A Robust Routing Algorithm for Overlay Networks

Baruch Awerbuch, Andreas Terzis

May 16, 2003

Abstract

A number of different Overlay network designs have been proposed to provide a large range of services ranging from content delivery and application-level multicast to robust data delivery in the presence of underlying network path outages and periods of degraded network performance. Previous work has shown that overlay networks can be very effective in protecting against faults that are due to the underlying network conditions. Our goal is to extend the existing model of protecting against network faults where faults are due to malicious attacks that specifically target the overlay network. We further assume that the adversaries can generate any fault pattern and have full knowledge of the overlay network including the routing algorithm used.

We present in this paper a novel routing algorithm that is provably competitive against the best offline routing strategy. Our algorithm works by measuring the reliability of the overlay links and then probabilistically routing packets along multiple paths using the computed reliability metrics. We have simulated the algorithm and our experimental results show that our solution performs as well as existing overlay routing algorithms in the case of non-malicious errors but offers orders of magnitude improvement over existing algorithms in the case of malicious attacks.

1 Introduction

Recent studies [18, 1] have shown that Internet routing is *suboptimal* in the sense that it does not follow the "optimal" path from the endpoints' point of view. At the same time, other studies[10, 11] have shown that Internet routing is slow to converge to a new path after an outage has occured resulting in prolonged periods where packets can be delivered to their final destination. To make matters even worse, Distributed Denial of Service attacks are becoming prevalent [15] further reducing the fidelity of Internet paths.

Recently, Overlay networks such as RON[1] and [18] have been proposed to solve the problems of path outages and performance failures. The basic idea, is that overlay network nodes detect failures in the underlying network and route *around* them by redirecting traffic through other nodes in the overlay network.

While previous attempts in this area have focused on combating against "random" faults we present in this paper a novel algorithm that is able to overcome faults caused by a malicious *adversary* that has total knowledge of the routing algorithm used by the overlay network and also has the capability to generate losses on any network path. In addition, our algorithm perfoms against random faults as well as existing algorithms. Our routing algorithm works by calculating the *reputation* of different network links and sending packets on the paths that have the highest repuation.

We simulated our algorithm for both random as well as malicious fault patterns in synthetic and realistic topologies. Our experimental results show that our algorithm perfoms as well as RON for random fault patterns but it is not vulnerable to malicious DoS attacks. In one of the tested scenarios against a malicious adversary, while the RON strategy resulted in 97% of lost packets, our algorithm was able to deliver close to the optimal 33% of the packets.

The contributions of this paper are:

1. We propose a new routing algorith that can route around random faults as well as faults caused by a malicious adversary. The performance of our algorithm can be proved theoretically.

2. We experimentally verified that our algorithm performs comparably to existing overlay routing algorithms for random error patterns but it's not vulnerable to DoS attacks.

The rest of the paper is organized as follows. In Section 2 we describe the overlay model and introduce the failure model. In Section 3 we present our algorithm and give the intuitive as well as an outline of the mathematical proof of our algorithm's performance. Section 5 presents our results where we show the performance

of our algorithm against RON in adversarial and non-adversarial cases. We place our work in the context of related work in Section 6 and we close in Section 7 with a summary of our results and our thoughts for future work in this area.

2 Background

2.1 Overlay Networks

As [8] argues, overlay networks have many advantages:

Incrementally Deployable An overlay network requires no changes to the existing Internet infrastructure, only additional servers. As nodes are added to an overlay network, it becomes possible to control the paths of data in the substrate network with ever greater precision.

Adaptable Although an overlay network abstraction constrains packets to flow over a constrained set of links, that set of links is constantly being optimized over metrics that matter to the application.

Robust By virtue of the increased control and the adaptable nature of overlay networks, an overlay network can be more robust than the substrate fabric. For instance, with a sufficient number of nodes deployed, an overlay network may be able to guarantee that it is able to route between any two nodes in two independent ways. While a robust substrate network can be expected to repair faults eventually, such an overlay network might be able to route around faults immediately.

Customizable Overlay nodes may be multi-purpose computers, easily outfitted with whatever equipment makes sense. For example, Overcast makes extensive use of disk space. This allows Overcast to provide bandwidth savings even when content is not consumed simultaneously in different parts of the network.

Adaptivity and robustness are two of the major driving forces behind overlay networks. In [1], the authors describe a system designed to realize this potential by utilizing redundant network connections among overlay network participants. The goal of RON as presented by the authors is to enable a group of nodes to communicate with each other in the face of problems with the underlying Internet paths connecting them. RON detects problems by aggressively probing and monitoring the paths connecting its nodes. If the underlying Internet path is the best one, the path is used and no other RON node is involved in the forwarding path. If the Internet path is not the best one, the RON will forward the packet by way of other RON nodes. RON nodes exchange information about the quality of the the paths among themselves via a routing protocol and build forwarding tables based on a variety of path metrics, including latency, packet loss rate and available throughput. Each RON node obtains the path metric using a combination of active probing experiment and passive observations of on-going data transfers. RON networks by design are small and organized as a fully-connected network so each node collects path information on the paths to each other node in the overlay. RON uses a link-state routing protocol to disseminate topology information between nodes which in turn is used to build forwarding tables. Each node in an N-node RON has N-1 virtual links. Each node periodically requests summary information of the different performance metrics to the $N-1$ other nodes from its local performance database and disseminates its view to the others. To estimate loss rates, RON uses the average of the last $k = 100$ probe samples as the current average. Loss metrics are multiplicative on a path: if we assume that losses are independent, the probability of success on the entire path is roughly equal to the probability of surviving all hops individually: $lossrate_{path} = 1 - \prod_{l \in path}(1 - lossrate_l)$

2.2 Failure model

Internet paths losses due to congestion and physical errors (although the percentage of physical errors is very small at the core of the network). Paxson in [17] studied the loss rate for a number of Internet paths and found that it ranged from 0.6% to 5.2%. Furthermore in this study and a follow-up [24], Paxson discovered that loss processes can be modeled as spikes where loss occurs according to a two-state process, where the states are either "packets not lost" or "packets lost". According to the same studies, most loss spikes are very short-lived (95% are 220 ms or shorter) but outage duration spans several orders of magnitude and in some cases the duration can be modeled by a Pareto distribution.

