The Traveling Miser Takes Detours

Reuven Cohen, Sarit Kraus, Ariella Richardson, Yuval Shavitt

***Abstract*— Several important management applications, e.g., billing, generate a relatively high volume of traffic which is not delay sensitive. Both the management and the user traffic use the same network infrastructure. However, the management traffic does not yield any immediate benefits for the users, and is thus an overhead from their point of view.**

In a previous work it was suggested to store such traffic inside the network in case of congestion, e.g., by using the power of active networks to look for the least expensive place one can backtrack to. In this paper, we suggest instead to use active networks in order to deflect the time insensitive management traffic from congested areas. To optimize the deflection we use the recent discovery about the power law characteristics of the Internet, and show that this knowledge improves our deflection performance. We further show that by applying learning techniques on locally gathered deflection information performance improves.

I. Introduction

Several important management applications, e.g., large scale monitoring for the purposes of accounting, planning, performance evaluation *etc.*, generate a relatively high volume of traffic. Both the management and the user traffic use the same network infrastructure. However, the management traffic does not yield any immediate benefits for the users, and is thus an overhead from their point of view.

Much of the network management traffic has no strict time constraints. Examples include various logging facilities that subsequently transfer the accumulated data over the network, software distribution facilities, distributed backups, accounting and billing, long term monitoring of the large-scale systems *etc.* These applications have usually a considerable flexibility of when actually to use the network and thus maybe directed to possibly slower routes. Furthermore, since the end-users are not directly effected by the management services described above, their impact on the user-visible network services should be minimized.

Brietgand *et al.* [3] suggested to differentiate between the higher priority user-visible traffic and the lower-priority management traffic. It is important to stress, however, that the terms high-priority and low-priority refer both to best effort traffic with different timing constraints. The user traffic, say HTTP packets, has to arrive at its destination within a short period of time (2-3 seconds for the HTTP example) while the low-priority management traffic can be delayed much longer. It is important to note that both the high-priority and the low-priority traffic compete for the same limited amount of network resources.

Brietgand *et al.* further suggested the *traveling miser problem* whose objective is to improve the *goodput* of the network by preferring the high priority (user) traffic over the low priority (management) traffic at times of high load. In other words, if there are plenty of resources in the network there is no difference between the low and high priority traffic. However, when the resources become scarce the user traffic is given priority over the management traffic.

The traveling miser (TM) problem can be modeled (as we do in Section III) as a routing problem in a graph where edge weights are dynamically changing. In the simplest case edges can be either UP or DOWN. If the edge dynamics is heterogeneous, i.e., edges have significantly different probability distribution, one can use learning techniques to be able to avoid risky edges and route through alternative ones. This has been done in the case of QoS routing (e.g., [16], [17], [2]) where the number of edge 'sampling' is fairly large. In the TM problem one cannot expect many attempts to deliver management data and, thus, the number of sampling for each edge is relatively small. We see the development of learning algorithms that are based on limited sampling as a major challenge.

Thus, Brietgand *et al.* [3] proposed solving the TM problem by attempting to recover from deadlock, rather than trying to avoid risky edges using historical data. Their algorithmic solutions were based on storing the management traffic in case of high load. When high

Reuven Cohen is with the Department of Computer Science and Applied Mathematics, Weizmann Institute of Science, Rehovot, Israel. Email: r.cohen@weizmann.ac.il

Sarit Kraus is with the Dept. of Computer Science, Bar Ilan University, Ramat Gan, Israel. Email: sarit@macs.biu.ac.il

Ariella Richardson is with the Dept. of Computer Science, Bar Ilan University, Ramat Gan, Israel. Email: richara@macs.biu.ac.il

Yuval Shavitt is with the Department of Electrical Engineering — Systems, Tel-Aviv University, Tel-Aviv, Israel. Email: shavitt@eng.tau.ac.il Fax: +972 3 640 5027, Tel: +972 3 640 8659

load was detected, their algorithm looked for the least expensive location to store the bulk of management traffic until the load will abate.

In this work we take a different approach that also attempts to recover from deadlock without use of historical data. We suggest to avoid storing very large amount of data inside the network, and instead to divert the management burst away from congested links. Such a solution fits well in the framework of active networks [18] which allows packets to use user defined flow-specific routing algorithms. The ability to use local deflection routing to lessen congestion on links was demonstrated by Kornblum *et al.* [13], where they suggested to divert portion of the traffic along local one or two hop alternative less loaded paths. Here we use global detours instead of local deflections, which enables us to divert the low priority traffic along longer routes, and potentially avert wide congested areas.

