Provably Competitive Adaptive Routing

Baruch Awerbuch*, David Holmer, and Herbert Rubens
Department of Computer Science
Johns Hopkins University
Baltimore, MD
Email: {baruch,dholmer,herb}@cs.jhu.edu
* Supported by NSF grants ANIR-0240551 and CCR-0311795.

Robert Kleinberg†
C.S.A.I.L.
Massachusetts Institute of Technology
Cambridge, MA
Email: rdk@csail.mit.edu
† Supported by a Fannie and John Hertz Foundation Fellowship. Part of this work was done while the author was an intern at Microsoft Research.

***Abstract*— An ad hoc wireless network is an autonomous self-organizing system of mobile nodes connected by wireless links where nodes not in direct range communicate via intermediary nodes. Routing in ad hoc networks is a challenging problem as a result of highly dynamic topology as well as bandwidth and energy constraints. In addition, security is critical in these networks due to the accessibility of the shared wireless medium and the cooperative nature of ad hoc networks. However, none of the existing routing algorithms can withstand a dynamic proactive adversarial attack. The routing protocol presented in this work attempts to provide throughput-competitive route selection against an *adaptive* adversary. A proof of the convergence time of our algorithm is presented as well as preliminary simulation results.**

I. Background

The basic service offered by every node in an ad-hoc network is that of *routing packets* from their source to their ultimate destination. In general, routing protocols are susceptible to a wide variety of attacks. For example, a malicious node may perform a denial of service attack by selectively jamming some areas of the network.

A great deal of work has been done in terms of guaranteeing practical security considerations in existing network protocols. In practice, adversarial attacks observed and documented in ad hoc networks might not be overly sophisticated. The ease of access to the medium has allowed extremely basic attacks to cause a great deal of damage. Consequently, such attacks can be thwarted by simple yet effective methods.

Existing work in the literature considered a number of strong adversary models. For example, [1] considers a random fault pattern; [2] deals with a static fault pattern and [3] deals with an *oblivious* (non-adaptive) pattern.

Our goal is to design routing protocols for networks that are provably tolerant of *arbitrary adaptive* DOS attacks. The adversary that we will consider selectively attacks packets on a given node or link. This adversary benefits from knowledge of the traffic pattern (including packet contents); this includes all current traffic and all past traffic history.

As a result, the algorithms and analysis techniques used in the previous work will not apply. Existing methods that do not ignore sophisticated adaptive attacks either use brute force (flooding) or assume the existence of *some* trusted servers or routers. We do not wish to make such restrictive assumptions. As a result, the task of designing a throughput-competitive routing algorithm is much harder.

It may appear that our adversarial routing model may lead to impractical algorithms in benign (non-adversarial) settings. However, routing algorithms similar to the one studied here were developed and tested in real network environments by British Telecomm and NTT for both wired and wireless networks with superior results [6], [20], [9], [7], [17], [18], [5], [10]. AntNet, a particular such algorithm, was tested in routing for data communication networks [6]. The algorithm performed better than OSPF, distributed Bellman-Ford with various dynamic metrics, and various modifications of shortest path with a dynamic cost metric [4], [17].

Our contribution: We propose a new algorithm for adaptively selecting routing paths in a network with dynamic adversarial edge failures, and we give a rigorous mathematical analysis of this algorithm, proving that its packet loss will match the minimal cumulative loss of any path, up to an additive error which is sublinear in the number of trials. The general framework we propose is appropriate for analyzing routing protocols for networks operating under the extremely strong adversarial model specified above. Such strong models have not been considered in the literature to the best of our knowledge. In fact, adaptive dynamic denial of service attacks are sufficient to break most existing algorithmic work. (In Section III, we briefly explain why DoS attacks are so devastating.)

What distinguishes the present work is our insistence on proving that under *completely arbitrary* adversarial behavior, with essentially no assumptions about the network, the packet loss of our protocol will essentially match the minimal cumulative loss (i.e., sum of losses of individual links) of any path. While it may seem counter-intuitive that such a goal can be achieved, the key is that although we assume an arbitrary *dynamic* adversary, we compare the algorithm against the best *static* path; if the adversary works hard to damage

the algorithm's throughput, it must necessarily inflict a large cumulative loss on *every path* in the network.

At this point, one could debate whether such sophisticated adversaries are ever going to surface in reality, or whether they are only monsters in our imagination. Our counter-argument is that if, as we theorize, ubiquitous wireless networks become the underlying fabric that binds our society together, we cannot afford not to plan against an adversary with arbitrary powers.

The rest of this paper is organized as follows. In Sections II and III, we review some of the existing work in this area and survey some of the challenges which illustrate why our problem is not solved by simpler approaches. In Sections IV we outline the main ideas underlying our algorithm, which is specified precisely in Section V, analyzed mathematically in Section VI, and experimentally tested in Section VII. The algorithm contains some tunable parameters, e.g. a "sampling rate" δ and a "learning rate" β. Adjusting β allows one to smoothly interpolate between the greedy algorithm (β near 0) and algorithms which are less responsive but more robust against an adaptive adversary (β near 1). The mathematical analysis in Section VI indicates that setting β very close to 1 guarantees good performance against an arbitrary adaptive adversarial attack; however in many circumstances (e.g. random edge failures) greedy approaches achieve faster convergence, which leads one to expect superior performance from a smaller value of β in such circumstances. These theoretical considerations are substantiated by the experiments in Section VII, where the algorithm is tested in a variety of network topologies with random edge failures, and smaller values of β indeed achieve faster convergence.

II. Existing Work

a) Algorithmic Work: The only algorithmic results that possibly work under this strong adversarial model are based on a computational learning framework [12], [8]. Near optimal learning algorithms, *with reliable global information* for finding a shortest path in a graph, where at each time a different *known* cost is assigned to each edge, were studied in [12], [8]; these solutions have an exponential computational overhead. Schemes with polynomial computational overhead were recently suggested by [21], [11]. The solutions in [12], [8] and [21], [11] correspond to "link-state" routing, and to the case where the adversary can exercise DOS attacks with a dynamic but *non-adaptive* failure pattern. Byzantine behavior causes such algorithms to collapse. Most recently, "on-demand" routing against an *oblivious* adversary was suggested in [3]. In this model, an adversary cannot cheat and the pattern of cheating and blocking is assumed to be oblivious, i.e., this work does not handle *dynamic* DOS attacks.

b) Reinforcement Learning: The "Swarm Intelligence" paradigm is an approach to routing in distributed networks of cooperative agents, inspired by studying the process by which swarms of ants converge to the optimal route to a food source by progressively reinforcing the successful paths using pheromone secretions. Interest in applications of ant-based routing in mobile ad-hoc networks (MANETs) has risen, and many recent papers have addressed the subject [14], [4]. Gunes et al [15] considers an ant-based approach to routing in MANETs, with a completely reactive algorithm. Marwaha et al. [16] studies a hybrid approach using both AODV and reactive ant-based exploration. Baras et al [4] describes a new algorithm that utilizes the inherent broadcast nature of wireless networks to multicast control and signaling packets (ants). ARAMA [14] uses an analogous approach. Work on the Swarm Intelligence paradigm is described in [13], [6], [20], [9], [7], [17], [18], [5], [10], [14], [4].

