procedure.

We first consider the problem of finding feasible paths. The proposed testing heuristic is motivated by our goal: to maximally maintain routing performance. Therefore, the criterion of choosing a path is its feasibility with this regard, i.e., whether a path is able to transmit packets correctly with high delivery ratio. Any path that follows this pattern can be regarded as a feasible path; a path which deviates from this pattern is an infeasible path. On the basis of this idea, data packets transmission is used as a tool to measure the end-to-end performance. By comparing the measured path performance and *a priori* feasible path model, the testing heuristic determines the feasibility of a path. Our testing heuristic is based on end-to-end performance, thus requiring no additional support from the intermediate nodes, which might be misbehaving. Also, from an end-to-end point of view, our BFTR algorithm can provide a unified solution for many types of misbehavior.

2.3. ROUTING ALGORITHM

We start with feasible path modeling. Given a path π , we define a random variable $Y(\pi)$ as follows:

$$Y(\pi) = \begin{cases} 1 & \text{if } \pi \text{ correctly delivers the observed data} \\ & \text{packet with delay } d(\pi) \leq D \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

For a feasible path π^* , $Y(\pi^*)$ follows Bernoulli distribution. The probability that a packet is delivered successfully ($Pr\{Y(\pi^*) = 1\}$), denoted as $p(\pi^*)$, satisfies $p(\pi^*) \geq p_0$. $p_0 \in [0, 1)$ can be understood as the expected packet delivery ratio, which is a parameter determined from field measurement. The value of p_0 may vary depending on the network load and the wireless channel quality.

Given the model of a feasible path, we now determine whether a path π is feasible. Intuitively, if the packet delivery behavior of π follows the feasible path model, then it is likely to be a feasible path; otherwise, it is more likely to be an infeasible one. Formally, we identify the infeasible path using the method of hypothesis testing. First, we establish the one-tailed test with the null hypothesis H_0 and the alternative hypothesis H_1 as follows:

- \mathbf{H}_0^2 : path π is a feasible path, i.e., $p(\pi) = p_0$.
- \mathbf{H}_1 : path π is an infeasible path, i.e., $p(\pi) < p_0$.

² For a feasible path π^* , $p(\pi^*) \geq p_0$. Yet, in one tailed test “=” is used instead of \geq conventionally.

Then, we conduct n random experiments on path π by sending n packets on it. As a result, random samples $Y_1(\pi), Y_2(\pi), \dots, Y_n(\pi)$ are generated. Let $W(\pi, n) = \sum_{i=1}^n Y_i(\pi)$. We seek to determine whether or not to reject H_0 based on w , the outcome of $W(\pi, n)$. If H_0 is true, then $W(\pi, n)$ is a binomial random variable with distribution:

$$Pr\{W(\pi, n) = w \mid H_0 \text{ is true}\} \tag{2}$$

$$= b(w; n, p_0) \tag{3}$$

$$= \binom{n}{w} \cdot p_0^w \cdot (1 - p_0)^{n-w} \tag{4}$$

Intuitively, if $Pr\{W(\pi, n) \leq w \mid H_0 \text{ is true}\}$ is small, then H_0 is likely to be false. Let α be the threshold value to reject H_0 . α is the probability of rejecting a true hypothesis, namely the significant level of the test. We construct the critical region R , which consists of all w values such that

$$Pr\{W(\pi, n) \leq w \mid H_0 \text{ is true}\} < \alpha, \tag{5}$$

That is,

$$\sum_{i=0}^w b(i; n, p_0) < \alpha \tag{6}$$

If the outcome of $W(\pi, n)$ falls into this region, i.e., $w \in R$, then we reject H_0 in favor of H_1 , which means that path π is regarded as an infeasible path. Moreover, the probability that such rejection is wrong is less than α . That is, we reject H_0 at the significant level α . α takes very small values in hypothesis testing. Typical values of α include 0.05, 0.01, etc. Figure 3 illustrates an example of critical region. In Figure 3, the distribution of $W(\pi, n)$ is plotted. The shadowed region shows that the probability that the outcome of $W(\pi, n)$ is less than or equal to 32 (which is $\sum_{i=0}^{32} b(i; 40, 0.9)$) is less than 0.05 (the value of α). The corresponding values of w construct the critical region $[0, 32]$. Thus, if the outcome of $W(\pi, n) \in [0, 32]$,

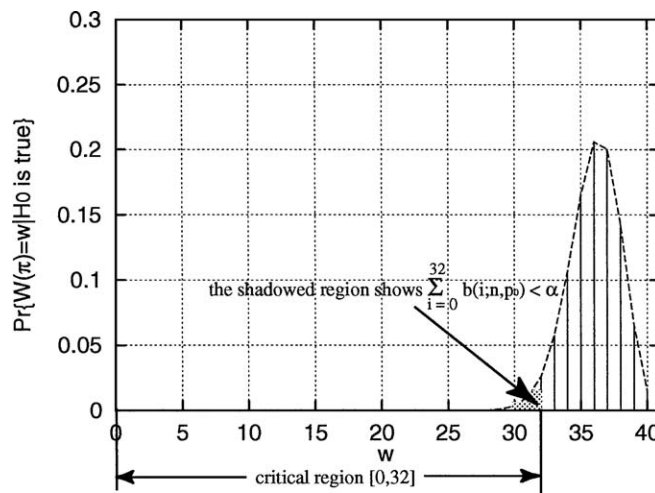


Figure 3. Example critical region. In the figure, $n = 40$, $p_0 = 0.9$ and $\alpha = 0.05$.

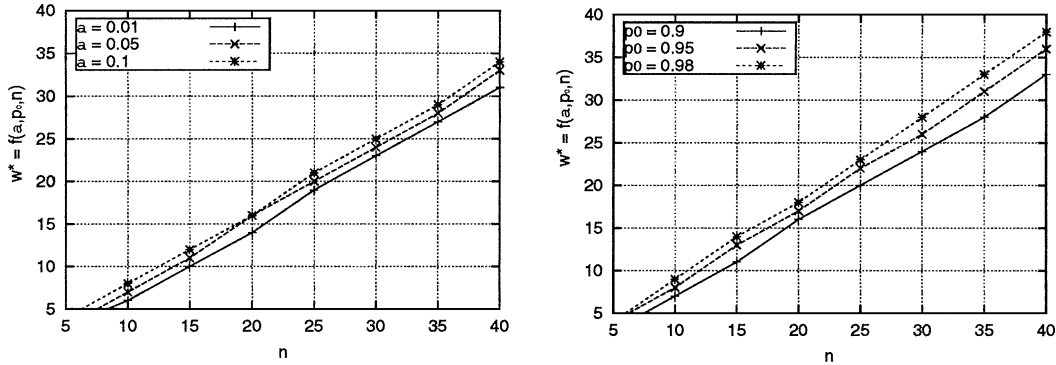


Figure 4. Example computational result of function $w^* = f(\alpha, p_0, n)$. The default parameter values are $n = 40$, $\alpha = 0.05$ and $p_0 = 0.9$.

then the path π is rejected as an infeasible path at significant level 0.05. This means that, the probability that the path π is a feasible path is less than 0.05.