The other major factor contributing to packet losses is Internet path outages due to equipment and link failures. Labovitz et al [10] use a combination of measurements and analysis to show that inter-domain routes in the Internet may take tens of minutes to reach a consistent view of the network topology after a fault. They find that during this period of "delayed convergence", end-to-end communication is adversely affected. In fact, outages on the order of minutes cause active TCP connections to terminate when TCP does not receive an acknowledgment for its outstanding data.

In [11], Labovitz et al. find that 10% of all considered routes were available less than 95% of the time and that

less than 35% of all routes were available more than 99.99% of the time. In a followup study [7], it was shown that 5% of all failures last more than 2 hours and that failure durations are heavy-tailed and can last as long as 20 hours before being repaired.

The Detour measurement study [19] made the observation that path selection in the wide-area Internet is sub-optimal from the standpoint of end-to-end latency, packet loss rate and TCP throughput. This study showed that for 75 to 85 percent of the paths there were alternates with lower loss rate and that for roughly 10% of the paths the best alternative has 50% better latency. The authors of this study argued for the first time about the potential benefits of using *waypoints* to route packets via intermediate nodes by comparing the long-term average properties of detoured paths against Internet chosen paths.

In addition to these "benevolent" packet losses that are part of the normal operation of the Internet, a new type of losses is wide-spread in the Internet today. These are losses due to malicious activity such as Denial of Service (DoS) attacks. Particularly in Distributed Denial of Service attacks, adversaries can use the resources of large numbers of end hosts or even routers (in the case of reflector attacks) to inundate their victims with a flood of packets. These huge packet flows not only overwhelm the resources of the victims but also severely congest the Internet paths leading to those end hosts. In [15] Moore et al. investigated the number and severity of DoS around the globe by observing the attacks' side effects on a network they monitored. During a period of one week they monitored 12,805 attacks targeted against 5,000 distinct victim IP addresses. Half of the observed attacks had a packet rate greater than 350 packets per second whereas the fastest attack had a rate of 679,000 packets per second. The duration of these attacks also varies a lot: while 50% of the attacks are less than 10 minutes in duration and 80% are less than 30 minutes, 1% are greater than 10 hours and dozens spanned multiple days!

Given the presence of malicious and endemic losses in the Internet today, we believe that an *adversarial* model is the appropriate model under which the performance of overlay routing algorithms should be compared. In this model, the adversary has potentially complete knowledge of the routing algorithm used by the overlay network (although might not know in advance the parameter values used by the algorithm) and furthermore has the capability to generate losses on each network path for any duration of time. While such an adversary is considerably more powerful compared to random packet losses, this model is not far from the reality today where software is developed using the open source model and attackers can have millions of end hosts at their disposal to execute their attacks.

3 Vulnerabilities of dynamic routing

Consider a generic dynamic routing algorithm where each node is measuring the *reliability* of its adjacent links, as in RON [1].

After the reliability of each link has been measured the nodes exchange these reliability metrics as they would exchange routing metrics. The reliability of a path is then computed as the product of the reliabilities of each of the links on that path. After calculating the different path reliabilities each node send a packet towards a destination by using *all* the paths. A packet is sent along path i with probability equal to $p_i / \sum_{k=1}^{N} p_k$

In general, there may not be an ideal fault-free path, but yet there may be 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. The problem is that we are making decisions "online", where the online algorithm needs to service requests by selecting actions while having only information regarding past requests, and no (or very limited) information regarding the future requests. There are no stochastic assumptions about the way adversary is acting. are generated. Moreover, it is also assumed that there is a powerful adversary that generates the worse input sequence for the online algorithm.

One may be tempted to think that converging to best fixed route may be easily accomplished by "greedy" heuristic: keep track of past error rates on different links, and simply select the path with the smallest overall failure rate, namely sum of failure rates of its links. The intuition is very strong: extrapolate past failure pattern into the future. This may work if failure pattern is static or if there is a statistical model of failures.

However this method fails quite spectacularly in case of dynamic adversaries. For example, consider 100 parallel path between sender and receiver. At time i path i (modulo 100) is under control of the adversary, and is failing all packets; at all other times path is perfect. The greedy algorithm will always pick the worst path, even though at nay moment in time only 1% of paths is faulty.

To see the principle flaw in this greedy strategy, consider performance of an investor who tries to follow the best performing stock on the stock market, with the naive assumption that "past performance is guarantee

of future results". In fact, in an adversarial setting, e.g., the stock market, it may well be the case that past performance correlates *negatively* with future results, and algorithms must not be fooled into easy trap.

Another "quick fix" is to try to select random path. This method fails in situation where adversary is firmly in control of 99 out of 100 paths. There is something appealing about random strategies in the sense they allows to spread the risk; what is wrong with the above quick fix is that there is no attempt to adapt the probabilities.

A fundamental question for online algorithms is how to evaluate their performance. It is impossible rare that one algorithm *always* outperforms another algorithm. One classical method has been to assume a stochastic model regarding the environment, and compare the performance of different algorithms on the same model. However, in adversarial setting, there is no reason why adversary should follow the rules of the stochastic model we invented - if anything, adversary will do exactly the *the opposite* of what our model would predict. (There no reason to hope adversary will not know our model.)