III. Research Challenges

c) Past performance is no guarantee of future success: The problem is that we are making decisions in an online environment, where the online algorithm needs to forward packets by selecting routes while having only information regarding past packets, and no information regarding the future conditions. We make no assumptions about the adversary's behavior or the sequence of fault patterns generated. Moreover, it is also assumed that there may be a powerful adversary generating the worst possible input sequence for the online algorithm. A fundamental question regarding online algorithms is how to evaluate their performance. It is rare, and in some cases outright impossible, that one deterministic algorithm *always* outperforms another deterministic algorithm. One classical method has been to assume a model where the future resembles the past (i.e. a *stochastic* model) regarding packet losses, and to compare the performance of different algorithms on the same model. However, in an adversarial setting, there is no reason why the adversary should follow the rules of any particular stochastic model. If anything, a malicious adversary will do exactly the *the opposite* of what our model would predict. (There is no reason to hope that the adversary will not know our model.)

One may be tempted to think that converging to the best fixed route may be easily accomplished by utilizing a "greedy" heuristic: keep track of past error rates on different links, and simply select the path with the smallest overall failure rate, namely the sum of the failure rates of its links. For example, existing work on routing in overlay networks, such as RON [1] runs a greedy strategy for windows of a specific size. The intuition is very strong: one extrapolates the past failure pattern into the future. This may work if the failure pattern is static or if there is a statistical model of failures. However this method will fail quite spectacularly in the case of dynamic adversaries. For example, consider 100 options for choosing a relay point between the sender and receiver, which define 100 different paths. At time i, path i (modulo 100) is under the control of the adversary, and is failing all packets; at all other times the path is perfect. The greedy algorithm will always pick the worst path, even though at any moment in time only 1% of

the paths are faulty.

To see the principal flaw in this greedy strategy, consider the performance of an investor who tries to follow the best-performing stock on the stock market, with the naive assumption that "past performance is a guarantee of future results." In fact, in an adversarial setting (e.g. the stock market) it may very well be the case that past performance correlates *negatively* with future results, and algorithms must not be fooled into this easy trap.

d) Our path selection must withstand competitive analysis: In general, there may not be an ideal fault-free path, yet we can define the best path in terms of accumulated loss on that path. Our goal is to make path selections, such that our overall performance is comparable to that of the "best path."

Our goal is thus to find a robust randomized algorithm that works well on all inputs, in the sense that the expected behavior of our algorithm is comparable to the optimum fixed path on each input. The goal of this paper is to introduce novel routing algorithms for route selection in an adversarial environment that are *provably optimal*, in the sense that the total number of messages lost in our algorithm exhibits a very small additive gap with respect to the *optimum prescient* route assignment. The optimum assignment is determined with complete knowledge of the adversary's actions in the past, present, and future, and with unlimited computational power. We are seeking an algorithm which meets a performance guarantee based on competitive analysis [19]. In other words, we compare the performance of our online algorithm to that of the best static offline selection. (One can think of this "best static offline selection" as the path chosen by an offline algorithm which must select a fixed path and use it during all time steps.) Our results bound the difference between the cost of the best static offline selection and the selections made by our online algorithm; this difference is referred to in the literature as *regret*.

For example, consider a wireless network in which each user has on average 200 neighbors, and in which there is at least *one fixed path* of 7 hops between sender and receiver having an average link fault rate of 0.01 %, i.e. 99.99% of the time the whole path is reliable. The only problem is to select one out of around 200^7 possible paths, while only having information about past experiments over these paths. In this case, one can construct a counter-example in which the greedy algorithm may *never* succeed in delivering a *single* message, i.e. it has a 100% fault rate, in spite of always selecting the best of the 200^7 paths so far! The explanation for this counter-intuitive fact is that any deterministic algorithm can be easily fooled by an adversary, forcing it to pick a path that always fails.

In contrast, the "competitive" randomized algorithm that we are seeking should be able to "zero in" on the reliable path, or at least get comparable performance. The algorithm we are seeking should guarantee, for any adversarial behavior (subject to the adversary's "pledge" to keep some path of length 7 hops being 99.99% reliable on each link), a fault rate of just below $7 \cdot 0.01\% = 0.07\%$ over long sequences.

e) Randomness is not a guarantee of success: Another "quick fix" is to try to select a random path. There is a misconception that selecting one of many node-disjoint paths, or selecting a random path, will somehow guarantee success. (The stock-market equivalent of this belief is asserting that a stock index such as the NASDAQ 100 cannot go down by much.) There is something appealing about random strategies in the sense that they allow the spreading of risk; what is wrong with the above quick fix is that there is no attempt to adapt the probabilities after receiving feedback about different paths. In fact, it is easy to generate counter-examples in which either one of these policies fails. For example, consider a faulty edge leading into a dense subnetwork. If one generates paths at random by assigning equal probability to traversing each edge in the network, one is very likely to select this faulty edge.

f) Multiple paths do not guarantee success: It has been generally recognized that sending packets along multiple paths is more fault-tolerant than sending packets over a single path. However, it is easy to come up with an example involving a collection of edge-disjoint paths such that each specified path has at least one fault, and yet a random path is likely to not have any fault with high probability.

In conclusion, while there exist different heuristics that can alleviate the problems caused by certain adversaries, the challenge is to utilize and combine these techniques in such a way that one can guarantee near-optimal performance even against very powerful adaptive adversaries.

IV. Proposed Approach

A. Solution Outline

In our routing approach, we act as follows: the process of route detection and fault avoidance is carried out by a distributed process of "learning" fault-free paths, in spite of deceptive techniques pursued by adversaries. In the course of the routing process, each node creates and adjusts a probability distribution on that node's set of neighbors. The probability associated with a neighbor is a local estimate of the the relative likelihood of that neighbor forwarding and eventually delivering the packet to the destination. The algorithm is similar to the one in [3]; however its analysis is different.

B. More details of our approach

The choice of the routing path used in the algorithm is best explained by working backwards from the destination towards the source. It is also easiest to imagine a "fictional" distributed algorithm running at the individual nodes, the purpose of which is to send messages to the source; in reality the whole route is determined by the source, and messages travel from the source toward the destination.