Given the feasible path model with parameter p_0 and the significant level of the test α , we define a function $f(\cdot)$ as follows in order to determine whether the measurement result from n samples are sufficient to reject H_0 :

$$\sum_{i=0}^{w^*} b(i; n, p_0) = \alpha \Leftrightarrow w^* = f(\alpha, p_0, n) \tag{7}$$

For each α and p_0 , $f(\alpha, p_0, n)$ can compute the value of w^* for each n . If w , the outcome of $W(\pi, n)$ based on n random experiments, is less than the computed value of w^* , then the path is rejected as an infeasible one. Otherwise, we keep on sending packets along this path. Figure 4 illustrates an example computational result of $f(\alpha, p_0, n)$ under different α and p_0 . The result shows that a) w^* is a monotonically increasing function of n ; b) increasing either α or p_0 will result in larger w^* .

Note that the above hypothesis testing procedure can only determine whether a path is infeasible, but can never confirm whether a path is feasible. Reflected in our routing algorithm, we keep sending packets along one path until the testing procedure indicates it as an infeasible one. As n (number of packets sent) increases, the probability that an infeasible path is not rejected monotonically decreases. On the other hand, as n gets larger, the procedure becomes more insensitive to the path with inconsistent misbehaviors. For example, a well-behaved node on path π may suddenly start to misbehave after successfully delivering a large number of packets. In this case, the testing procedure will respond slowly as it needs enough packet delivery failures to reject π . To balance this trade-off, we introduce the *observation window*, whose size n_0 is the maximum number of packets to look back. If more than n_0 packets have been sent along π , then the algorithm only uses the most recent n_0 packet deliveries to determine whether to reject π . The optimal value of n_0 under different network settings is evaluated in the simulation. The pseudo code of testing heuristic is given in Table 1. Given a path π , the heuristic tests its feasibility by sending data packets along π . The heuristic terminates when π is rejected. Otherwise, packets are kept sending along π .

Upon retrieving a set of paths via route discovery, the BFTR routing algorithm first chooses to route packets along the shortest path from the path set Ω . Using the test heuristic, BFTR determines whether this path is rejected. If the current path is rejected, it chooses the second shortest path to repeat this procedure. The detailed procedure is shown in Table 2. If

Table 1. Testing heuristic

```

/*  $\pi$  is the path under testing;  $\alpha$  is the significant level;
 $p_0$  is the packet delivery probability threshold value of feasible paths
 $n_0$  is the observation window size */
test ( $\pi, \alpha, p_0, n_0$ ) {
   $n = 0$ ; /* initialize the number of packets sent*/
  while (true) {
    send packets along  $\pi$ ;
    update  $n$  and measure  $w$  in recently sent  $\min(n, n_0)$  packets;
    if ( $w < f(\alpha, p_0, \min(n, n_0))$ )
      reject  $\pi$ ;
  }
}

```

Table 2. BFTR algorithm

```

route() {
  while (true) {
     $\Omega = \text{route\_discovery}()$ ; /* send route request and discover a set of paths */
     $\Omega' = \text{sort}(\Omega)$ ; /* sort all paths according to path length */
    while ( $\Omega' \neq \emptyset$ ) {
       $\pi = \text{dequeue}(\Omega')$ ; /* retrieve the shortest untested path */
      test( $\pi, \alpha, p_0, n_0$ ); /* test and route packet via this path, until it is rejected */
    }
  }
}

```

all paths in Ω have been rejected, a new route discovery procedure is triggered to retrieve a new set of paths, i.e., to renew Ω . The testing procedure then restarts every time Ω is renewed.

2.4. ANALYSIS

In this section, we analyze the performance of BFTR. Whether the feasible path can be correctly identified in the testing phase can greatly affect the overall routing performance. The testing heuristic may make two types of errors in path identification: a) Error A: Rejecting a feasible path; b) Error B: Not rejecting an infeasible path. Error A causes the algorithm to choose a sub-optimal path, since BFTR tests the paths according to the ascending order of their lengths. Error B causes the algorithm to keep sending packets on the infeasible path, resulting in higher packet loss ratio. The following two propositions show that any of these errors either happens with small probability, or incurs controlled performance degradation even when it happens.

Proposition 1. *The upper bound on the probability of Error A is α . That is $Pr\{w < w^* | H_0 \text{ is true}\} < \alpha$, where w is the outcome of $W(\pi, n)$ and $w^* = f(\alpha, p_0, n)$.*

The result can be directly derived from the hypothesis testing condition (6) and the definition of function $f(\cdot)$ (8).

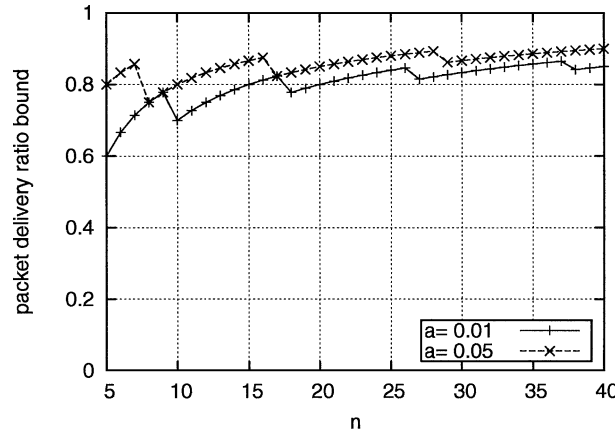


Figure 5. The lower bound of packet delivery ratio. $p_0 = 0.9$ in the figure.

Proposition 1 shows that, by choosing a small value of α , BFTR algorithm is unlikely to discard a good path (with probability $< \alpha$), and to choose a sub-optimal path. The result is independent of the node misbehavior pattern and the number of packets used for testing (n_0).