In has been generally recognized that sending packets along random path is more "fault-tolerant" than sending packet over a single path. Our goal is thus to find a robust randomized algorithm that works well on all inputs, in the sense that expected behavior of our algorithm is comparable to optimum fixed path on each input. Our goal is to design a distributed algorithms sending messages on paths with overall smallest loss ratio. The contribution of this paper is a novel distributed algorithm for choosing routes that and its proof of *optimality* in the sense that the total number of messages lost in our algorithm exhibits a very small additive gap with *optimum prescient* route assignment. The optimum assignment is made with complete knowledge of adversary actions, in the past, current, and future, and has infinite computational power. The only restriction on optimum route is that it must change infrequently; for simplicity of exposition we consider first the case that the route is fixed.

4 Outline of our approach

The solution uses the following components.

- *Local generation of link reputation.* Accumulated fault rate are transformed into "reputation" for each link. The *direct* reputation of the link is inversely proportional *exponential* of the past loss rate. (Our reputation values are essentially the inverse values of weights in [2]).

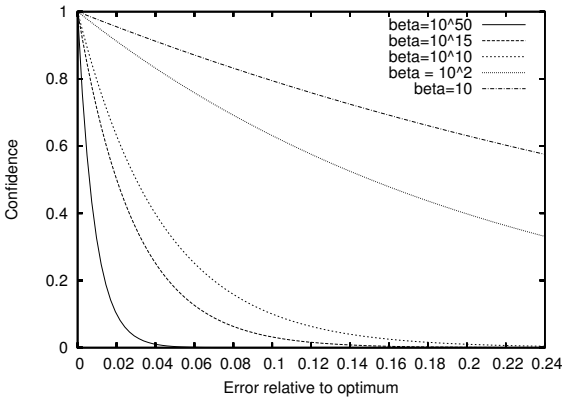

Figure 1: Confidence as a function of β

- *Dissemination.* After these "reputation" values are measured locally by endpoints of edge being sampled, they are *signed* by each endpoint and *flooded* thru the network, as in link-state routing algorithms.

- Upon receipt of Based on link reputations, we compute hop-reputations for nodes. The h-hop reputation of a node is the online estimate of the likelihood that node is capable of reaching destination on a path with h hops. Specifically, 0-hop reputation is 1 for receiver and 0 for everyone else. The h-reputation of an adjacent link is product of $h-1$-reputation of the endpoint of that link, and direct link reputation. Finally, h-reputation of a node is sum of $h-1$ reputation of the links.

- when node receives packet (for a certain destination), that traversed exactly $h-i$ links, it will forward the packet on its next edge with probability proportional to i-reputation of the links.

4.1 Reliability Measurements

Each overlay node sends periodic probes to each of its neighbors to measure the reliability of the links connecting the node with its neighbors. Each node sends a probe every 1 sec and if it does not receive an acknowledgement from the neighbor withing some period of time it considers that the probe was lost. These probes produce a statistical estimate on the *packet loss* on each overlay network edge. The loss rate is calculated using a sliding window of the last 100 probes sent.

4.2 Transforming a graph into a layered directed graph

As it turns out, it is more convenient to explain the algorithm in layered directed graphs. In this section we show how to accomplish this.

Transforming a graph into a layered directed graph. Without loss of generality, we assume that we can transform original undirected network graph $G(V, E)$ (e.g., see Fig. 2) into a directed layered graph, with the receiver being in layer 0 of this graph (Fig,3).

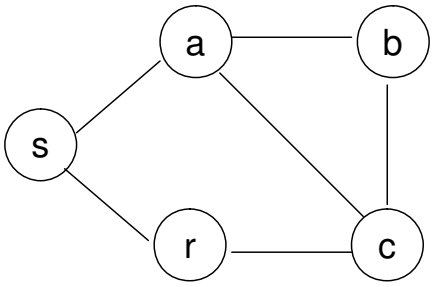

Figure 2: The original network.

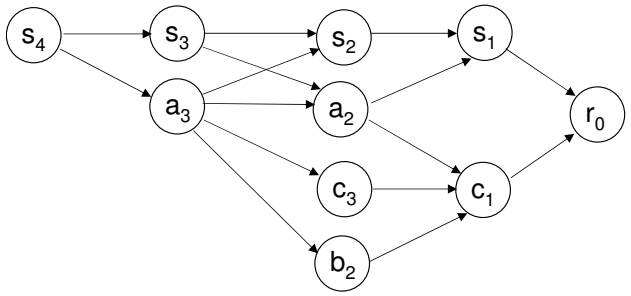

Figure 3: Layered graph w.r.t. receiver r (in layer 0).

All the nodes that can *potentially* reach the receiver in i hops (or less), for $0 \leq i \leq H$ are *represented* in layer i of the graph. Here H is the upper bound on hop count of a routing path. The directed edges connect these 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 packet arrives to node v after traversing i hops, we consider the packet as if it arrives at "virtual" node v_{H-i}. We can consider a directed levelled acyclic "virtual" graph $G'(V', E')$ where

$$V' = \{v_i | v \in V, i \leq H\}$$

$$E' = \{v_i \rightarrow u_{i+1} | (u, v) \in E \text{ or } u = v\}$$

Suppose we have a sender s, a receiver r, and a hop count H. The algorithm constructing the graph is given in fig. 16.

$\tilde{V}^0 \leftarrow s_0$
repeat $j = 0$ **downto** $H - 1$
  $\tilde{V}^{j+1} \leftarrow \{v_{j+1} \mid \exists u_j \in V^j, (u, v) \in E\}$
  $\tilde{E}^{j+1} \leftarrow \{v_{j+1} \rightarrow u_j \mid \exists u_j \in V^j\ , (u, v) \in E\}$
  remove all edges/vertices unreachable from s_H

Figure 4: Algorithm for constructing layered graph.