Imagine that each node selects a parent edge towards the source. The set of all parent edges forms a tree rooted at the source. The packets are sent on the unique path in this tree from the sender (the root) to the receiver, and are acknowledged by the receiver. (We refer to such acknowledgements as positive ACK's.) Each node on the path between the sender and the receiver sets a timer after forwarding the packet. If a positive ACK is not received before the timer expires, then that node assumes the packet was lost and instead sends back a negative acknowledgement (negative ACK, or simply NACK) to the source. Proper calibration of the timers leads to a single aggregated ACK message traversing each link, reporting on the status of the downstream portion of the path, i.e., we do not have hop-by-hop ACK's resulting in overhead growing linearly in the length of the path. Note that these acknowledgements are separate from acknowledgments used by upper-layer protocols (e.g. TCP) and are used solely for the purposes or our routing algorithm.

Using these acknowledgements, a lost packet decomposes the path into two parts: the part from the source to the last node that succeeded in acknowledging the packet, and the rest of the path that failed to return an acknowledgement. We now adjust the choice of parent edges as follows. The part of the path that succeeds in acknowledging reinforces its confidence in the parent, while the other part of the path reduces its confidence. The parent will be chosen probabilistically based on the confidences acquired. The intuition is that confidence in the parent reflects not only the reliability of the link between child and parent, but also the fact that the parent is "intelligent" enough to pick the right grandparent, and so on.

We wish to stress that these probabilistic confidence measures are being maintained and updated at the source, not at the nodes in the interior of the path. Upon receiving a positive ack from the destination or a negative ack from an intermediate node, the source has the full picture of both portions of the path and emulates the adjustments of probabilities on behalf of the nodes on the path. Thus, counter-intuitively, it will often be the case that nodes beyond the failing edge have never seen some of the packets that were destined for them, and are not aware that their "confidence" in their parent is being degraded.

By using a probability distribution over the parent edges we generate a probability distribution over all source-rooted trees, such that the probability of each tree grows exponentially as a function of its performance. This discriminates against edges which are frequently controlled by the adversary, and reinforces edges where the adversary is very often absent.

Now, we proceed step by step in explaining the different phases taken by our algorithm to assign probabilities to each edge of the network graph.

g) Transforming a graph into a layered directed graph: The first step is to transform the original undirected network graph $G(V, E)$ (e.g., see Fig. 1,2) into a directed layered graph, with the destination being in layer 0 of this graph while the source is at the other "end" of the graph. Note that the same process is repeated for each destination.

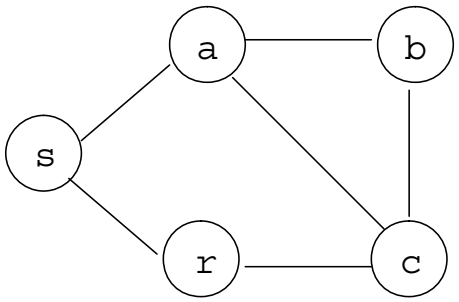

Fig. 1. The original network w.r.t. receiver r (in layer 0)

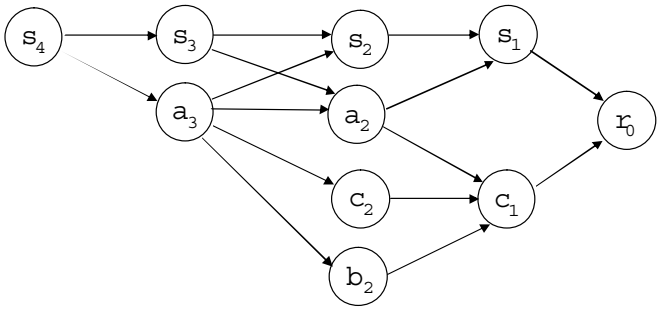

Fig. 2. The Layered graph w.r.t. receiver r (in layer 0)

Informally, all we are doing here is inserting into each packet a TTL (time-to-live) counter, and using a separate routing table for each destination and each TTL counter value. We will say that the packet arrived to a node in layer i if it has TTL value equal to i hops. Thus, all the nodes that can *potentially* reach the destination in i hops (or less), for $0 \leq i \leq H$ are *represented* in layer i of the graph (e.g., that's why node b is not on layer 1 in Fig. 1). Here H is the upper bound on hop count of a routing path (i.e. the original TTL value). The directed edges connect the representative nodes in layer i to representative nodes in layer $i-1$. Suppose that a packet starts at source s, and while traversing the network carries a hop count. When the packet arrives at node v after traversing i hops, we consider the packet as if it arrives at "virtual" node v_{H-i}.

The reader can easily see that a directed leveled graph G' of depth H simulates any communication where packets traverse a bounded number of hops $H \leq |V|$. Thus, for the rest of this description, without loss of generality, we consider our network graph to be leveled, directed, acyclic, and we consider every node to be reachable from the source.

h) Establishing a feedback mechanism : As we already said, we request that each packet be acknowledged using a secure acknowledgement scheme by the destination. The purpose of these ackowledgements is to identify nodes that

drop packets. The adversary can definitely interfere with either the packet or the ACK propagation process. However, any adversarial action can be summarized as follows: the source has received a negative acknowledgement from some intermediate node, indicating the downstream edge on the path (we call it the *separator*) that appears to have failed. All the edges on the path from sender to receiver prior to the separator are called *lucky*. All the edges on the path from sender to receiver beyond the separator edge are called *unlucky*.

i) Quantitatively ranking the incoming edges: Once the source receives a NACK from a node, it views this as a positive ack from all edges prior to the node sending the NACK, and as a NACK from all the edges beyond this node. As in [3], the source ranks the edges based on the percentage of "luck" they bring. Luck is the ratio of positive ACK's from the edge, and the total number of ACK's (positive and negative) associated with this edge. Each node will be selecting the best edge in terms of "luck" on its path towards the sender. Let us call such edge a "parent edge". The set of parent edges forms a routing tree rooted at the source. In reality, routing from the source to the destination is performed over the tree from its root (source) towards the destination, from parents to their children on the path.

The next question is how to set edge weights as a function of each edge's "luck". The greedy strategy (which does not work) would be to set edge weights in decreasing order of luckiness. Instead, it turns out that setting edge weights based on the *exponent of luckiness* — i.e. assigning weight β^x, where β is a parameter between 0 and 1, and x is the number of times that an edge has been unlucky — will essentially yield optimal performance. The value of β is crucial for the performance of our algorithm. Low values of β result in our decision sequence resembling that of the greedy algorithm, and thus will be vulnerable to adversarial attacks. On the other hand, values of β which are close to 1 will cause the algorithm to respond slowly to changes. The optimal value of β can be determined using the "best expert" framework from machine learning (see [12], [8]). The mathematical analysis in Section VI indicates that the optimal value of β for withstanding arbitrary adversarial attacks is essentially $1-\sqrt[3]{m/HT}$, where m is the number of edges in the network, H is an upper bound on the path length, and T is the length of time that the protocol has been running.

j) Choosing a random path from the source: When the source wishes to select a path to the destination along which to route packets, it emulates the following process in the layered graph: the destination at layer 0 selects a parent node in layer 1 randomly according to its probability distribution on incoming edges. This node, in turn, uses its probability distribution on incoming edges to sample a parent node in layer 2, and so on up the layers of the graph until the source is reached. The path from source to destination is chosen to be the reverse of this path. An equivalent (but less computationally efficient) way of sampling this random path would be to have *every* node in the layered graph sample a random parent using its probability distribution, resulting in a tree of edges directed outward from the source; the source then designates a path to the destination which is the unique such path contained in this directed tree.

k) Sampling unexplored territory: A subtle but important component of the algorithm is its ability to sample edges in the network, since this is the only way it is able to detect changes in the adversarial fault pattern, and to sample relatively "stale" portions of the network. This means that the algorithm is occasionally (and probabilistically) deviating from the optimal routes in order to make sure that feedback is obtained from each edge in the network.