Proposition 2. *The packet delivery ratio $p(\pi)$ of a path π satisfies the following formula, even if Error B occurs.*

$$p(\pi) > \frac{f(\alpha, p_0, n_0)}{n_0} \tag{8}$$

The result of Proposition 2 is straightforward from the testing heuristic, because any path whose packet delivery ratio is below this bound will be definitely rejected. Proposition 2 shows that even when our algorithm sends packets along an infeasible path, the packet delivery ratio of this path is still guaranteed with certain lower bound. Figure 5 illustrates the relation between the observation window size n_0 and the lower bound of the packet delivery ratio. From the figure, we see that for a fixed value of α , the value of the lower bound increases with the observation window size. (The jitter of the bound shown in the figure is due to the integer rounding operation in calculation.) This shows the trade-off between agility and delivery ratio: large observation window size may guarantee a high packet delivery ratio, yet it is also slow in response to the emergence of misbehaving nodes. Also a small value of α will decrease the lower bound of the packet delivery ratio. This shows the tradeoff between delivery ratio and overhead: a small value of α leads to a low probability of Error A by Proposition 1, and thus low probability of choosing a sub-optimal path. Yet it also may result in lower packet delivery ratio.

3. Protocol

We implement the proposed algorithm in an on-demand source routing protocol, called BFTR protocol. Every data packet in BFTR has a source path in its header, which consists of the node addresses through which the packet should be forwarded to reach the destination. We use an on-demand routing protocol because on-demand routing protocols have been demonstrated to perform better than proactive routing protocols with significantly low overheads under many scenarios. BFTR has three routing components: route discovery, route selection, and route

maintenance. The route discovery and maintenance of BFTR are similar to DSR [2]. Thus we give a brief overview of DSR first.

3.1. OVERVIEW OF DSR

In DSR, each mobile node maintains a route cache in which it keeps the source routes that it has learned. When a node sends a packet, it first checks its route cache for a route to the destination. If a route is found, then packets are transmitted along this path; otherwise, the source node initiates route discovery to find a new route. In route discovery, a route request packet (RREQ) is flooded, specifying the target of that discovery and a unique identifier. In the flooding procedure, each intermediate node that receives RREQ appends its own node address to a list in RREQ and rebroadcasts RREQ, if it has not recently seen this request identifier from the source; otherwise, it discards this RREQ. When the RREQ reaches its destination, the destination sends a route reply packet (RREP) back to the source including a copy of the list of addresses from RREQ either along a known route or the reverse path of RREQ. When RREP reaches the source, it is added as a new route in the route cache.

In route maintenance, if a node detects a link failure, it returns a route error packet (RERROR) to the source, identifying the broken link from itself to the next node. The source node then removes the identified broken link from its route cache. For subsequent packets to this destination, the source may use any other route to that destination in its route cache, or it may attempt a new route discovery after a certain back-off time interval.

3.2. PROPOSED BFTR PROTOCOL

BFTR is similar to DSR in route discovery except that, a) BFTR *requires* the RREP packet to be sent along the reverse path of the RREQ packet; b) BFTR *requires* the destination to send multiple replies, so that the source can have multiple paths between the source and the destination; c) RREP packets are signed with the shared secret key between the source and the destination to prevent them from fabrication and replay attacks.

Given the set of paths acquired through route discovery, the source node makes its local decision in path selection using the proposed BFTR algorithm. The following issues are addressed in the protocol design:

1. *w evaluation.* The source needs to know the number of received packets w along the path under testing. To achieve this, the destination needs to send a feedback to the source to inform it about the value of w . The feedback can be implemented by piggybacking onto TCP acknowledgment (ACK) packets or through additional ACK packets in the network layer. The ACK packets are signed using the shared secret key with sequence numbers to protect them from fabrication and replay attack. In this way, only the authenticated ACK packets from the destination will be accepted, thus ensuring the correct w evaluation.
2. *Packet delay.* In BFTR algorithm, packets are regarded as successfully delivered, if they are received correctly with delay less or equal to D (refer to Formula (1)). Thus, for each packet sent, the source node sets a timer with value $2 \cdot D$. If the timer expires and no ACK is received, then the packet is regarded as lost. In this way, the round trip delay of the packet delivery is implicitly used as an end-to-end performance to determine feasible paths. Actually, large delay in packet delivery does not necessarily mean that the path in use is infeasible. Yet, it at least indicates bad network conditions along the path, such as

congestion or long route. For this reason, BFTR prefers paths with small bounded delay (D). Setting the appropriate value of D is an important issue. Too small D will wrongly reject feasible paths; too large D will accept infeasible paths (e.g., misbehaving nodes delay the packets for some time), resulting in large end-to-end latency. In the current implementation, we use two times the expected round trip time of all the paths as D . Such round trip time information is acquired through route discovery.

3. *Route maintenance.* BFTR maintains routes the same way DSR does. If a route failure report is received, the protocol will discard the current routing path and proceed with the next shortest path in the route cache. Moreover, if all paths in the current route cache have been rejected, BFTR will initiate new route discovery just as what DSR does, attempting to discover more paths (Recall the routing algorithm in Table 2). In a word, BFTR does not distinguish between route failure due to the mobility and test failure caused by misbehaving nodes, as in either case the path is infeasible. Also to control the overhead of possible repeated route discovery, BFTR adopts the same approach as DSR, which spaces out the route request with back-off intervals.

3.3. ADDRESSED NODE MISBEHAVIORS

BFTR can ensure the correct routing operation under various node misbehaviors. Below we list a subset of them and how BFTR performs in the presence of these behaviors.

- *Dropping.* If data packets are dropped by misbehaving nodes, then as being discussed in Section 2, the testing heuristics in BFTR can effectively identify the paths so that the paths in use will give high packet delivery ratio. If either RREQ or RREP packets are dropped, the path on which misbehaving nodes are located will not be discovered. Thus, the routing performance will not be affected, since BFTR is unaware of this bad path. If ACK packets are dropped, BFTR will have a low measurement on the packet delivery ratio of this path ($\frac{w}{n}$), which lowers the probability that we identify this path as a feasible one.
- *Corruption.* If packets are corrupted, the source and destination nodes can detect the corruption through error detection code and discard the packets. Therefore, packet corruption can be treated the same as packet dropping.
- *Mis-routing.* When data packets are routed to a wrong next hop, the node at the next hop will find that it is not on the source route, thus drop the packets. If the node at the next hop is also a misbehaving node and continue to forward the packets, two scenarios may happen from the destination point of view: a) packets do not arrive at the destination within the required delay constraint; b) packets reach the destination and satisfy the delay constraint. In the first case, delayed packets are regarded as lost packets. BFTR can successfully reject this path. As a special example, if several misbehaving nodes collude to forward data packets along a very long path, which results in long delay in packet delivery, then timeout will be triggered at the source. BFTR can identify this type of attack. If the second scenario happens, it means that even the mis-routed path may consist of misbehaving nodes, it can still deliver the packets in a preferred way and will be used in BFTR. Similar situation happens when ACK packets are mis-routed.
- *Tampering.* If the source route is changed, then the tampered path is tested. From an end-to-end point of view, two scenarios may happen: (a) the path in use has small w . In this case, the source will discard this path; (b) the tampered path is able to deliver packets with a high rate. Then this path will be used for packet delivery, even if there may exist bad nodes on this

path. The packet delivery ratio of BFTR will not be greatly affected by mis-selecting such paths. One problem with BFTR is that, if many misbehaving nodes collude to claim they form a shorter path, BFTR can not identify this scenario, and may choose this sub-optimal path. If the routing information in ACK packets is changed, ACK packets will be mis-routed, resulting in the same situation as in *misrouting*. Other changes except route change in ACK packets or RREP packets will not affect the performance of BFTR. Tampering the signed fields of ACK or RREP packets will corrupt the packets, resulting in the same situation as in *corruption*.