We observe that a directed leveled graph G' of depth H simulates any communication where packet traverses a bounded number of hops $H \leq |V|$. Thus, for the rest of the paper, we consider, without loss of generality, our network graph to be leveled, directed, acyclic, and every node is reachable from the source.

Now, we will imagine a node s and r which we refer to as the "source" and "sink" nodes respectively. The sink node r will be placed in layer 0 of the graph.

4.3 Reputation-based solution

Our setting is similar to the classic "online experts" setting in [12], and we will base our solution on the algorithms in [12]. The major difficulty in this construction is the fact that the number of experts is exponential, which makes the algorithms in [12] infeasible. We also comment that Kalai and Vempala have independently suggested an alternative framework to handle exponential number of choice using general LP-based geometric approach [9] that is applicable to oblivious adversaries.

Consider $\gamma(e, t)$ being the current measurement of loss of a link e at time t.

Define the average error rate in a sliding window of length T (namely, the result of our measurements) as

$$\Gamma(e, t) = \sum_{\tau=t-T_1}^{t} \gamma(e, \tau)$$

As in the weighted majority learning algorithm [12], define a *reputation* of an edge e at time t:

$$\Lambda(e, t) = \beta^{-\Gamma(e,t)}$$

Analogously, define for each path of length H $\Pi =< e_1, \ldots, e_H >$ of length H as product of reputations of individual links

$$\Lambda(\Pi, t) = \prod_{e_i \in \Pi} \Lambda(e_i, t) = \beta^{-\Gamma(\Pi,t)} \quad (1)$$

where $\Gamma(\Pi, t)$ is average loss on path Π. also, define the total reputation of the system as the sum of reputation of all the paths Π consisting of h hops and connecting source s to receiver r

$$\Lambda(t) = \sum_{\Pi \in \Pi} \Lambda(\pi, t)$$

.

4.4 Creating routing scheme from confidence levels

The previous section suggests using $\Lambda(\Pi, t) = \beta^{-\Gamma(\Pi,t)}$ as probability for a path Π. It remains to show how to pick up a path with probability proportional to $\beta^{-\Gamma(\Pi,t)}$ at time t. This is not that easy, because of exponentially many paths exists.

We solve this problem in a way similar to [21]. The algorithm is formally presented in Appendix (Fig. 16). Here, we informally sketch its operation, and show how it works via a numerical example.

Suppose we have the above edge confidences $\Lambda(e) = \beta^{-\Gamma(e)}$ as in fig. 5. (We omit time index t in this section.) Our first step is assigning these confidence levels to edges in our layered graph. Here, full confidence (1) is assigned to self-edges (see fig.6). Now, we will assign cumulative confidences to vertices. Confidence of the receiver (node r_0) in fig. 6 is 1. Confidence of the nodes at layer 1 (neighbors of the receiver s_1 and c_1) is the confidences of their direct edge to the receiver. Confidence of a node at layer 2, say, a_2, is the sum of two terms:

- the confidence of going thru s_1, which is product of confidence of edge from a_2 to s_1, times cumulative confidence of vertex s_1.
- the confidence of going thru c_1, which is product of confidence of edge from a_2 to c_1, times cumulative confidence of vertex c_1.

This process continues, as indicated in fig. 6.

The final routing is very simple: source (s_4) observes its neighbors, s_3 and a_3, and checks their cumulative confidences, which are 0.41 and 1.041. It then splits its decisions on its outgoing edges proportionally to the cumulative confidence levels. That is, it decides to go thru a_3 with probability $1.041/(1.041 + 0.41)$. Namely, packet arriving at a node proceeds to an outgoing edge with probability equal to the fraction of the total confidence of that node, contributed by the paths going that outgoing edge. With remaining probability we route to s_3 (which actually means staying in the same physical node). If we do go to a_3, we choose to continue to go to b_2 with probability 0.9/1.041, etc.

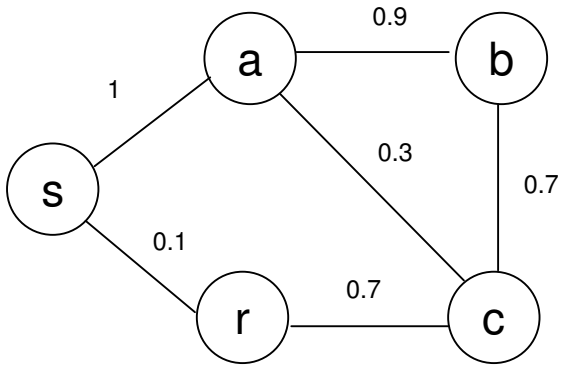

Figure 5: Confidence on each edge .

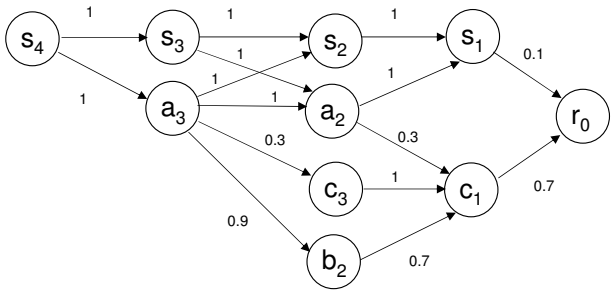

Figure 6: Confidence on each edge in the layered graph.

.

4.5 Statement of competitive bounds

The following theorem is a corollary of [12], [21]. (Similar result was obtained in [9].)