This deviation is accomplished by forcing the algorithm to pass, with a small probability which is completely under our control, over a completely random edge $e = (u, v)$ in the layered graph, using a different path-sampling process. The portion of the path from the source to e is computed as above, i.e. we start at u and work our way backward to the source, level by level, using the designated probability distribution at each node along the way to sample the next edge. The portion of the path from e to the destination may be taken to be an arbitrary path, e.g. a fixed route determined at the outset.

Let us give some intuition on why this is desirable. By allowing β to be small the algorithm will move quickly to the least fault path, but will only utilize even slightly less reliable paths with low probability. In order to ensure that the algorithm is making correct decisions, it continuously needs to gain up-to-date feedback on the other paths in the network. This is accomplished by random sampling. If every data packet were used as a sampling packet, meaning the sampling rate was 100%, the algorithm would have extremely high loss rates from exploring faulty paths, but would have extremely accurate estimates of the current reliability of paths in the network. On the other hand, if the algorithm only samples paths with 1% of its traffic, then it will be slightly slower in detecting changes, but will be sending a higher percentage of its packets over links that do not fail.

V. Specification of the algorithm

Our algorithm is a variant of the algorithm in [3] for adapting to a reliable network path. We will use the following notations. We are given a directed graph with specified sender s and receiver r. Time steps run from 1 to T and are denoted by t. Cost functions $C_t : E \to \{0, 1\}$ are specified by an adaptive adversary. The interpretation of C_t is that 1 represents an edge failure, 0 represents an edge which does not fail. In each time step the algorithm chooses a path π_t from s to r and is charged a cost of 1 if any edge of this path failed, 0 otherwise. The algorithm receives feedback regarding the location of the first edge failure (i.e. the one nearest to s) if there were any edge failures.

As explained above, we first transform the network graph into a leveled directed graph $G = (V, E)$ with sender s and

receiver r, and with H levels altogether, as described in section IV-B. The algorithm runs in phases of length τ; later we will specify τ so as to optimize the regret. (The optimal value of τ will be roughly $\sqrt[3]{(m/H)^2T}$.) A typical phase will be denoted by ϕ.

For each vertex v, the algorithm maintains a black box $\texttt{BEX}(v)$, which runs an online learning algorithm, namely the Weighted Majority Algorithm of [12]. (For pseudocode, see Figure 5.) This algorithm depends on a parameter $\beta = 1 - \varepsilon$, where ε will be specified later after the analysis of the algorithm. (The optimum ε will be roughly $\sqrt[3]{m/HT}$.) The black box $\texttt{BEX}(v)$ outputs a probability distribution p_ϕ on the incoming edges of v during each phase ϕ. This probability distribution does not change during the phase. (Roughly speaking, $p_\phi(e)$ is proportional to $\beta^{c(e)}$ where $c(e)$ is the cost of edge e in the phases preceding ϕ.) At the end of a phase, $\texttt{BEX}(v)$ receives as input a vector containing a "simulated cost" $\tilde{C}_\phi(e) \in [0,1]$ for each incoming edge e of v. The black box satisfies the following performance guarantee:

$$\sum_\phi \sum_e p_\phi(e)\tilde{C}_\phi(e) \le \sum_\phi \tilde{C}_\phi(e_0) + O\left(\frac{\varepsilon T}{\tau} + \frac{\log(\Delta)}{\varepsilon}\right). \tag{1}$$

(Here Δ is the in-degree of v, and e_0 denotes any fixed incoming edge of v.) Roughly speaking, this means that the random edges output by $\texttt{BEX}(v)$ perform nearly as well as any single incoming edge e_0, if performance is measured according to the *simulated cost* $\tilde{C}_\phi$.

Every time step t within a phase ϕ is classified as either a *sampling step* or an *exploitation step*, according to the outcome of an independent random coin toss with probability δ of determining a sampling step. In an exploitation step each vertex v independently chooses an incoming edge using the distribution specified by $\texttt{BEX}(v)$; this edge set defines an arborescence directed away from s, and we choose the path joining s to r in this arborescence. In a sampling step, we choose an edge $e = (v, w)$ in G uniformly at random. We choose a random arborescence directed away from s using the same rule as in an exploitation step. We take the path in this arborescence from s to v, join it with the edge e, and the join it with "our favorite" path from w to r. (Our favorite path may be, for example, the path from w to r in some fixed arborescence directed toward r. The only important thing is that this path does not depend on e, only on w.) The sampling step is judged to be "successful" if there were no edge failures on the sub-path from s to w, otherwise unsuccessful.

The simulated cost of an edge e in a phase ϕ is the number of unsuccessful sampling steps for that edge, divided by the expected total number of sampling steps. To express this symbolically in terms of a formula, let A_t be the random arborescence chosen at time t using the black-box at each vertex, let $\pi_t(v)$ denote the path from s to v in this arborescence, and let $C_t(v)$ denote the maximum of $C_t(e)$ over all edges in $\pi_t(v)$. Finally let $\chi_t(e)$ denote the event that t is a sampling step for e and let $\sigma_t(e)$ denote the event that t is an unsuccessful sampling step for e. Then for an edge $e = (v, w)$,

$$\tilde{C}_\phi(e) = \frac{\sum_{t\in\phi} \sigma_t(e)}{\delta\tau/m}.$$

The pseudo-code for all steps in the algorithm is specified in Figures 3, 4, 5.

```
for  φ = 0, 1, ..., ⌊T/τ⌋
    for  t = τφ + 1, ..., τ(φ + 1)
        /* Choose routing paths for phase φ. */
        Call ROUTE procedure to obtain path π_t
                and distinguished edge e_t.
        π_t^+ ← ACK'ed sub-path of π_t.
        if e_t ≠ null and e_t ∉ π_t^+
            σ_t(e_t) ← 1
        σ_t(e) ← 0 for all other edges e.
    end
    for e ∈ E, /* Compute simulated edge costs. */
        C̃_φ(e) ← (Σ_{t∈φ} σ_t(e))/(δτ/m).
    for (u, v) ∈ E /* Update edge probabilities */
        ν_{φ+1}(u, v) ← BEX_φ(v)[u, v].
end /* End main loop */
```