- *Delaying*. Delayed packets will be viewed as either packets dropped or packets delivered at the end hosts, depending on how long the packets are delayed. If the packets are delayed for a time larger than D , the source node can correctly identify the bad path; otherwise, the source may mis-identify the bad path as a good one. However, the routing performance will not be greatly affected by such mistakes, since only a small amount of end-to-end latency can be introduced by these mistakes.
- *Fabrication and replaying*. The misbehaving nodes cannot successfully fabricate or replay RREP packets or ACK packets, as they are signed with sequence numbers.
- *Faked Route Failure Report*. Our protocol will abandon the current routing path once a route failure report is received, regardless its truthfulness. The reason is obvious: the path is infeasible for routing in either way. It is either broken, or contains a misbehaving node which refuses to relay packets.

3.4. SUMMARY

In summary, BFTR can maximally maintain the routing performance in the presence of node misbehaviors. It achieves this goal via end-to-end performance measurement provided by the ACK packets. As long as the ACK packets indicate the successful data packets receipt (which is guaranteed by the destination's signature), BFTR can make a preferable route selection decision. As we just elaborated, this approach offers a unified solution for various types of misbehavior.

Note that BFTR is best-effort in that it does not guarantee routing performance. The probability of packet delivery failures will inevitably increase when more misbehaving nodes are present. Such a condition may also happen when the network becomes heavily-loaded. From the view of our protocol, these two cases are indistinguishable. However, both of them indicate that the network is infeasible for routing: no protocol will work well in these conditions.

4. Performance Evaluation

To evaluate the effectiveness and the efficiency of BFTR, we simulate it on a variety of network topologies in ns-2. To compare the performance of BFTR and existing ad hoc routing protocols, we also simulate Johnson et al.'s DSR. DSR has been shown to offer high packet delivery ratios and low routing protocol overhead under network environments, where all nodes are well behaved and cooperative in routing operation.

4.1. SIMULATION SETUP

The simulated network is deployed in a flat square with 700 m on each side. The network has 50 nodes, which include both well-behaved nodes and misbehaving nodes. The number of

Table 3. Default simulation parameter

Simulation time	800 s
Number of nodes	50
Number of misbehaving nodes	10
Number of connections	10
Network size	$700 \times 700 \text{ m}^2$
Sending rate	4 packet/s
Packet size	512 byte
Maximum speed	20 m/s
Pause time	200 s
Wireless random loss rate	10^{-4}
n_0	20 packets
p_0	0.95
α	0.01

misbehaving nodes is a simulation parameter which will be varied in different experiments. By doing so, we show how our protocol could provide robust routing with the presence of arbitrary number of misbehaving nodes. Both well-behaved nodes and misbehaving nodes use the IEEE 802.11 radio and MAC model. Misbehaving nodes forward routing control packets such as RREQ and RREP to intrigue other nodes to use them as relaying nodes, but drop data packets with a configurable probability. The algorithm is simulated under varied dropping probability values to evaluate its performance under different misbehaviors. The default value of dropping probability is 1. The transmission range is approximately a disc with a 250 m radius. The wireless channel has a random loss rate of 10^{-4} .

Each node in the network moves according to the “random waypoint” model [2]. In this model, each node chooses a random destination and moves toward it with a constant speed chosen uniformly between zero and a *maximum speed*. When the node reaches the destination, it chooses a new destination and begins moving toward the destination after a *pause time*. In our simulation, we choose *maximum speed* of 20 m/s. The pause time is a simulation parameter. By varying pause time, we show how our protocol performs under different mobility patterns. Each simulation runs 800 s. During the simulation period, 10 CBR traffic will be generated randomly between nodes with rate 2k byte/second. The default simulation parameters are summarized in Table 3.

4.2. SIMULATION RESULT

First we show the effectiveness of BFTR in improving the packet delivery ratio, when misbehaving nodes exist in the network. Figure 6 plots the packet delivery ratio of BFTR in comparison with DSR under varied number of misbehaving nodes. We have the following observations from this figure: a) the packet delivery ratio of BFTR is consistently higher than DSR in the presence of varied number of misbehaving nodes; b) The packet delivery ratio of BFTR drops more slowly than DSR as the number of misbehaving nodes increases.

The reason behind these two observations can be explained through Figure 7. In this figure, the feasible path usage percentage of BFTR and DSR are plotted respectively. Figure 7 shows that BFTR has a much higher success rate in using the feasible path than DSR. Using a feasible path to deliver packets obviously increases the packet delivery ratio of BFTR.

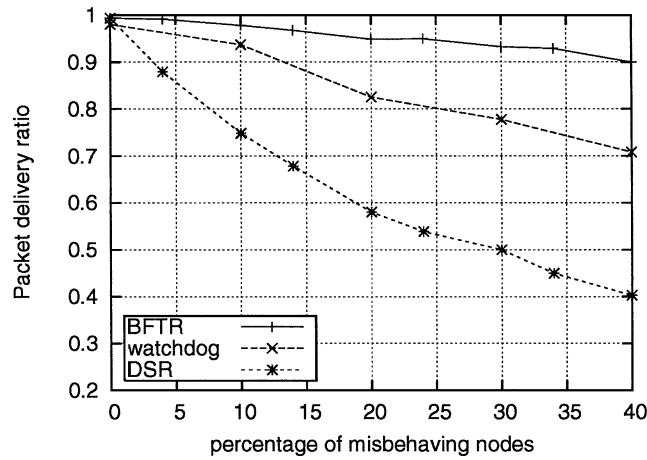


Figure 6. Packet delivery ratio vs. percentage of misbehaving nodes.