Theorem 4.1 *Consider an algorithm that picks each path* Π *at time* t *with probability*

$$\frac{\Lambda(\Pi, t)}{\Lambda(t)} \sim \beta^{-\Gamma(\pi,t)} \quad (2)$$

Consider any *adversarial sequence of fault patterns on edges, that is potentially changed adaptively based on*

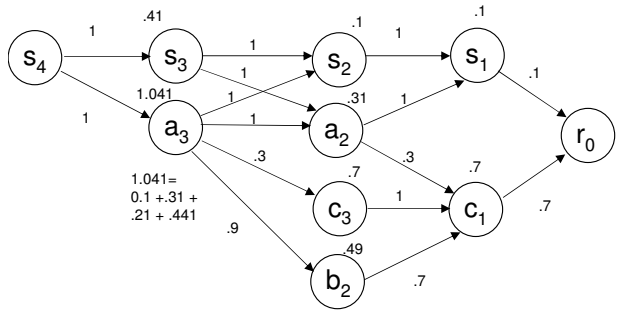

Figure 7: Confidence on each edge in the layered graph.
.

algorithms decisions, under the constraint that the average loss of some other optimal solution is at most ρ.

Then, the loss exhibited by the above strategy exceeds optimal loss ρ by an additive factor of at most

$$\frac{H \cdot \log E}{\epsilon \cdot T} + \epsilon$$

This is accomplished by choosing

$$\beta = \frac{1}{(1-\epsilon)^T} \quad with \quad \epsilon = \sqrt{\frac{4H \log n}{T \cdot \rho}}$$

We can make the following observations:

- error is increasing linearly with square root of the number of hops H on the optimal path
- error decreasing with the square root of the time window T
- length of window T required to arbitrarily approach optimal rate ρ is inversely proportional to ρ. There is no chance to get close to small ρ with a small window.

4.6 Practical Enhancements

One problem with using multiple paths is that subsequent packets from the same flow can be sent through different paths causing out-of-order arrivals at the receiver. Out of order arrivals have adverse effects both for real-time and non-real time applications using TCP since they can cause multiple duplicate acknowledgements to be sent back to the sender. To avoid this problem we hash packets into buckets that have "length" proportional to the reputation of each of the next hop neighbors. That way packets from the same flow will be consistently sent towards the same neighbor while the percentage of packets sent to each of the neighbors will follow the percentages calculated by our routing algorithm.

5 Experimental Evaluation

We use a locally written, packet-level, event based simulator to evaluate our routing protocol. The simulator assumes shortest delay routing between any two nodes. The simulator models the propagation delay of physical links but does not model queuing delay. This was done to make our simulations more scalable. There are two kinds of nodes in the simulator: *network nodes* and *overlay nodes*. The underlying network nodes provide the network connectivity and simulate packet losses by drop packets according to two different modes: **A. random** where a node drops each incoming packet with a specified probability and **B. pulse** where a node drops *all* incoming packets during it's OFF interval while it allows all packets to pass through during it's ON interval. This second model tries to match the findings from Internet path losses we mentioned in Section 2.2.

Overlay nodes implement two overlay routing algorithms: Our algorithm described in Section 3 and a skeleton version of RON that takes only loss measurements. Both algorithms send a probe every 1 sec and use a sliding window W of the last 100 samples to calculate the links' error rate. Every 20 seconds, each node floods the network with link state advertisements as well as recalculating its routing table. When a packet arrives at an overlay node, the node looks up the packet's destination in it's overlay forwarding table. If that forwarding table contains no entry then the packet is forwarding along the underlying network path towards its destination, otherwise the packet is sent to the overlay node returned by the forwarding table lookup. If the forwarding table lookup returns as result the ID of the node the packet is currently on then we reduce the packet's hop count and do another lookup (but this time with reduced with the hop count reduced by one).

The data sources that we implemented, simulated UDP sources sending packets at a constant rate. We are able to measure the number of sents packets that were received at the destination and use the ratio of received packets to evaluate the performance of the two algorithms.

Using this simulator we can show how the different algorithms parameters control the performance of our algorithm and also how our algorithm compares to RON in the cases of random drops as well as drops generated by malicious adversaries. We have used both synthetic

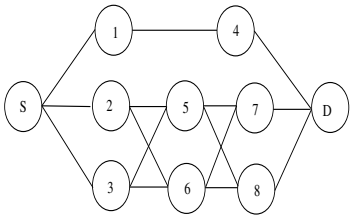

Figure 8: Multi Hop Topology

and realistic topologies in our simulation to highlight the contribution of different parameters to the algorithms performance and to show their performance under (somewhat) realistic conditions.

5.1 The effect of β

As we explained in Section 3, our algorithm uses the parameter β to calculate the *reputation* of an edge $\Lambda = \beta^e$ where $e = d/W$ is the link's error rate, calculated as the number of dropped probes d over the size of the sliding window W. As Fig. 1 shows, large values of β translate to very low linke reputations for relatively low link error rates. As a result, given the option between two paths, one having slightly higher error rate than the other one our algorithm will have a strong negative bias against the path with higher loss. The question is though, if our algorithm will pick as a next hop a node that leads to *multiple* moderateley lossy paths over a next hop that leads to a single higher quality path. The rationale behind this question is that a node's weight is the sum of all the paths that go through that node and so a node that leads to multiple paths might look as attractive as a node that leads to a single path.

To answer this question we have used the topology shown in Fig. 8. In this network node S is the source and node D the data sink connected by the other nodes in the network. When sending packets towards D node S has three options: going to node $1, 2$ or 3. Node 1 has only a single path to the destination while nodes 2 and 3 have 4 paths to D. Each link connecting adjacent overlay nodes randomly drops packets with varying probability. The most reliable links are $S \rightarrow 1$, $1 \rightarrow 4$ and $4 \rightarrow D$ that drop 5%, 4% and 3% of all packets respectively. The worst links are the ones connecting nodes $5, 6, 7$ and 8 with drop rates ranging between 20% to 30%. The justification for picking these numbers was that nodes 2 and 4 had each four paths but they were "four times worse" that 1's path.