Our algorithms allow the low priority burst to use the shortest route to the destination, as long as no congestion is encountered. Upon hitting a congested link it attempts to divert the packet to an intermediate node, from which it will continue to the destination. Selecting such a node randomly is a difficult task without global view of the network. To overcome this difficulty, we use our knowledge that the network has a small number of nodes with very high connectivity [11], [6], [20]. Their high degree makes these nodes excellent candidates for detour mid-point, and the fact that they tend to reside away from the network edges reduces the expected detour length. It is natural that the short list of the top high degree nodes will be maintained in the network nodes. Selecting the midpoint target node among the high priority nodes becomes thus an easy task much easier than selecting a random node. Our simulation also show that this approach yields better success ratio – overhead tradeoff than the latter.

Combining learning techniques with the basic deflection algorithm further improves performance. We compared different learning approaches: saving either information about successful paths, or about attempts to find paths that failed. We found that avoiding paths that failed leads to better results than using paths that had been successful in the past.

II. Related Work on QoS Routing

In the following we describe some QoS routing techniques that can be adapted to solve the TM problems.

Routing protocols define the route that a packet takes when traveling through the network. When working with QoS requirements the existing protocols may not suffice. Traditional protocols do not take into account the constraints that are introduced by QoS traffic. Some of the QoS constraints are dependent on the dynamic state of the network and this is not taken into consideration in the traditional protocols.

SPR: Single Path routing protocols are designed mainly for best-effort data traffic. The path chosen to connect a source to a destination is the unicast path between them. This is usually the shortest path in term of hops. In our case, if the probability of the low priority traffic to be blocked is high, SPR will perform miserably.

MPR: Multi path routing protocols, such as spanning joins [5] and QoSMIC [22], search for a path in different directions and then select the best candidate. In Spanning Joins a source node broadcasts within a certain neighborhood a join request. When the destination receives the request it replies with a message to the source node. In case that the destination is a multicast tree, multiple replies can be received, and the source node chooses the best candidate from the responses. This process is repeated iteratively with larger neighborhoods until a path is found. In this case the broadcasted message will have the same QoS restrictions as the traffic that we intend to send along the path, therefore if the destination receives the message we know that we have a candidate path. In the case of the unicast scenario Spanning Joins is the same as the SPR protocol but with a much higher overhead, therefore we did not simulate it. QoSMIC, which has many advantages over spanning joins in a multicast scenario, was not simulated in this work since for unicast it is the same as the spanning-joins protocol.

QMRP: QoS Aware Multicast Protocol [8] is a protocol that combines SPR with MPR. The search for the path is conducted initially with an SPR protocol. On each link we check that the necessary resources defined for the low-priority traffic are met. Once the resources needed are not available at a specific point the protocol changes to the MPR mode. From this stage multiple candidate paths are searched, and then the best of the candidate paths is chosen.

III. Network Model

We model the network as a Graph, $G(V, E)$ where the existence of an edge $(u, v) \in E \subseteq V \times V$ means that the edge in the opposite direction, (v, u), exists as well. Each edge $e = (u, v)$ is associated with a cost, $c(e)$. However two edges connecting two nodes in the opposite direction may have different cost values. We model congestion on an edge as a binary state, an edge might be blocked or free. When an edge is blocked its cost is infinite, when it is free the cost is one. If a unidirectional edge is blocked it is still capable of carrying high priority data and control packets of the law priority traffic.

IV. Motivation

We want to search for a route that allows the low priority traffic to avoid congested areas, but there is no obligation to pick the fastest route. Namely, in cases of congestion packets should make a detour. Different bursts of packets can take different routes and this way there are less chances of congestion since the traffic is well balanced across the network. Our problem is, thus, to select potentially good route with minimal overhead.

A similar problem appears in QoS routing [7], where routes that obey some QoS criterion are searched, typically in multiple directions. However, in QoS routing the search is done in the signaling phase, where the cost of searching multiple routes is mainly in the overhead on the signaling computing capacity (clearly the bandwidth cost of a signaling packet is negligible). In our case, where the entire packet burst is temporarily stored and forwarded to a different direction, it is important to keep the overhead of this search low, and thus, copying the packets and sending them in multiple directions is too costly to be an attractive solution. Thus, our detour algorithm searches in one direction each time the burst is blocked. We are prepared to incur the cost of routing along long paths, but cannot accept splitting the data stream.