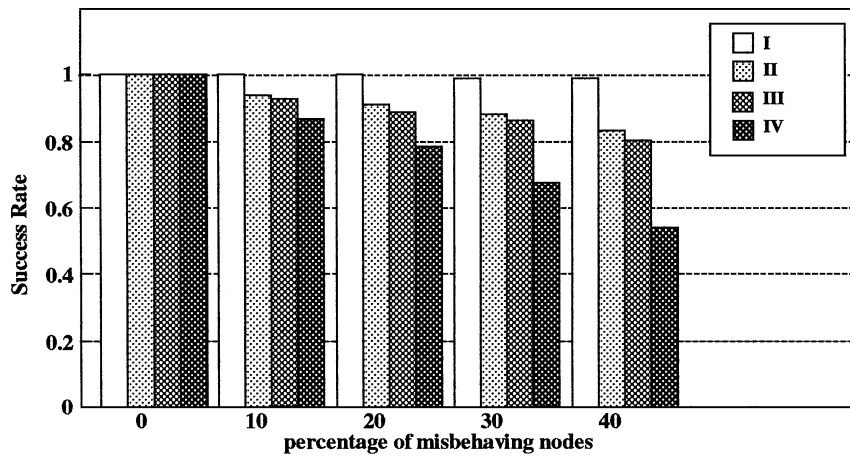


Figure 7. I – the percentage of scenarios where the network has a good path; II – the percentage of scenarios where flooding discovers a feasible path; III – the percentage of scenarios where BFTR uses a feasible path; IV – the percentage of scenarios where DSR uses a feasible path.

Comparing with the existing solution, so called *watchdog*, which mitigates the effect of misbehaving node [9], BFTR can work in environments where malicious nodes collude. To demonstrate the effectiveness of BFTR, we simulate both approaches in an environment where nodes collude with each other. In this scenario, *watchdog* may fail to identify two malicious nodes that are neighbors. The performance of BFTR and *watchdog* is compared in Figure 6. From the results we observe that BFTR outperforms *watchdog* in terms of packet delivery ratio, especially when the number of misbehaving nodes is increased. This is because increasing misbehaving nodes lead to higher probability of successful collusion, which cannot be detected by *watchdog*.

Now we show the efficiency of BFTR. The design of BFTR incurs minimum overhead. The overhead incurred by destination’s feedback can be saved by cooperating with transportation-layer protocol such as TCP. Some additional overhead is discussed in detail here. Figure 8 plots the protocol overhead of BFTR in comparison with DSR. The protocol overhead of BFTR includes the RREQ, RREP, and RERROR packets, which are also the protocol overhead

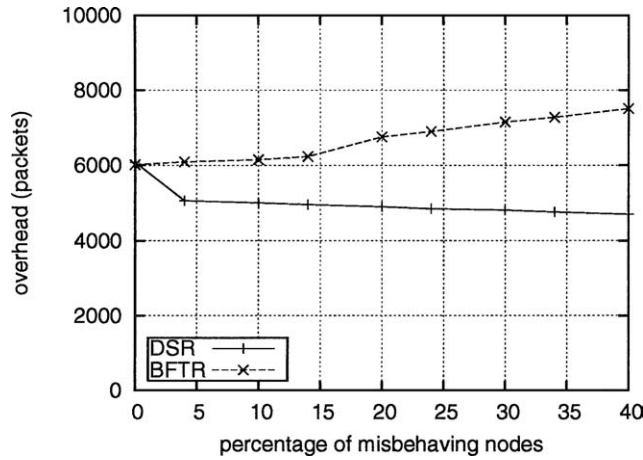


Figure 8. Protocol overhead vs. percentage of misbehaving nodes.

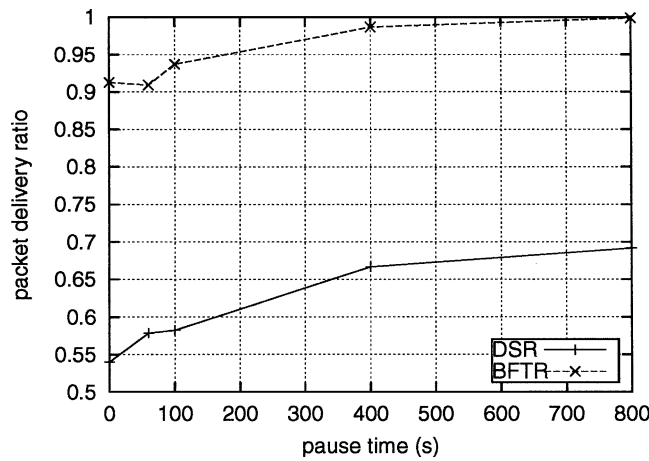


Figure 9. Packet delivery ratio vs. pause time.

of DSR. We see that the overhead increases slightly as the number of misbehaving nodes increases. This is because more route discovery procedures may be triggered by BFTR as a result of absence of feasible paths in the initial rounds of testing. DSR's overhead slightly decreases with larger number of misbehaving nodes. This is because in DSR, a large number of packets can be lost on bad paths. Thus timely RERROR may not be triggered when links break.

Then we vary the average pause time of each node, which determines the frequency of node movement and the changing speed of network topology. Figure 9 compares the packet delivery ratio of BFTR and DSR under different mobility patterns. From the figure, we see that BFTR consistently outperforms DSR under different movement frequencies. In Figure 10, the overhead of BFTR is plotted in comparison with DSR. We see that the overhead of BFTR is comparable to DSR under low mobility environments. When node mobility is high, BFTR can introduce more overhead than DSR. This is because BFTR needs to start its test procedure whenever a new set of paths are discovered. When the test fails in the first round, additional route discovery procedure will be triggered.

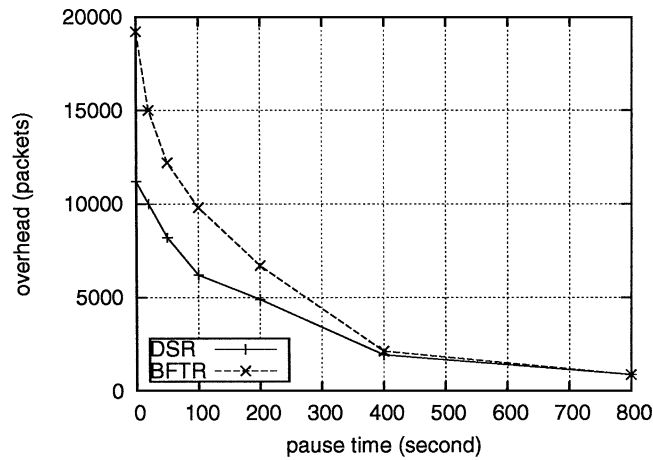


Figure 10. Protocol overhead vs. pause time.