Figure 9 shows the percentage of packets received at D for different values of β. When β is small then a larger portion of packets is lost since our algorithm

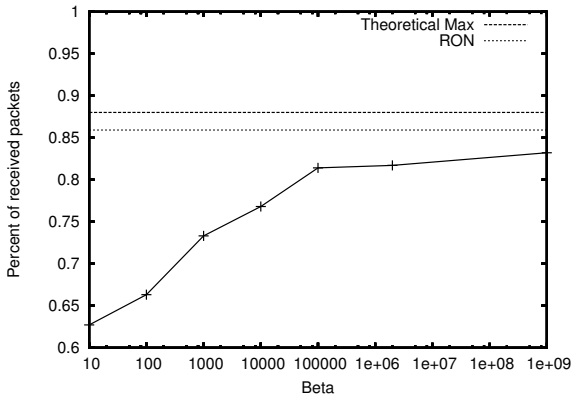

Figure 9: Effect of Beta

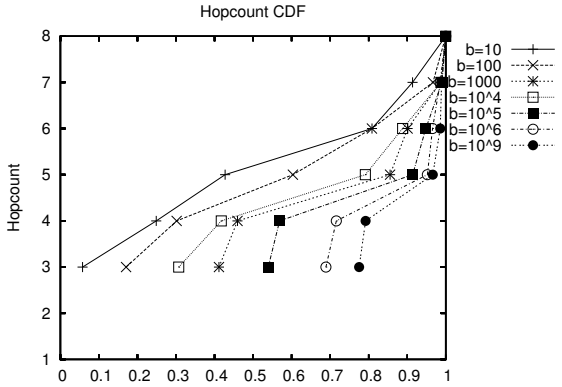

Figure 10: Hopcount Distribution

sends more packets down the lossy paths. However as the value of β increases our algorithm reaches the theoretical maximum (equal to $(1 - 0.05) * (1 - 0.04) * (1 - 0.03) = 0.88$) showing that most of the packets are routed through the high-quality path over node 1. When $\beta = 10^9$ our algorithm achieves 95% of the optimal value. For comparison, RON successfully sends 85% of the packets to the destination.

Another potential concern regarding our algorithm has to do with the maximum number of hops h and whether this hopcount forces the packets to take a longer path through the network. The concern is that a packet might "bounce" inside the network until it "consumes" it's allocated number of hops before finally reaching the destination. To investigate the validity of this claim, we used the topology shown in Fig. 8, and logged the accumulated hop counts of the packets arriving to D.

Figure 10 shows the cumulative density function (CDF) of the packet hopcount's for different values of

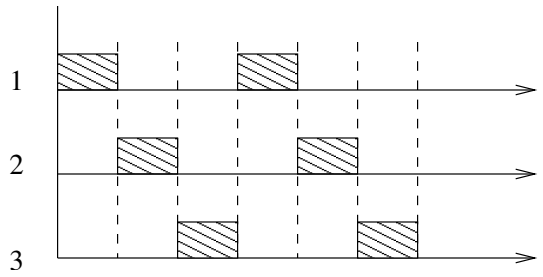

Figure 11: Adversarial strategy

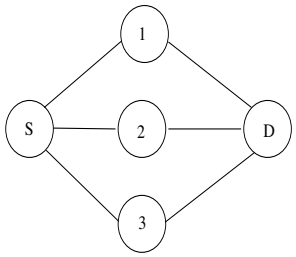

Figure 12: 3 Ark Topology

β. Again we notice that when β has small values, then packets arrive to D using longer paths. Packets with hop count larger than 4 have traveled a path through the network that includes a loop. After closer investigation of the results we discovered that most of these paths are cases where node 1 sends a packet to node 2 or 3 which subsequently send the packet back to 1 since sending the packet back to the origin is better than forwarding it to towards D since the origin has paths high higher *reputation*. However, as the value of β increases the vast majority of the packets (¿ 80%) have a hop count of three which means that they traveled though the $(S \rightarrow 1 \rightarrow 4 \rightarrow D)$ path.

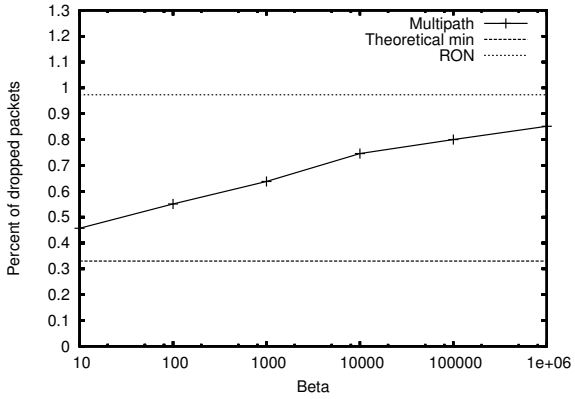

Figure 13: Packet Loss on "3-ark" network

5.2 Performance against malicious adversaries

Next, we evaluate the performance of the two overlay routing algorithms against *malicious adversaries* that potentially know the algorithm used by the overlay nodes and have the capability of creating losses on every path of the network.

We start by describing the strategy of the adversary. This strategy is based on the knowledge that RON uses the average error rate over the past W (=100) samples to determine which path to use over the next period and each period is 20 seconds long. Using this knowledge the adversaries create a pattern where knowledge of the past has *negative* correlation to the future. In other words, the path that has been the best in the past (and consequently is going to be the one that RON will pick) is going to be the worst in the future.

Figure 11 shows a loss pattern that has this exact effect for the topology shown in Fig. 12. Let's assume that RON starts by picking path 1 during the first interval. During that interval all packets are lost and then during the next interval RON will pick the second interval[1] again suffering total loss. At this point, the third path looks ideal since during the last two intervals all probes were successfull. So, RON will pick the third path only to incur more lost packets. At this point the cycle repeats and all the packets get dropped. Note that in this case, the best strategy, which is the one that consistently sends packets down the same path, should get 2/3 of the packets successfully delivered to the destination.