The detour can be made to a predefined node or to a random node. One of the benefits of choosing a random node is that we may be able to spread the load in different directions and avoid congestion of specific nodes (in the spirit of optimal routing in parallel machines, e.g., [15]). We decided to use an alternative approach to the pure random node selection based on our knowledge of the Internet structure. It has been discovered [11] and recently verified [20] that the degree distribution in the Internet obeys a power law, i.e., only a few nodes have a very high degree. We select one of these nodes when we are blocked and redirect the blocked packet to it. The rational for using high ranked nodes was the assumption that nodes with high rank are relatively easy to reach and easy to leave because of the high number of links connecting them. We do not have to worry too much about congestion at these high ranked nodes, even though they attract much of the traffic, because they have many incoming and outgoing links.

After establishing that choosing high ranked nodes is better we proceeded to look at the problem in a dynamic network. We improve the protocol by combining learning from data that has been accumulated over time. We assumed that after trying various paths we could use the experience we had accumulated about which directions were good and which were bad. Our first assumption was that reusing paths that had successfully lead us to destinations would be useful, and we experimented with saving these paths and then choosing them again rather than trying to find new ones. We discovered, much to our surprise, that this did not help at all, and reusing paths that had been found in the past was no better than finding new ones. We then tried to save information about paths that failed to lead us to our destination. This proved to be a much better strategy. We managed to reach the destination with a higher success rate while lowering our overhead. Finally we tried to combine both strategies, and although this improved our chances of reaching the destinations, it also had a higher overhead.

V. Protocol description

A. *The basic* detr *protocol - How to detour in a static network*

When a management burst is sent between two points, we must find a route that is not blocked to the low priority traffic. The burst starts to travel along the unicast route between the source and the destination nodes. At each node we check if the next link has the necessary resources (namely, that it is not blocked to the low priority traffic). In the case that the link is not blocked the burst traverses the link. In the case that these requirements are not met we look for a detour. The burst is routed towards an alternate temporary destination, and then later rerouted back towards the original destination.

When the burst is sent towards a detour node, it does not necessarily reach that node, it may get blocked again along the way. In this case we select a different detour node, and repeat this up to $\mathcal{B}^{in}$ times or until we finally reach the most recent detour node. Upon reaching the detour node or if we exhausted the number of attempts we head back towards the original destination. We also limit the number of times we can start such a detour to no more than $\mathcal{B}^{out}$.

As mentioned in the introduction, the interesting question answered in this paper is how to select the detour node. We study two alternatives. The detour node is selected uniformly at random either from the group of all nodes or from the group of high degree nodes.

Conceptually, one may think of the algorithm as been performed by an agent traveling in the network. This makes the discussion much easier thus we will keep this metaphor in the remaining of the paper.

A pseudo code for the algorithm appears in Fig. 1. Advance(d) is a subroutine where the agent advances toward a destination d until d is reached or the shortest path to d is blocked. The subroutine returns the node that the agent stopped at (either the destination or the block).

Algorithm 1 (FindPath(src, dst))

1. $count \leftarrow 0$;
2. $blk \leftarrow$ Advance(dst);
3. if ($blk = dst$) 'deliver data';
4. if DeadEnd(blk) 'data delivery failed';
5. while ($count < \mathcal{B}^{out}$ AND $blk \neq dst$) {
6. $count \leftarrow count + 1$
7. $dcount \leftarrow 0$;
8. while ($dcount < \mathcal{B}^{in}$ AND $blk \neq dst$) {
9. $dcount \leftarrow dcount + 1$
10. $det \leftarrow$ GetDetourNode();
11. $blk \leftarrow$ Advance(det);
12. if DeadEnd(blk) 'data delivery failed'; }
13. $blk \leftarrow$ Advance(dst); }
14. if ($blk = dst$) 'deliver data';
15. else 'data delivery failed';

Fig. 1. Pseudo-Code for the *detr* algorithm

Algorithm 2 (UpdateGoodPath(P,dst))

1. for all $n_i \in P, i \neq |P|$ {
2. GoodTable$[n_i][dst] \leftarrow n_{i+1}$ }

Algorithm 3 (AdvanceGood($curr$,dst,P,idx))

1. while($curr \neq dst$) {
2. if(GoodTable[$curr$][dst] $\neq NULL$) {
3. $next \leftarrow$ GoodTable[$curr$][dst]; }
4. else {
5. $next \leftarrow$ routingTable[$curr$][dst]; }
6. if(linkBlocked($curr, next$)) {
7. return '$curr$' }
8. else {
9. $curr \leftarrow next$;
10. $P[idx] \leftarrow next$;
11. $idx \leftarrow idx + 1$; }

Fig. 2. Routines for the algorithm with learning of the good paths

DeadEnd(d) is a subroutine that returns TRUE when there are no links with available resources leading out of the node d. The algorithm uses two tunable constants: $\mathcal{B}^{out}$ which is the maximum number of allowed detours per journey, and $\mathcal{B}^{in}$ which is the maximum number of allowed blockings within a single detour.