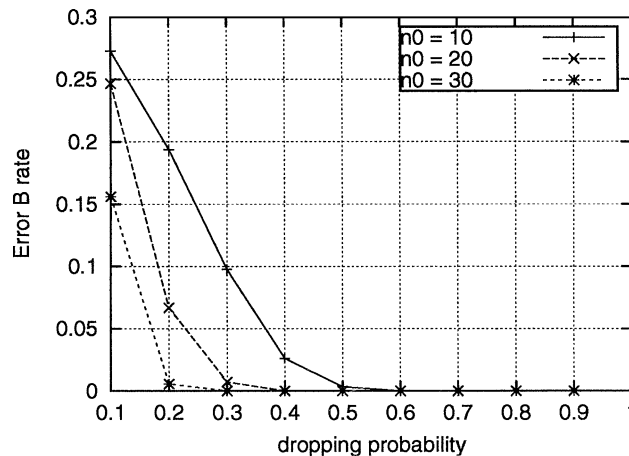


Figure 11. Error B rate vs. dropping probability.

First, we show the impact of observation window size and the pattern of node misbehavior on BFTR performance. We show the probability that BFTR mistakes a bad path as a feasible path under different packet dropping probabilities. We define a metric, Error B rate as $\frac{G'}{G}$, where G is the number of paths which are identified as feasible paths, G' is the number of paths among G which are actually bad paths. From Figure 11, we see that the Error B rate increases with decreasing dropping probability. This observation shows that when bad nodes behave close to good nodes (with small dropping probability), more Error B will be introduced. Also, we see that a large value of n_0 will decrease the occurrence of Error B.

Second, we show that although the dropping probability can affect the Error B rate, it does not greatly affect the packet delivery ratio of BFTR. Figure 12 plots the packet delivery ratios of BFTR using different n_0 . The result shows that the packet delivery ratio of $n_0 = 10$ is slightly lower than the packet delivery ratio of $n_0 = 20$. For $n_0 \geq 20$, further increasing n_0 will not improve the packet delivery ratio.

We further study the effect of p_0 on the performance of BFTR. In the simulation, the misbehaving nodes drops packets with probability 0.6. The packet delivery ratio under

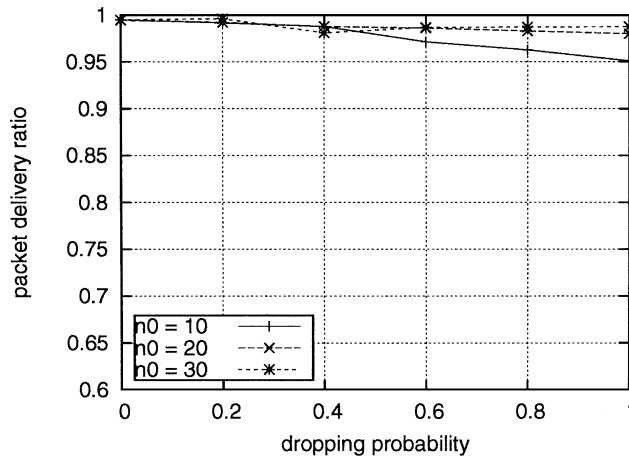


Figure 12. Packet delivery ratio vs. misbehaving node dropping probability.

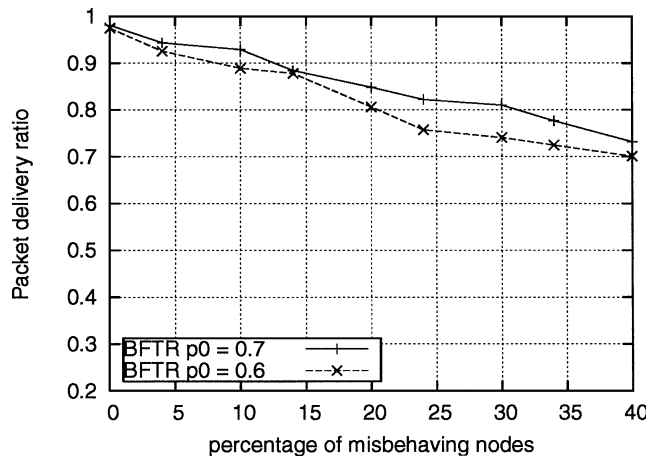


Figure 13. Packet delivery ratio with different values of p_0 .

different values of p_0 is shown in Figure 13. From the results we observe that underestimating the parameter p_0 will lead to imprecise feasible path testing results and affect the performance of BFTR, especially when the misbehaving nodes only drop part of the packets. On the other hand, overestimating p_0 may make the testing algorithm yield no feasible path, result in repeated route discovery and lead to higher overhead. From the simulation, we find out that choosing p_0 from range 0.9–0.95 gives the best performance of BFTR.

Finally, we show the impact of observation window size on the agility of the routing system. In this simulation study, a configurable percentage of misbehaving nodes behave well at the beginning of the simulation. They turn to misbehaving nodes at random selected time instances. Figure 14 shows the relation between packet delivery ratio and the percentage of behavior changing nodes. We have the following observations from this figure. First, BFTR remains a satisfactory packet delivery ratio with different percentage of changing misbehaving nodes. Second, small value of observation window size gives slightly better performance when a large percentage of nodes change their behaviors during simulation. This shows

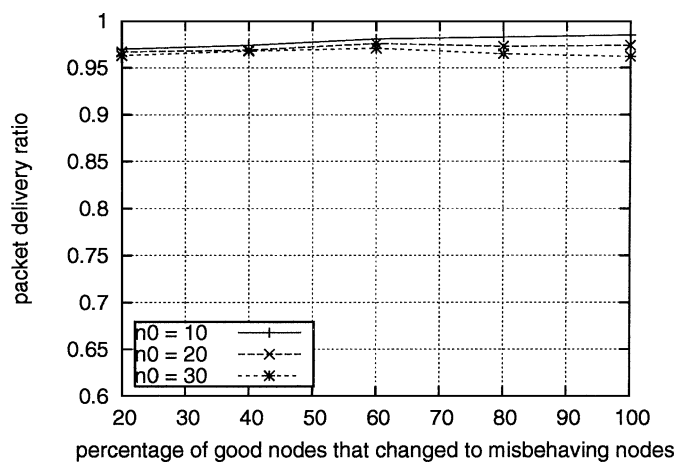


Figure 14. Packet delivery ratio vs. percentage of behavior changing nodes.

the benefit of small observation window size: more agile to the emergence of misbehaving nodes.

5. Related Work

We classify the existing works that address the problem of misbehaving nodes in the ad hoc networks into three categories, namely prevention, detection and toleration. Prevention approach builds a line of protection, trying to prevent misbehaving nodes from participating the routing; detection allows the existence of misbehavior nodes, but detects them and avoids using them in routing; tolerating approach does not even detect misbehaving nodes, instead it seeks to function well in the presence of misbehaving nodes.