Figure 13 shows the results of simulating the scenario we just described. The line labeled "RON" shows the percentage of dropped packets for RON. As one can see, for the case of RON almost all (actually 97.3%) packets are dropped. The line at the bottom of the graph corresponds to the best case where only 1/3 of the packets are dropped. The same graph shows the performance of our algorithm for varying values of β. In this case, small values of β are actually *better* since the algorithm does not severely penalize past bad behavior and thereby spreading packets across the three paths almost evenly.

At this point the reader might think that this adverse effect is specific to the particular topology or the attack pattern used. In order to show that this is not

[1]Since we assume that the adversary has total knowledge of the algorithm, it knows what interval RON will pick up next

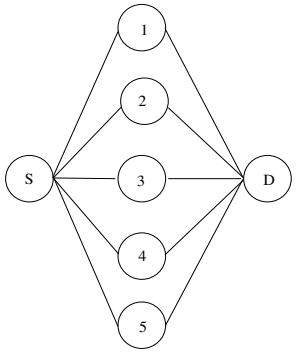

Figure 14: 5 Ark Topology

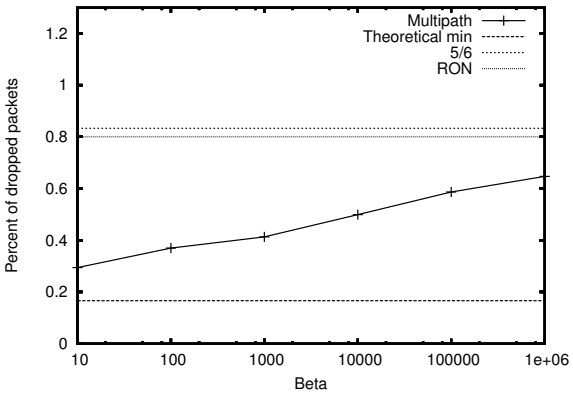

Figure 15: Packet Loss on "5-ark" network

true, we repeated our experiment but this time using the topology shown in Fig. 14. The attack pattern used in this case is one where the attacker drops all packets during one interval and then stays quiet for the subsequent five intervals. Adversaries again align themselves in a "ladder" so loss interval from adversaries happen at successive intervals. Note that in this case there is one interval where no adversary is dropping packets and so at least 1/6 of the packets should be delivered. On the other hand, the best strategy will deliver 5/6 of the packets by consistently staying on the same path.

Figure 15 shows the results in this case. Once again, we see that RON is very close to the worst possible strategy (where 5/6 of the packets are dropped) while our algorithm comes very close to the optimum for small values of β.

Finally, to remove any doubts that these effects are not applicable to real topologies but are rather an artifact of the specific artificial topologies we simulated the same overlay network as the one mentioned in the deployment of RON [1]. For this scenario the adversary has control over some of the nodes on a network "cut" separating the senders from the receivers.

Table 1 the results from this simulation. The first row correspond to the case where the adversary randomly drops packet with probability between 20% and 50%. The column titled "Static" corresponds to an overlay routing algorithm that simply uses the best underlying network path. In the case of random drops we see that both RON and our algorithm are able to find the least error-prone path and route packets through that path effectively doubling the percent of packets successfully received. For the second row, we replaced the adversary with one that uses "pulses" like the ones shown in Fig. 11. In this case, the adversary has ON and OFF periods of 20 seconds, that is it drops all the packets in one period and then it stays quiet for the next period. We made no effort to "align" the loss patterns but the results show that RON is adversely affected in this case while our algorithm is very close to the optimal strategy (which has a loss probability of 50%).

Scenario	Multipath	RON	Static
Random	0.834	0.816	0.382
Pulse	0.44	0.18	0.49

Table 1: Percent of successfully received packets

The results in this paragraph point us a *tradeoff* in the choice of β. Small values of β are bad for random loss patterns since they don't penalize bad paths but are good for adversarial attacks such as those we described in this section. We are currently investigating how the value of β can be dynamically adjusted.

6 Related Work

RON [1] is a well-known overlay network used to improve delivery characteristics over the Internet. The difference between RON and our work, is that RON was designed to overcome path outages not caused by adversaries. On the other hand, our work is able to protect the overlay network participants against both random as well as adversarial losses. Dynabone [23] which is the follow-on to the X-Bone [22] project, mentions using adaptivity to protect against active attackers but offers no description of these adaptive protocols. Our algorithm has sound theoretical background and our experimental results verify it's performance in practice.

The algorithmic results that we are basing our work upon are based on a computational learning framework [13, 6]. 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 [13, 6]; these solutions have an exponential computational overhead. Polynomial computational overhead schemes were recently suggested by [21, 9]. Our work is relying on these recent theoretical advances.

Our work is related to the Work on "Swarm Intelligence" , described in [14, 4, 3].

This work is very relevant for reactive routing protocols, rather than pro-active routing considered in this paper. The difference is that our work used global link state approach, rather than exploration by individual agents as in aforementioned work. Thus, our work will feature faster convergence.

7 Summary

Overlay networks such as RON[1] have the potential of masking the effects of Internet path outages by routing around network faults. We presented in this paper a new algorithm that is similar to RON in the sense that nodes probe links to evaluate their reliability and disseminate the links' *reputation* to the other overlay nodes. Based on the links' reputations, the network nodes compute their routing tables so that packets travel down paths that have the highest reputation.

The novel aspect of our algorithm is that is *tunable* so it's more or less sensitive to changes to the error rates on different network links. Theoretical work proves that, with proper tuning, this type of algorithms are not vulnerable to DoS attacks. Theory also shows that in extreme case, of our dynamic algorithms turn into "greedy routing algorithms, such as RON, which are vulnerable to adaptive DoS attacks.