B. The detr *protocol with learning in a dynamic network*

Once we established that it is better to detour to a high degree node rather than to a random node (see section VII) we attempted to improve the protocol by adding learning. We save information about paths that have been traveled, and use this information when making the routing decision.

Three different approaches were tested with the detour protocol. One was to save the information about good routes and to reuse them. The second was to save information about bad routes in order to avoid them, and the third to combine both.

The addition of learning to the basic algorithm changed it in two ways. First, we added functions that saved the data about paths that we used UpdateData() (either good or bad paths). The second difference is in how we choose the next step to take at each node - changes to Advance(). Rather than simply looking up the routing table for the next step, we use the data that we saved in the past.

1) Saving good routes: The motivation for saving good routes was to use routes that had already proven successful rather than searching for new ones. We defined a routing table named **GoodTable** for saving the good paths. **GoodTable** has the same structure as the regular routing table (see Section V-A), but it is only partially filled. Most entries are valued NULL specifying that we have no information about a better path than the default path. When a successful path is found it is saved in the GoodTable. For the traveled path $P(n_1 = source, n_2, ..., n_k = dest)$ the update function is given in Fig. 2 (top). When we search for paths we use the GoodTable where information is available rather than searching for new detours (see Fig. 2 (buttom)).

The Pseudo-Code of *detr* algorithm with learning of good data follows appears in figure 3.

2) Avoiding bad routes: In this version we want to use information about bad paths that we collected over time. Bad paths are paths on which we failed to reach our destination. The data is collected in a table named **BadTable**, and has the same structure as the routing table. When we failed to deliver the data after trying to travel along a path $P(n_1 = source, n_2, ..., n_k \neq dst)$, where the link between n_{k-1} to n_k was blocked we update the BadTable. When we encounter a path that was found to be bad, we detour instead of using the bad path. Figure 4 shows a pseudo code for handling the bad path data. The Pseudo-Code of *detr* algorithm with learning of bad data is given in Fig. 5.

As we show in Section VII-B.2, this version proved to be better than using 'good data' and improved our ability of successfully reach the destination in comparison to the version without learning, while lowering the cost of the search.

The algorithm where both good and bad pathes are stored is a combination of the two described above and was thus omitted. It can be found in [19].

Algorithm 4 (FindPathLG(src, dst))

1. $count \leftarrow 0$;
2. $P \leftarrow NULL$
3. $idx \leftarrow 0$
4. $blk \leftarrow$ AdvanceGood($curr$,dst,P,idx);
5. if ($blk = dst$) {
6. 'deliver data';
7. updateGoodPath(P,dst); }
8. if DeadEnd(blk)'data delivery failed';
9. while ($count < \mathcal{B}^{out}$ AND $blk \neq dst$) {
10. $count \leftarrow count + 1$;
11. $dcount \leftarrow 0$;
12. while($dcount < \mathcal{B}^{in}$ AND $blk \neq det$) {
13. $dcount \leftarrow dcount + 1$;
14. $det \leftarrow$ GetDetourNode();
15. $blk \leftarrow$ AdvanceGood($curr$,dst,P,idx);
16. if DeadEnd(blk) 'data delivery failed'; }
17. if($blk \neq dst$) {
18. $blk \leftarrow$ AdvanceGood($curr$,dst,P,idx); } }
19. if($blk = dst$) {
20. 'deliver data';
21. updateGoodPath(P,dst); }
22. else 'data delivery failed';

Fig. 3. The algorithm with learning of the good paths

Algorithm 5 (UpdateBadPath(P,dst))

1. for all $n_i \in P, i \neq |P|$ {
2. BadTable$[n_i][dst] \leftarrow n_{i+1}$ }

Algorithm 6 (AdvanceBad($curr$,dst,P,idx))

1. $next \leftarrow$ routingTable[$curr$][dst];
2. while(linkNotBlocked($curr, next$) AND $curr \neq dst$) {
3. if (BadTable[$curr$][dst] = next) {
4. return '$curr$'; }
5. $curr \leftarrow next$;
6. $P[idx] \leftarrow next$;
7. $idx \leftarrow idx + 1$;
8. $next \leftarrow$ routingTable[$curr$][dst];
   }
9. return '$curr$';

Fig. 4. Routines for the algorithm with learning of the bad paths

Algorithm 7 (FindPathLB(src, dst))