Security-aware routing (SAR) by S. Yi et al. [7] employs a prevention approach. It separates nodes into trusted and non-trusted, and forwards packets only through nodes that share a priori trust relationship. Specifically, a trust relationship outside of the context of network is used to associate each node with a trust level. SAR ensures that a node can only process the packet or forward it, if the node itself has required trust level. By this method, it blocks the non-trusted nodes from participating the routing. SAR has the following limitations: a) the assumed priori trust relationship and the fixed trust level assignment may not be suitable for self-organized ad hoc networks; b) the trusted nodes based on a prior relationship can still be misbehaving nodes due to attacks or software/hardware failures. In our work, we only assume the existence of a security association between the source node and the destination node. No assumption is made regarding the intermediate nodes. Instead, we test the behavior of a certain path before we trust it. Such an approach provides more flexible and robust solution for the misbehaving node problem.

Papadimitratos and Haas [6] present a secure route discovery protocol that mitigates the detrimental effects of node misbehaviors, and provides correct connectivity information. The protocol guarantees that fabricated, compromised or replayed route replies would either be rejected or never reach the source node. The only requirement of the proposed scheme is the existence of a security association between the source node and the destination node; no assumption is made regarding the intermediate nodes. One main limitation of this scheme is

that, if misbehaving nodes collude, the protocol will fail. BFTR is similar to this protocol in the security requirements they have. BFTR differs from this protocol in that, a) BFTR does not secure the route discovery procedure, instead, it identifies feasible paths among all discovered paths which may include bad paths; b) BFTR can work well in the presence of node collusion.

Hu and Johnson present a secure on-demand ad hoc network routing protocol, called Ariadne [8]. Ariadne can prevent a large number of denial-of-service attacks. It assumes that each node in the network has: a) a synchronized clock with a maximum bound, b) a shared secret key with each node which it communicates with and c) an authentic TESLA key in the ad hoc network. It makes use of TESLA [10] to secure the route discovery so that the source node has the correct topology information to route packets. Ariadne addresses more node misbehaviors than BFTR does. Yet, it also requires more security support (such as TESLA) than BFTR. To address the routing misbehaviors, BFTR is similar to Ariadne in that they both use packet delivery properties for path selection. BFTR is different from Ariadne in that BFTR sequentially tests the discovered paths; while Ariadne uses “proportional” multipath routing. Thus, BFTR will incur less overhead than Ariadne. Y. Hu et al. also present a secure efficient distance vector (SEAD) routing protocol [11]. The protocol uses one-way hash functions to prevent attackers from creating incorrect routing state. Comparing with SEAD, BFTR incurs less overhead to the network.

As an important component of prevention, authentication among a group of mobile nodes has been extensively studied [3, 4, 12–15]. Prevention approach attempts to authenticate all the good nodes in the network and establish trust relationship among them, thus would incur significant amount of overhead to the networks. In contrast, BFTR only requires the trust relationship between the communicating peers and does not require authentication of intermediate relaying nodes. Thus it is more efficient.

Detection is considered to be a reactive approach to defend ad hoc networks against misbehaving nodes. Zhang and Lee [16] propose an intrusion detection architecture for ad hoc networks. S. Marti et al. [9] present a technique to detect misbehaving nodes in a source routing protocol and help the routing protocol to avoid these nodes.

This scheme may have incorrect detection results for several reasons. Besides the list of scenarios provided by the author (e.g., collisions and node collusion), the most important vulnerability of this scheme is that, without any protection on the detection report, the misbehaving node can generate fake alert and disrupt the routing protocol. The idea of detection report is further extended as reputation system in the work [17, 18]. These schemes may have incorrect detection results for similar reasons. Our approach does not rely on any report from the intermediate nodes, thus does not suffer from false report problems. Moreover, BFTR also functions well in the presence of collisions and node collusion.

Zhou and Haas [4] pointed out that routing protocols can take advantage of the inherent redundancy in ad hoc networks, which provides multiple routes between nodes, and defend against route disruption attacks. If multipath routing is used to explore network redundancy, the network can tolerate misbehaving nodes. Multipath routing protocols and their performances have also been extensively studied [19–22]. One main limitation of this approach is the high overhead introduced by the multipath routing. BFTR explores the network redundancy through sequential path testing. Thus, it does not incur so much overhead as multipath routing does.

There exist works [23–26] that study a particular misbehavior of the ad hoc node—selfishness. In [23, 24], nuglets are served as virtual credit to encourage packet forwarding.

Nuglets need to be implemented in a secure module in each node. The work in [27] presents a cheat-proof system for such credit exchange in ad hoc networks. In comparison, BFTR addresses a wider range of misbehaviors than these solutions.

6. Conclusion

In this paper, we target on the problem of providing routing service in adversarial ad hoc networks, which consist of misbehaving nodes. We propose a new routing protocol: Best-Effort Fault-Tolerant Routing (BFTR). The design philosophy of BFTR is not to detect whether routing path consists of any misbehaving node, but to evaluate its routing feasibility based on its end-to-end performance (e.g., packet delivery ratio and delay). BFTR offers a unified solution for various types of misbehavior. Analytical and experimental results show that BFTR greatly improves the routing performance in the presence of misbehaving nodes.

Acknowledgements

We thank Nitin Vaidya and Pradeep Kyasanur for their comments and helpful suggestions. This research was supported by the ONR MURI NAVY CU 37515-6281 grant, and the NSF EIA 99-72884EQ grant. Any opinions, findings, and conclusions are those of the authors and do not necessarily reflect the views of the above agencies.