Fig. 3. Routing algorithm.

```
Function ROUTE
T_R ← a fixed tree in G directed toward r.
for v ∈ V
    Sample a random incoming edge in(v) using
            probabilities ν_φ(·, v).
T_S ← {in(v) : v ∈ V}.
Flip coin with δ probability of heads.
if heads /* Exploration */
    Choose e_t = (v, w) uniformly at random from E;
    Π(v) ← the path from s to v in T_S;
    Π̃(w) ← the path from w to r in T_R;
    π_t ← Π(v) · (v → w) · Π̃(w)
else /* Exploitation */
    e_t ← null;
    π_t ← the path from s to r in T_S;
return (π_t, e_t)
```

Fig. 4. Path generation procedure ROUTE with parameter δ.

```
Function BEX_φ(v)[u, v]
/* Sum weights of incoming edges to v. */
W ← Σ_{(w,v)∈E} β^{C̃_φ(w,v)}
/* Return probability of edge (u, v). */
return β^{C̃_φ(u,v)}/W
```

Fig. 5. $\texttt{BEX}_\phi(v)$: Black box $\texttt{BEX}_\phi(v)$ with parameter β using scores $\tilde{C}_\phi(v, w)$; picks edge (v, u) with prob. $\nu_\phi(v, u)$.

VI. Analysis

A. Overview

Our goal is to prove, for *every* vertex v, that the random path $\pi_t(v)$ is nearly the optimum path from s to v, up to a regret term which depends on the distance from s to v. This claim will be established by induction on the distance from s to v, the base case $s = v$ being trivial.

In order for this idea to work, we need a provable relationship between our simulated costs and the true costs. Proving this in the context of an adaptive adversary is trickier than in the oblivious case, which explains why we have a subtler definition of $\tilde{C}_\phi$ than in [3].

The bound on the algorithm's performance is expressed in terms of two notions of cost for a path π from s to r. For such a path, we define $BCOST_t(\pi)$ to be 1 if any edge of π failed at time t, and 0 otherwise. We define $ACOST_t(\pi)$ to be the total number of edge failures on π at time t. Finally, we define

$$\begin{aligned} BCOST(\pi) &= \frac{1}{T}\sum_{t=1}^{T} BCOST_t(\pi) \\ ACOST(\pi) &= \frac{1}{T}\sum_{t=1}^{T} ACOST_t(\pi). \end{aligned}$$

Thus, $BCOST(\pi)$ measures the average number of times the path π failed, and $ACOST(\pi)$ measures the average cumulative number of edge failures on π.

Theorem 6.1: For appropriate choices of the parameters $\tau, \varepsilon, \delta$, the algorithm's performance satisfies the bound

$$\begin{aligned} &\mathbf{E}(BCOST(\text{algorithm})) \le \mathbf{E}(ACOST(\pi)) \\ &\quad + \quad O\left(\left(\frac{H^2 m \log(mT)\log(\Delta)}{T}\right)^{1/3}\right). \end{aligned}$$

Note that this implies that for T sufficiently large, i.e. for $T = \omega(H^2 m \log(mT)\log(\Delta))$, the actual cost of the algorithm $\mathbf{E}(BCOST(\text{algorithm})$ will only be marginally larger than the benchmark cost $\mathbf{E}(ACOST(\pi))$. The following subsections will prove the theorem, and Section VII will examine what this asymptotic bound means in practice.

B. Characterization of $\tilde{C}_\phi$

For a random variable Y, let $\mathbf{E}_t(Y)$ denote the conditional expectation of Y, conditioned on all of the algorithm's random decisions before time t.

Lemma 6.2: For each edge $e = (v, w)$ and each time step t, $\mathbf{E}_t(\sigma_t(e)) \le \frac{\delta}{m}\left[\mathbf{E}_t(C_t(e)) + \mathbf{E}_t(C_t(v))\right]$.

Proof: From the definition of $\sigma_t(e)$ we have

$$\sigma_t(e) = \chi_t(e)\ \max\{C_t(e), C_t(v)\}.$$

Hence

$$\begin{aligned} &\mathbf{E}_t(\sigma_t(e)) \\ &\quad = \quad \mathbf{E}_t(\chi_t(e)) \cdot \mathbf{E}_t(\max\{C_t(e), C_t(v)\} \,\|\, \chi_t(e) = 1) \\ &\quad \le \quad (\delta/m)\,\mathbf{E}_t(C_t(e) + C_t(v) \,\|\, \chi_t(e) = 1). \end{aligned}$$

The lemma now follows, because the random variables $C_t(e)$ and $C_t(v)$ are independent of $\chi_t(e)$. In the case of $C_t(e)$, this is simply because the algorithm's decision of whether or not t will be a sampling step for e is independent of the adaptive adversary's choice of edge costs at time t. In the case of $C_t(v)$, it is also because the random path $\pi_t(v)$ is sampled from a distribution which does not depend on whether t is a sampling step for edge e. ∎

Corollary 6.3:

$$\tau\mathbf{E}(\tilde{C}_\phi(e)) \le \left[\mathbf{E}(C_t(e)) + \mathbf{E}(C_t(v))\right].$$

Lemma 6.4: For a vertex v, let $E^-(v)$ be the set of incoming edges of v. For a time step t in phase ϕ,

$$\mathbf{E}_t(C_t(\pi_t(v))) = \frac{m}{\delta}\left(\sum_{e\in E^-(v)} p_\phi(e)\mathbf{E}_t(\sigma_t(e))\right).$$

Proof: Both sides are equal to the probability of an edge failure on a random path sampled according to the following rule: choose a random incoming edge at v according to the probability distribution p_ϕ, let u be the other endpoint of this edge, and join the edge to the random path $\pi_t(u)$. ∎

The other thing we need to know about $\tilde{C}_\phi$ before continuing with the analysis is that it is bounded above by a constant, with high probability.

Lemma 6.5: If $\tau \ge m\log(mT)/\delta$ then with probability at least $1 - 1/mT$, $\tilde{C}_\phi(e) < 5$ for all edges e and all phases ϕ.

Proof: Under the hypothesis on τ the expected number of sampling steps for any edge in any phase is at least $\log(mT)$. The true number of sampling steps for edge e in phase ϕ is a sum of independent Bernoulli random variables, so Chernoff's bound tells us that

$$\begin{aligned} &\Pr\left(\sum_{t\in\phi}\chi_t(e) > 5\log(mT)\right) \\ &\quad < \quad \left[e^4/5^5\right]^{\log(mT)} < e^{-2\log(mT)} = (1/mT)^2. \end{aligned}$$

Using the fact that $\sigma_t(e) \le \chi_t(e)$, this immediately gives an upper bound of $(1/mT)^2$ on the probability that $\tilde{C}_\phi(e) > 5$. Summing over all pairs (e, ϕ), we obtain the result stated in the lemma. ∎

In the following analysis, we will assume throughout that $\tilde{C}_\phi(e)$ is bounded above by 5. The probability that this assumption fails is $< 1/mT$, and in the event that it fails the total cost of all paths is at most T, so this event contributes at most $1/m$ to the algorithm's expected cost and may therefore be ignored.