1. $count \leftarrow 0$;
2. $P \leftarrow NULL$
3. $idx \leftarrow 0$
4. $blk \leftarrow$ AdvanceBad($curr$,dst,P,idx);
5. if ($blk = dst$) 'deliver data';
6. if DeadEnd(blk) {
7. 'data delivery failed';
8. updateBadPath(P,dst); }
9. while ($count < \mathcal{B}^{out}$ AND $blk \neq dst$) {
10. $count \leftarrow count + 1$;
11. $dcount \leftarrow 0$;
12. while($dcount < \mathcal{B}^{in}$ AND $blk \neq det$) {
13. $dcount \leftarrow dcount + 1$;
14. $det \leftarrow$ GetDetourNode();
15. $blk \leftarrow$ AdvanceBad($curr$,dst,P,idx);
16. if DeadEnd(blk) {
17. 'data delivery failed';
18. updateBadPath(P,dst); } }
19. if ($blk \neq dst$) {
20. $blk \leftarrow$ AdvanceBad($curr$,dst,P,idx); } }
21. if($blk = dst$) 'deliver data';
22. else {
23. 'data delivery failed';
24. updateBadPath(P,dst); }

Fig. 5. The algorithm with learning of the bad paths

VI. A Simulation Study

A. *Simulating* detr

The performance of the detour algorithm was tested by simulation. Since there is no prior work on detour of low priority traffic we compared our algorithm with algorithms that were suggested for QoS routing. However, one must be cautious with this comparison: in QoS routing the search for a feasible route is done using signaling messages which allow at reasonable cost to split the search to multiple routes. Here, we divert a burst 'on-the-fly', and while one can divert-and-copy, namely, 'multicast' the burst to several possible routes, the overhead of such a solution deems it impractical.

The protocols we used in the comparison are: shortest path routing (SPR), QMRP [8], and spanning joins [5]. For spanning joins, we implemented its direct flooding version called directed spanning joins. We used two common performance metrics [8], success ratio and average message overhead, which are defined as follows:

$$\text{success ratio} = \frac{\text{number of routes found}}{\text{number of routes searched}} \qquad (1)$$

$$\text{avg msg overhead} = \frac{\text{links traveled by messages}}{\text{number of routes searched}} \quad (2)$$

In our simulations we varied the link success probabilities, which is the probability that a link is unblocked. We assumed this probability is uncorrelated between links. This was simulated by choosing a certain percentage of the links, and randomly setting their values to 'blocked'.

We used Power-Law topologies that are based on the results reported in [11], [20]. These results shows that the node degrees in the internet obey a power log law: most nodes have small degrees and a small number of nodes have high degrees. As the degree increases the number of nodes with that degree decreases polynomially. We used the Inet topology generator [12] that was shown to reasonably mimic the Internet power law.

We ran the simulation on networks of 600 nodes. We tested 6 different networks. For each network we tested 6000 source-destination pairs. The source and destination nodes were selected randomly from the list of nodes. We checked the confidence intervals of our results using a confidence level of 95 percent, and found that they were the size of the markers we used on the graphs, thus omitted the error bars from the graphs.

There is a natural tradeoff between the success rate and the overhead. A high aggressiveness parameter leads to high overhead, but also to higher success rates, therefore we looked for the optimal values. Two strategies were tested for choosing the detour node. One was choosing a high ranked node, and the other was choosing a random node.

B. *Simulating* detr *with learning*

This part of the simulation was performed in order to determine how it is best to perform the learning: what data is best to save and how to use it. We used the same simulation settings as before and ran the simulation with different learning strategies, saving information about either good paths, or bad paths, or both. While the information about good routes is used in order to use them, the information about bad routes is used in order to avoid them, therefor information about bad routes causes a detour.

We also studied how the learning is affected by network dynamics over time, which raises a question about the duration data should be saved. In the dynamic network scenario an initial set of costs is provided for the edges, and these costs change over time. This provides us with a dynamic network where edges become blocked and unblocked over time. The decision of which edges to change is based on a Zipf distribution, $P_i \sim i^{-a}$ with the exponent $'a'$ for a node of rank i. We used several values of a, the default parameter used was $a = 0.8$. namely most of the links do not change very often and a small number of the links change their state often. The changing network is implemented by first setting the initial state of the network, and then changing some of the links in each iteration. Each iteration is a search for a path between a new set of src and dest.

There are three parameters that were varied in the simulation: a, the Zipf distribution parameter; 'itr'; and 'dyn'. 'itr' is the number of iterations that the data we saved is valid for. 'dyn' refers to the number of links that can be changed in each iteration, and thus strongly effects the network dynamics. We ran the simulation for 12000 iterations, each iteration is a selection of a (src,dest) pair and attempt to find a path between them.

VII. Results

A. detr *vs. other algorithms*

We simulated QMRP with a maximum branching level[1] of 6 and with maximum branching degrees[2] of 10 or 1.