References

1. E. Royer and C. Toh, "A Review of Current Routing Protocols for Ad-Hoc Mobile Wireless Networks", *IEEE Personal Communications Magazine* pp. 46–55, 1999.
2. D.B. Johnson and D.A. Maltz, "Dynamic Source Routing in Ad Hoc Wireless Networks", *Mobile Computing* pp. 153–181, 1996.
3. J. Kong, P. Zerfos, H. Luo, S. Lu and L. Zhang, "Providing Robust and Ubiquitous Security Support for Mobile AdHoc Networks" in *International Conference on Network Protocols (ICNP)*, Riverside, CA, November 11–14, 2001.
4. L. Zhou and Z.J. Haas, "Securing Ad Hoc Networks", *IEEE Network Magazine* Vol. 13, 1999.
5. W. Diffie and M. E. Hellman, "New Directions in Cryptography", *IEEE Transactions on Information Theory*, 1976.
6. P. Papadimitratos and Z.J. Haas, "Secure Routing for Mobile Ad Hoc Networks", in *SCS Communication Networks and Distributed Systems Modeling and Simulation Conference (CNDS 2002)*, 2002.
7. S. Yi, P. Naldurg and R. Kravets, "Security-Aware Ad-Hoc Routing for Wireless Networks", in *Technical Report No. UIUCDCS-R-2001-2241, UILU-ENG-2001-1748*, 2001.
8. Y.-C. Hu, A. Perrig and D.B. Johnson, "Ariadne: A Secure on-Demand Routing Protocol for Ad Hoc Networks", in *The Eighth Annual International Conference on Mobile Computing and Networking*, 2002.
9. S. Marti, T. J. Giuli, Kevin Lai and Mary Baker, "Mitigating Routing Misbehavior in Mobile Ad Hoc Networks", in *The Sixth Annual International Conference on Mobile Computing and Networking (MOBICOM)*, Boston, Massachusetts, August 6–11, 2000.
10. A. Perrig, R. Canetti, D. Song and D. Tygar, "Efficient and Secure Source Authentication for Multicast", in *Network and Distributed System Security Symposium (NDSS)*, San Diego, California, February 8–9, 2001.
11. Y.-C. Hu, D.B. Johnson and A. Perrig, "SEAD: Secure Efficient Distance Vector Routing for Mobile Wireless Ad Hoc Networks", in *The 4th IEEE Workshop on Mobile Computing Systems and Applications (WMCSA 2002)*, pp. 3–13, 2002.

12. F. Stajano and R. Anderson, "The Resurrecting Duckling: Security Issues for Ad hoc Wireless Networks", in *The 7th International Workshop on Security Protocols*, 1999.
13. H. Luo, J. Kong, P. Zerfos, S. Lu and L. Zhang, "Selfsecuring Ad Hoc Wireless Networks", in *IEEE Symposium on Computers and Communications (ISCC)*, Taorima, Italy, July 1–4, 2002.
14. J.P. Hubaux, L. Buttyan and S. Capkun, "The Quest for Security in Mobile Ad Hoc Networks", in *Proceedings of the ACM Symposium on Mobile Ad Hoc Networking and Computing (MobiHOC)*, Long Beach, CA, USA, 146–155, October 4–5, 2001.
15. J. Kong, H. Luo, K. Xu, D.L. Gu, M. Gerla and S. Lu, "Adaptive Security for Multi-layer Ad-hoc Networks", in *Wireless Communications and Mobile Computing, Special Issue on Mobile Ad Hoc Networking*, 2002.
16. Y. Zhang and W. Lee, "Intrusion Detection in Wireless Ad-hoc Networks", in *The Sixth Annual International Conference on Mobile Computing and Networking (MOBICOM)*, Boston, Massachusetts, August 6–11, 2000.
17. S. Buchegger and J.Y.L. Boudec, "Performance Analysis of the CONFIDANT Protocol (Cooperation of Nodes - Fairness in Dynamic Ad-hoc NeTworks)", in *Proceedings of MobiHoc*, Lausanne, Switzerland, June 9–11, 2002.
18. P. Michiardi and R. Molva, "Core: A Collaborative Reputation Mechanism to Enforce Node Cooperation in Mobile Ad Hoc Networks", in *Proceedings of Communications and Multimedia Security* pp. 107–121, 2002.
19. S.J. Lee, and M. Gerla, "Split Multipath Routing with Maximally Disjoint Paths in Ad hoc Networks", in *ICC* pp. 3201–3205, 2001.
20. M. Marina, and S.R. Das, "On-Demand Multipath Distance Vector Routing in Ad Hoc Networks", in *Proceedings of the 9th International Conference on Network Protocols*, 2001.
21. R. Leung, J. Liu, E. Poon, A. Chan, and B. Li, "On-Demand Multipath Routing for Mobile Ad Hoc Networks", in *Proceedings of the 26th International Conference on Local Computer Network*, 2001.
22. K. Wu and J. Harms, "Performance Study of a Multipath Routing Method for Wireless Mobile Ad Hoc Networks", in *Proceedings of the 9th International Symposium on Modeling, Analysis and Simulation*, 2001.
23. L. Buttyan, and J.P. Hubaux, "Enforcing Service Availability in Mobile Ad-Hoc WANS", in *Proceedings of IEEE/ACM Workshop on Mobile Ad Hoc Networking and Computing (MobiHOC)*, Boston, Massachusetts, August 6–11, 2002.
24. L. Buttyan, and J.P. Hubaux, "Stimulating Cooperation in Self-Organizing Mobile Ad Hoc Networks", *ACM/Kluwer Mobile Networks and Applications* Vol. 8, No.5 2003.
25. S. Sundaramurthy and E.M. Belding-Royer, "The AD-MIX Protocol for Encouraging Participation in Mobile Ad hoc Networks", in *Proceedings of the International Conference on Network Protocols (ICNP)*, Atlanta, USA, pp. 156–167, November 4–7, 2003.
26. V. Srinivasan, P. Nuggehalli, C. F. Chiasserini, and R. R. Rao, "Cooperation in Wireless Ad Hoc Networks", in *Proceedings of IEEE INFOCOM*, 2003.
27. S. Zhong, Y. Yang, and J. Chen, "Sprite: A Simple, Cheat-proof, Credit-Based System for Mobile Ad Hoc Networks", in *Proceedings of IEEE INFOCOM*, 2003.



Yuan Xue received her B.S. degree in 1998 from Department of Computer Science and Engineering, Harbin Institute of Technology, and her M.S. in 2002 from the Department of Computer Science, University of Illinois at Urbana-Champaign. Currently she is a PhD candidate in the Department of Computer Science at University of Illinois at Urbana-Champaign. Her research interests include wireless network, network-level and application-level quality of service provisioning and mobile computing. Her email address is xue@cs.uiuc.edu.



Klara Nahrstedt (M '94) received her A.B., M.Sc degrees in mathematics from the Humboldt University, Berlin, Germany, and Ph.D in computer science from the University of Pennsylvania. She is an associate professor at the University of Illinois at Urbana-Champaign, Computer Science Department where she does research on Quality of Service(QoS)-aware systems with emphasis on end-to-end resource management, routing and middleware issues for distributed multimedia systems. She is the coauthor of the widely used multimedia book 'Multimedia:Computing, Communications and Applications' published by Prentice Hall, and the recipient of the Early NSF Career Award, the Junior Xerox Award and the IEEE Communication Society Leonard Abraham Award for Research Achievements, and the Ralph and Catherine Fisher Professorship Chair. Since June 2001 she serves as the editor-in-chief of the ACM/Springer Multimedia System Journal. Her email address is: klara@cs.uiuc.edu.