We have experimentally validated these algorithms via simulations, which indeed demonstrate theoretical vulnerabilities of greedy algorithms such as RON, as well as show our algorithm does not have these vulnerabilities.

As future direction, we will be working on extending this work to other types of attacks, in addition to DoS attacks considered in this paper. Specifically, we will also consider various Byzantine attacks and various security mechanisms to prevent them.

Appendix A

Below, we formally present operation of the algorithm (see Fig. 16) which is computing outgoing probabilities from confidences, which was informally described in Section 4.4.

Initialization
$\bar{\phi}^0(v) \leftarrow 0 \ \ \forall \ \ v \neq r$
$\bar{\phi}^0(r) \leftarrow 1$
repeat $i = 0$ to $i = h$
$\phi^i(v,u) = \bar{\phi}^i(u) \cdot \Lambda(v,u)$
$\phi^{i+1}(u) = \sum_{(u,v)\in E} \phi^i(v,u)$

Path Generation
$i \leftarrow h$
$v_i \leftarrow s$
repeat h times
select $v_{i-1} \leftarrow u$ with probab. $\frac{\phi^{i-1}(v_i,u)}{\phi^{i-1}(v_i)}$
$i \leftarrow i - 1$

Figure 16: Links state version of the algorithm. The above algorithm is performed at the source.

Appendix B - Comparison with stochastic model

If the failure were totally randomly distributed and independent, then in fact we would be interpreting $\Gamma(e,t)$ as the estimate of fault probability, and $\mu(e,t) = 1-\Gamma(e,t)$ as estimate of success probability. Under this assumption, the algorithm such as RON [1] try to pick path maximizing

$$\mu(\Pi,t) = \prod_{e_i\in\Pi} \mu(e_i,t) = \prod_{e_i\in\Pi} (1-\Gamma(e_i,t)) \qquad (3)$$

$$= \quad 1 - \sum_{e_i\in\Pi} \Gamma(e_i,t) + \delta = 1 - \Gamma(\Pi,t) + \delta \qquad (4)$$

Where term δ is the second-order cross-probabilities term, which is generally small for low probabilities. In this case, *maximizing* Eqn. 4 above means picking path with total *minimal* fault rate $\Gamma(\Pi,t)$. This is accomplishing by picking paths 1 with probability $\beta^{-\Gamma(\Pi,t)}$, provided that β is large enough. (Remember that β needs to grow with duration of window T.)

Notice that if we choose $\alpha = 1/\beta$, then for small $\Gamma(e_i)$, we have

$$\alpha^{\Gamma(e_i)} \approx 1 - (1-\alpha)\cdot\Gamma(e_i) \approx 1 - \alpha\Gamma(e_i)$$

Thus, our algorithm will be acting very similarly in this case to existing work such as RON [1].

References

[1] David Andersen, Hari Balakrishnan, Frans Kaashoek, and Robert Morris. Resilient overlay networks. In *Proceedings of the18th ACM SOSP*, Banff, Canada, 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, Dave Holmer, and Herb Rubens. Swarm intelligence and secure routing, 2003. unpublished manuscript.

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

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

[6] Nicolò Cesa-Bianchi, Yoav Freund, David P. Helmbold, David Haussler, Robert E. Schapire, and Manfred K. Warmuth. How to use expert advice. In *Proceedings of STOC 93*, pages 382–391, 1993. To appear, *Journal of the Association for Computing Machinery*.

[7] B. Chandra, M. Dahlin, L. Gao, and A. Nayate. End-to-End WAN Service Availability. In *Proceedings of 3rd USISTS*, March 2001.

[8] John Jannotti, David K. Giford, Kirk L. Johnson, M. Frans Kaashoek, and Jr. James W. O'Toole. Overcast: Reliable Multicasting with an Overlay Network. In *Proceeding of OSDI*, October 2000.

[9] Adam Kalai and Santosh Vempala. Geometric algorithms for online optimization, 2003. unpublished manuscript.

[10] C. Labovitz, A. Ahuja, and F. Jahanian. Delayed Internet Convergence. In *Proceedings of SIGCOMM 2000*, August 2000.

[11] C. Labovitz, C. Malan, and F. Jahanian. Internet Routing Instability. *IEEE/ACM Transactions on Networking*, 5(6):515–526, 1998.

[12] N. Littlestone and M. Warmuth. The weighted majority algorithm. *Information and Computation*, 108(2):212–260, 1994.

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

[14] 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.

[15] D. Moore, G. Voelker, and S. Savage. Inferring Internet Denial of Service Activity. In *USENIX Security Symposium*, August 2001.

[16] M. Lee O. Hussein, T. Saadawi. Ant routing algorithm for mobile ad-hoc networks(arama). In *Procedings of Annual ARL CTA Symposium*, 2003.

[17] Vern Paxson. End-to-End Packet Dynamics. *IEEE/ACM Transactions on Networking*, 7(3):277–292, 1999.

[18] Stefan Savage, Tom Anderson, Amit Aggarwal, David Becker, Neal Cardwell, Andy Collins, Eric Hoffman, John Snell, Amin Vahdat, Geoff Voelker, and John Zahorjan. Detour: a case for informed internet routing and transport. *IEEE Micro*, January 1999.

[19] Stefan Savage, Andy Collins, Eric Hoffman, John Snell, and Thomas Anderson. The End-to-End Effects of Internet Path Selection. In *Proceedings of SIGCOMM 1999*, August 1999.

[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.

[22] J. Touch and S. Hotz. The x-bone. In *Third Global Internet Mini-Conference at Globecom '98*, November 1998.

[23] Joe Touch. Dynabone. http://www.isi.edu/dynabone/.

[24] Y. Zhang, N. Duffield, V. Paxson, and S. Shenker. On the Constancy of Internet Path Properties. In *Proceedings ACM SIGCOMM Internet Measurement Workshop*, November 2001.