We first compared the use of high-ranked nodes with the use of random nodes as our detour nodes (see Fig. 6 and Fig. 7). The simulations show, as we expected, that using high-ranked nodes is better. The success rate while using high-ranked nodes was higher than when detouring to a random node. In addition, the overhead for high-ranked nodes was lower than for detours to random nodes. Note that both have better success ratio than QMRP which looks for local detours, while the overhead of detour with high degree nodes selection is similar or lower than QMRP (see detailed discussion below).

We also simulated different definitions of a high-ranked node: we tried defining a high-ranked node as a node with more than 5, 10, 15, or 50 links. We found that 5, 10, and 15 all gave similar success rates; 50 gave lower success rates (see Fig. 8). The reason for this is that there are not enough nodes with degree 50 or higher that can be reached with high probability. When the high ranked nodes are not reached the detour becomes similar to a detour using random nodes.

Another set of parameters that was evaluated were the aggressiveness parameter, $\mathcal{B}^{out}$, and the detour length, $\mathcal{B}^{in}$. For this set of simulations we set the high rank

[1]If too many nodes enter the branching mode the overhead will be much too large. Thus, we limit the number of branches allowed over the route by the maximum branching level.

[2]The other parameter that can cause a very high overhead is the number of adjacent links a node has. When this number is very high QMRP sends out a large number of messages. The maximum branching degree defines the maximum number of adjacent links we send messages to. For our detour protocol we tried random or high degree detour nodes.

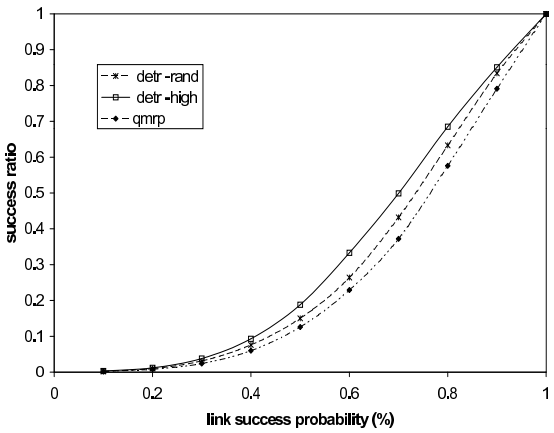

Fig. 6. Success ratio for detour using high-ranked and random nodes in detours

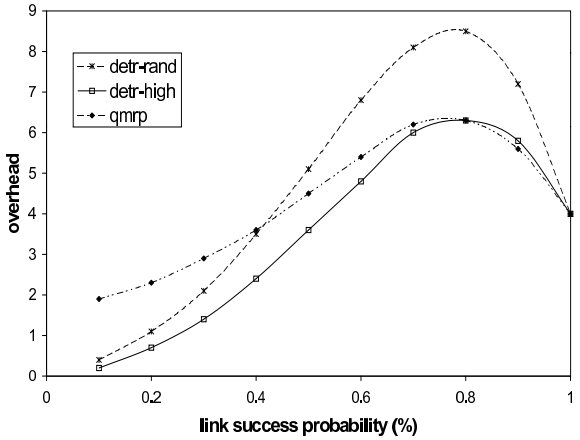

Fig. 7. Overhead for detour using high-ranked and random nodes in detours

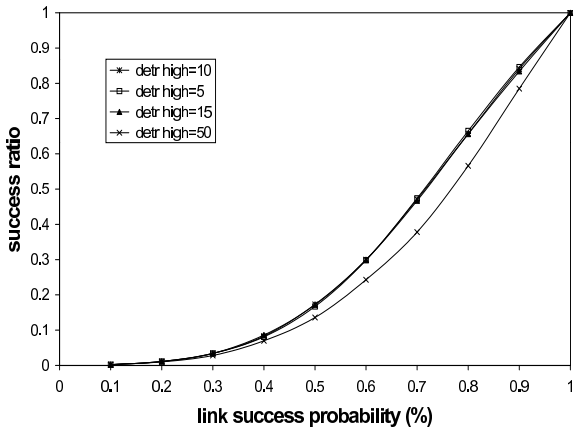

Fig. 8. Success ratios for different high rank definitions

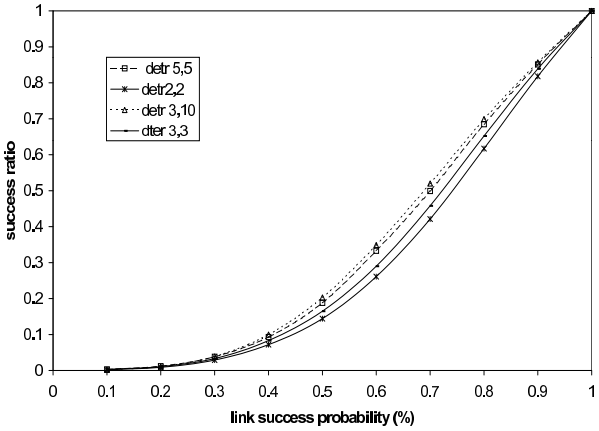

Fig. 9. Success ratios for various ($\mathcal{B}^{in}$, $\mathcal{B}^{out}$) parameter combinations.