C. Local performance guarantee

The induction step of the analysis is a local performance guarantee relating the average cost of $\pi_t(w)$ to the average cost of $\pi_t(v)$, where v is an upstream neighbor of w in G.

Lemma 6.6: If $e = (v, w)$ is any edge of G, then

$$\frac{1}{T}\sum_{t=1}^{T}\mathbf{E}(C_t(\pi_t(w))) \leq \frac{1}{T}\sum_{t=1}^{T}\mathbf{E}(C_t(\pi_t(v))) + \tfrac{1}{T}\textstyle\sum_{t=1}^{T}\mathbf{E}(C_t(e)) + O\left(\varepsilon + \frac{\tau\log(\Delta)}{\varepsilon T}\right).$$

Proof: The performance guarantee for $\texttt{BEX}(w)$ ensures that

$$\frac{\tau}{T}\sum_{\phi}\sum_{e\in E^-(w)} p_\phi(e)\tilde{C}_\phi(e) \leq \frac{\tau}{T}\sum_{\phi}\tilde{C}_\phi(e_0) + O\left(\varepsilon + \frac{\tau\log(\Delta)}{\varepsilon T}\right).$$

Take the expectation of both sides of this inequality. Using Lemma 6.4, the expectation of the left side is

$$\frac{\tau}{T}\left(\sum_{t=1}^{T}\sum_{e\in E^-(w)} p_\phi(e)\mathbf{E}(\sigma_t(e))\right)\frac{m}{\tau\delta} = \frac{1}{T}\left(\sum_{t=1}^{T}\mathbf{E}(C_t(\pi_t(w)))\right).$$

Using Corollary 6.3, the expectation of the right side is bounded above by

$$\frac{1}{T}\sum_{t=1}^{T}\mathbf{E}(C_t(\pi_t(v))) + \frac{1}{T}\sum_{t=1}^{T}\mathbf{E}(C_t(e)) + O\left(\varepsilon + \frac{\tau\log(\Delta)}{\varepsilon T}\right).$$

■

D. Global performance guarantee

Let π be any static path from s to r. Denote the nodes of π by $s = v_0, v_1, \ldots, v_H = r$, and let π_j denote the subpath of π joining s to v_j. Using induction on j, Lemma 6.6 implies the following regret bound for each j:

$$\frac{1}{T}\sum_{t=1}^{T}\mathbf{E}(C_t(\pi_t(v_j))) \leq \frac{1}{T}\sum_{t=1}^{T}\sum_{e\in\pi_j}\mathbf{E}(C_t(e)) + O\left(\varepsilon j + \frac{\tau\log(\Delta)j}{\varepsilon T}\right). \quad (2)$$

Applying (2) for $j = H$, and adding in the expected cost of the sampling steps, which is bounded above by δT, we obtain

$$\mathbf{E}(BCOST(\text{algorithm})) \leq \mathbf{E}(ACOST(\pi)) + O\left(\varepsilon H + \frac{\tau\log(\Delta)H}{\varepsilon T} + \delta\right).$$

Recall that τ is constrained to be at least $m\log(mT)/\delta$. We optimize the right side by setting

$$\varepsilon = \left(\frac{m\log(mT)\log(\Delta)}{HT}\right)^{1/3}$$
$$\delta = \varepsilon H$$
$$\tau = m\log(mT)/\delta$$

which leads to the bound asserted in Theorem 6.1.

VII. Implementation Results

In order to substantiate the claims made in this work, simulations were conducted to investigate the convergence time of the algorithm. The simulations were conducted by developing a simple program which would simulate the decision-making process of the algorithm and examine its performance against adversarial inputs. The simulation consisted of a source selecting a path to the destination at each time step. An adversarial model would then select which nodes at the current time step were faulty. The packet would traverse the graph and receive positive feedback from the destination if there were no faulty nodes on the path, or from the last non-faulty node before the packet was dropped. Using this feedback the algorithm would adjust its probabilities and compute a new path for the next packet.

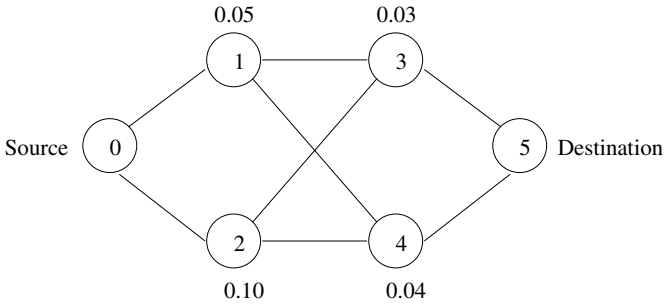

Fig. 6. Simple Simulation Topology

A. Simple Configuration

The first set of simulations consisted of 10,000 packets which were sent from the source to the destination. The nodes were arranged as indicated in figure 6. The intermediary nodes are labelled 0 through 5 and their fault rates are indicated. At every time step each node was probabilistically selected to fail based on the node's fault rate. On this particular simple configuration the optimal path would be from the source to node 1, from node 1 to node 3, and then from node 3 to node 5. The results of the simulation are the source's estimated link preference metrics at every time step. The experiment was run for different values of β, which is the base of the exponent described in the algorithm specification. As the value of β decreases, the algorithm responds faster to changes and is able to converge faster. However, as the algorithm responds faster it begins to resemble the greedy algorithm which has