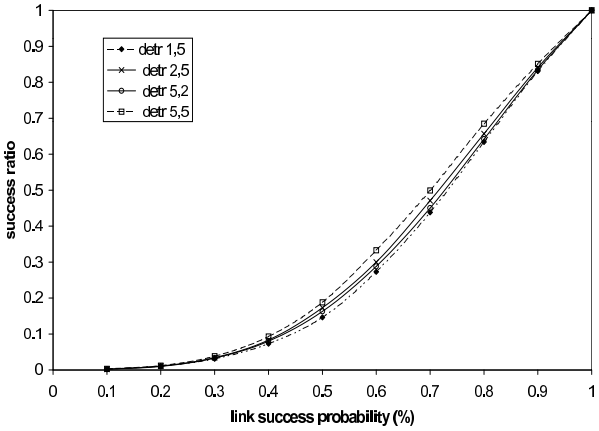

Fig. 10. Success ratios for various ($\mathcal{B}^{in}$, $\mathcal{B}^{out}$) parameter combinations.

node definition to a node whose degree is 10 or higher. We found that changing $\mathcal{B}^{in}$ and $\mathcal{B}^{out}$ in the range 1-5 demonstrated the tradeoff between overhead and success rate. We can achieve a higher success rate if we pay with a higher overhead (see Fig. 11). Note that ($\mathcal{B}^{in}$, $\mathcal{B}^{out}$)=(2,5) is better than (5,2). In other words when more effort was placed in the outer loop of trying to reach the destination we had better results, see Fig. 9 and Fig. 10 for various parameter combinations.

We compared our protocol to other protocols. The detour protocol proved to have higher success rate than SPR. When comparing with QMRP the following conclusions were obtained: We compared the detour protocol to QMRP with a branching degree of 1. This is a fair comparison because we do not allow more than one branch to be performed at one node in both algorithms. In this case we found that our algorithm has a higher success rate and a lower overhead than QMRP. This means that when we are satisfied with the success rate that can be obtained by the detour protocol (higher

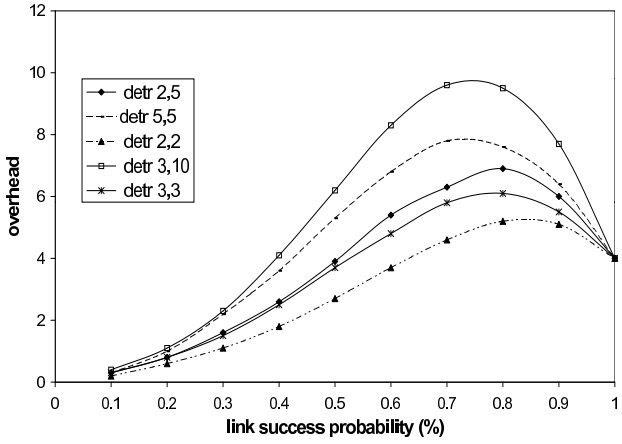

Fig. 11. Overhead for various ($\mathcal{B}^{in}$, $\mathcal{B}^{out}$) parameter combinations.

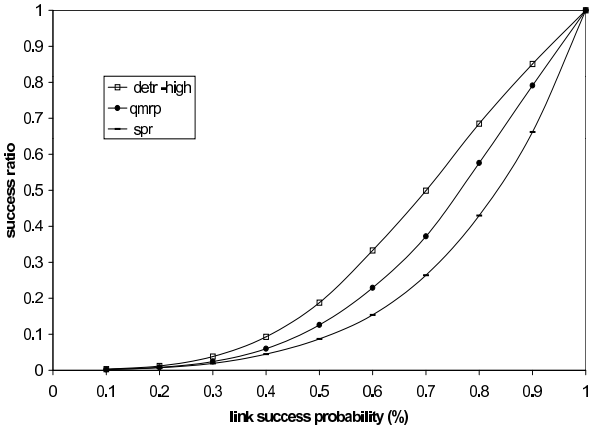

Fig. 12. Success ratio for detour vs. the other algorithms (QMRP with branching degree of 1)