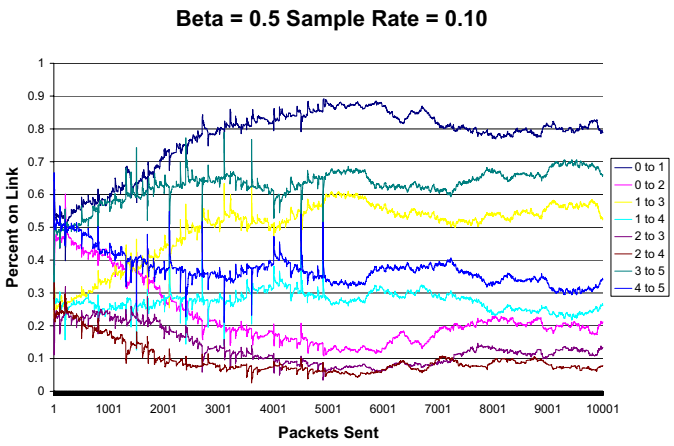

Fig. 7. Simple Configuration Results

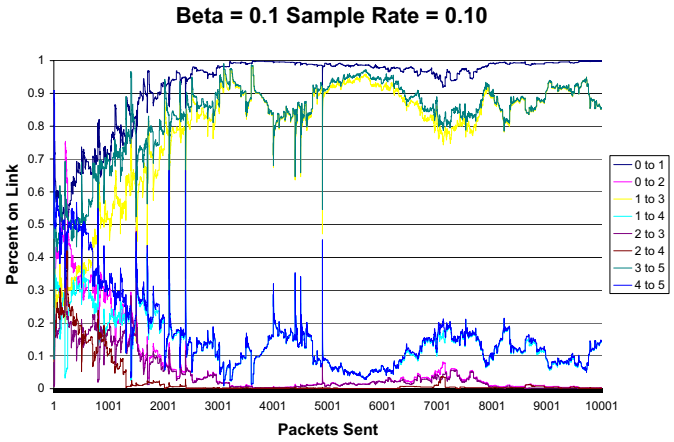

Fig. 8. Simple Configuration Results

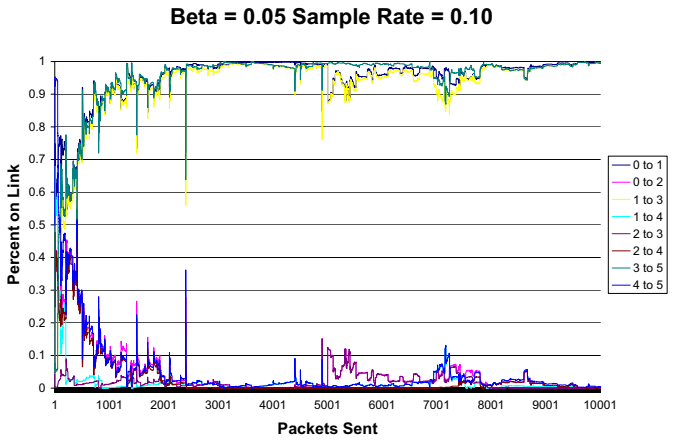

Fig. 9. Simple Configuration Results

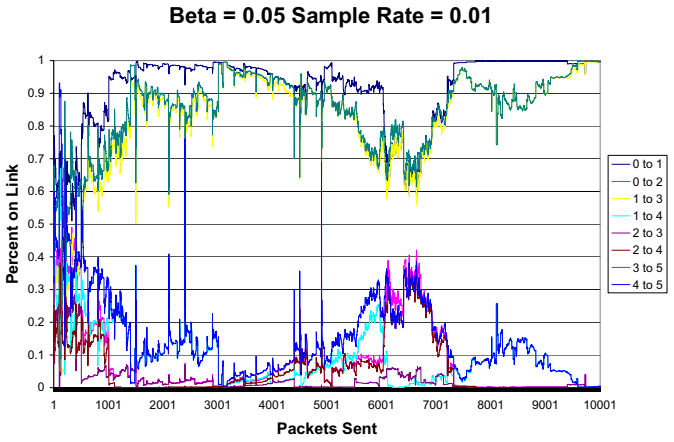

Fig. 10. Simple Configuration Results

vulnerabilities. In order to explore the effects of β on the convergence time of the algorithm, multiple experiments were run with the same adversarial input. The results are indicated in Figures 7,8, and 9. These figures show the probability of the source selecting a specific edge at every time step.

With β=0.5 the results show that the source is slowly realizing which path is correct and is sending a majority of its traffic over the correct sequence of nodes. Notice that once the algorithm converged it was sending approximately 90% of its traffic across the first hop to node 1 which was experiencing a 5% loss rate and only 10% of its traffic to node 2 which had a 10% loss rate. While this seems reasonable, notice the convergence when β=0.1 or, better yet, 0.05. With these values the source is able to completely differentiate between nodes and converge quickly on the best path. Since the algorithm is continuously exploring sub-optimal paths with a small fraction of its traffic, this low value of β allows it to react quickly as the adversary changes its fault pattern. While the optimal algorithm might know the fault pattern ahead of time, the algorithm we present is able to follow it very closely. If the speed at which the adversary moves is slightly slower then our decision making process, then our algorithm would in fact follow the optimal at every step.

In order to investigate the effects of the sampling percentage on the convergence time an additional experiment was done using the same simple configuration described above, but with the sampling rate decreased from 10% to 1%. The results of this experiment are indicated in Figure 10. It appears that the lower sampling rate inhibits the algorithm's ability to converge as rapidly as the 10% sampling rate indicated in Figure 9.

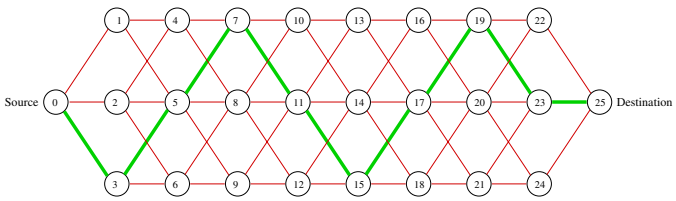

Fig. 11. Large Simulation Topology

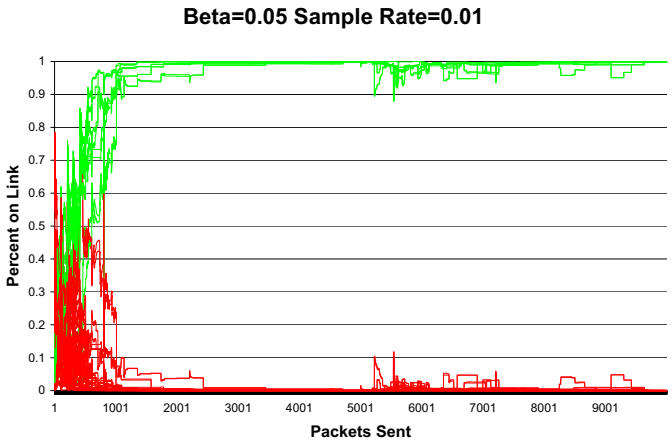

Fig. 12. Large Simulation Results

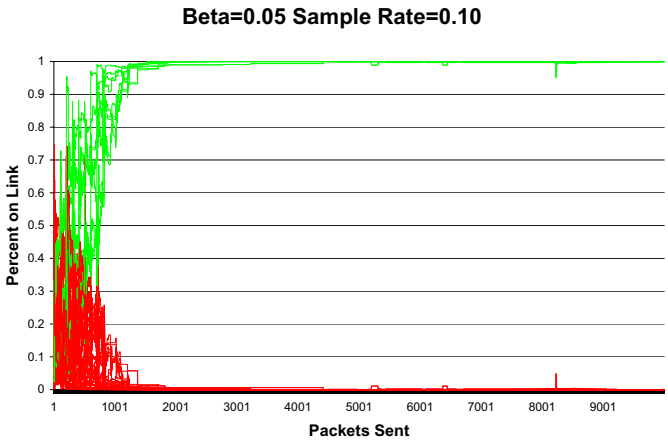

Fig. 13. Large Simulation Results

B. Large Configuration

The previous example provided a demonstration of the effects of various parameters on the performance of our algorithm. While the previous example showed the convergence of the algorithm, it is somewhat less interesting since the example consisted of a small set of both nodes and links. In this second example we consider a network with 25 nodes forming a graph of 10 layers, with 3 nodes in each layer (except at the source and destination). The topology of the network is indicated in Figure 11. In this example there is exists an optimal path from the source to the destination which experiences no loss. This optimal path is indicated in Figure 11 by the bold line. All other links in the network exhibit 10% loss, meaning that when we sample them, they successfully forward the packet 90% of the time. As a result, this should make it more difficult for our algorithm to discover the optimal path since the non-optimal paths in the network only experience marginal loss. This simulation consisted of a source attempting to deliver 10,000 packets to the destination. The graphs in Figures 12 and 13 show the results when we are performing random sampling with 1% and 10% of our packets respectively. In this experiment the value of β was set to 0.05.

The results indicate that the algorithm is able to successfully converge on the optimal path after approximately 1000 packets are sent. Once the algorithm learned the best path it was able to send approximately 99% of its traffic successfully to the destination. The graph visually indicates this by showing the source's link preferences at every time step. When the simulation begins the source considers all of the links in the network to be equal and then learns their reliability by sending traffic across the links and receiving feedback. As the number of packets (or trials) increases, the source's knowledge of the network continuously becomes more accurate. This is evident as the reliable paths become separated from the less reliable paths and are selected with nearly 100% probability. Since the source is continuously sampling the less desirable edges it is able to respond quickly to changes in the adversarial fault pattern.

CONCLUSION

Through mathematical analysis and simulation results we have presented an online adaptive routing algorithm and have shown the algorithm's competitive performance under a strong adversarial model consisting of dynamic proactive adversarial attacks, where the network may be completely controlled by adaptive adversaries.

The results of this work affirm the validity of our approach and motivate the need for future work in this direction. We intend on implementing this protocol in a more realistic simulation environment and exploring the effects of both mobility and more sophisticated active adversarial attacks on the algorithm.

REFERENCES

[1] David G. Andersen, Hari Balakrishnan, M. Frans Kaashoek, and Robert Morris. Resilient overlay networks. In *Proceedings 18th ACM SOSP*, Banff, Canada, 10 October 2001.

[2] Baruch Awerbuch, Dave Holmer, Cristina Nita-Rotaru, and Herb Rubens. An on-demand secure routing protocol resilent to byzantine failures. In *Wireless Security Workshop Proceedings*, September 2002.

[3] Baruch Awerbuch and Yishay Mansour. Online learning of reliable network paths. In *PODC*, 2003.

[4] John S. Baras and Harsh Mehta. Dynamic adaptive routing in manets. In *Proc. Annual ARL CTA Symposium*, 2003.

[5] E. Bonabeau, F. Henaux, S. Guérin, D. Snyers, P. Kuntz, and G. Theraulaz. Routing in telecommunications networks with "smart" ant-like agents. In *Intelligent Agents for Telecommunications Applications '98 (IATA'98)*, 1998.

[6] G. Di Caro and M. Dorigo. AntNet: a mobile agents approach to adaptive routing. Technical Report IRIDIA/97-12, Université Libre de Bruxelles, Belgium, 1997.

[7] Gianni Di Caro and Marco Dorigo. Antnet: Distributed stigmergetic control for communications networks. *Journal of Artificial Intelligence Research*, 9:317–365, 1998.

[8] Nicolò Cesa-Bianchi, Yoav Freund, David P. Helmbold, David Haussler, Robert E. Schapire, and Manfred K. Warmuth. How to use expert advice. In *Proc. 25th ACM Symp. on Theory of Computing*, pages 382–391, 1993. To appear, *Journal of the Association for Computing Machinery*.

[9] P. Druschel D. Subramaniam and J. Chen. Ants and reinforcement learning : A case study in routing in dynamic networks. In *Proceedings of IEEE MILCOM, Atlantic City, NJ*, 1997.

[10] Marco Dorigo, Vittorio Maniezzo, and Alberto Colorni. The Ant System: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics Part B: Cybernetics*, 26(1):29–41, 1996.

[11] Adam Kalai and Santosh Vempala. Efficient algorithms for the online decision problem. In *Proc. of 16th Conf. on Computational Learning Theory, Wash. DC*, 2003.

[12] Nick Littlestone and Manfred K. Warmuth. The weighted majority algorithm. *Information and Computation*, 108:212–261, 1994. A preliminary version appeared in FOCS 1989.

[13] M. Littman and J. Boyan. A distributed reinforcement learning scheme for network routing. In *Proceedings of the International Workshop on Applications of Neural Networks to Telecommunications. Alspector, J., Goodman, R. and Brown, T. X. (Ed.)*, pages 45–51, 1993.

[14] U. Sorges M. Gunes and I. Bouazizi. The ant colony based routing algorithm for manets. In *Proc. of the 2002 ICPP Workshop on Ad Hoc Networks (IWAHN 2002)*, pages 79–85. IEEE Computer Society Press, 2002.

[15] U. Sorges M. Gunes and I. Bouazizi. "ara -" the ant colony based routing algorithm for manets. In *Proc. 2002 ICPP Workshop on Ad Hoc Networks (IWAHN 2002)*, pages 79–85. IEEE Computer Society Press Stephan Olariu edt., 2002.

[16] C. K. Tham S. Marwaha and D. Srinivasan. Mobile agents based routing protocol for mobile ad hoc networks. In *in Proceedings of IEEE Globecom,*, 2002.

[17] R. Schoonderwoerd, O. E. Holland, and J. L. Bruten. Ant-like agents for load balancing in telecommunications networks. In *Proc. 1st ACM Intl. Conference on Autonomous Agents, Marina del Rey, California*, pages 209–216, 1997.

[18] Ruud Schoonderwoerd, Owen E. Holland, Janet L. Bruten, and Leon J. M. Rothkrantz. Ant-based load balancing in telecommunications networks. *Adaptive Behavior*, 5(2):169–207, 1996.

[19] Sleator and Tarjan. Amortized efficiency of list update and paging rules. *Comm. ACM*, 28(2):202–208, 1985.

[20] Devika Subramanian, Peter Druschel, and Johnny Chen. Ants and reinforcement learning: A case study in routing in dynamic networks. In *IJCAI (2)*, pages 832–839, 1997.

[21] Eiji Takimoto and Manfred K. Warmuth. Path kernels and multiplicative updates. In *COLT Proceedings*, 2002.