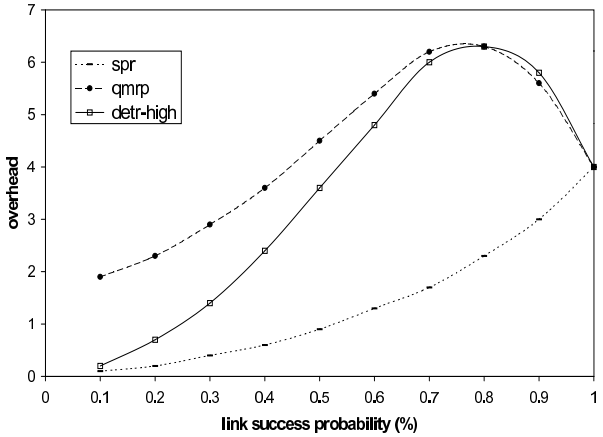

Fig. 13. Overhead for detour vs. other algorithms (QMRP with branching degree of 1)

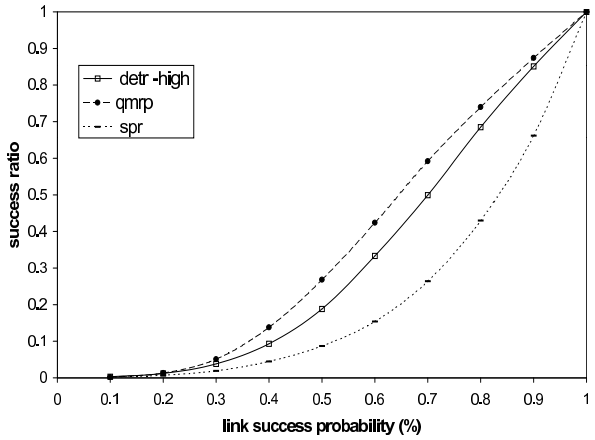

Fig. 14. Success ratio for detour vs. other algorithms (QMRP with branching degree of 10)

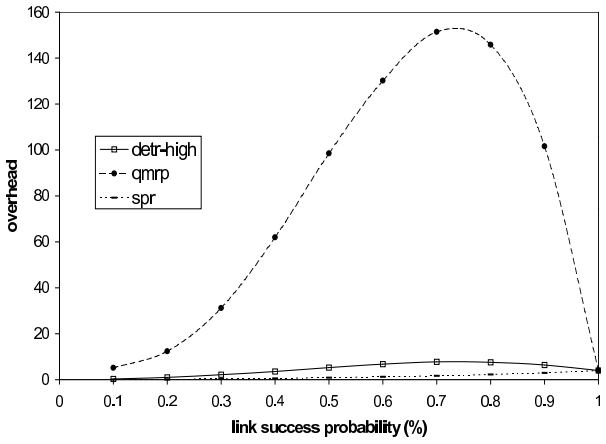

Fig. 15. Overhead for detour vs. other algorithms (QMRP with branching degree of 10)

than SPR) we can do it with an overhead that is slightly lower than that of QMRP (see Fig. 12 and Fig. 13). We must note that if we are prepared to pay the price of an ever higher overhead we can use QMRP with a branching degree of 10 and obtain a higher success rate (see Fig. 14 and Fig. 15) but as we already mentioned this is not practical for our problem. This means that for QoS routing it is not clear which protocol, QMRP or an adaptation of our algorithm, is better.

B. detr *with learning*

In the simulations of the *detr* protocol with learning we first establish what we need to learn: bad or good routes. After establishing that the 'bad' version was better we proceeded to the fine tuning of the different parameters of the algorithm. In the end we checked a few more variations of the algorithm in order to understand better why it works the way it does.

1) Results of 'good' learning: We first saved the data about good paths and then reused these paths

Fig. 16. Success ratio for *memory-detr* saving **good** paths vs. *detr* (in a static network)

Fig. 17. Overhead for *memory-detr* saving **good** paths vs. *detr* (in a static network)

Fig. 18. Success ratio for *memory-detr* saving **bad** paths vs. *detr* in a static network (itr=3000)

Fig. 19. Overhead for *memory-detr* saving **bad** paths vs. *detr* in a static network (itr=3000)

again. Results showed, surprisingly, that saving good data improved success only marginally, but it increased the overhead substantially (see Fig. 16 and Fig. 17). The reason that reusing the good paths does not help is that some of them end up to be very long paths that contain loops, these paths are not worth reusing. Removing the loops from paths that we find does not help since the paths remain very long.

2) Results of 'bad' learning: When we saved data about the bad paths, in order to avoid them, the results improved substantially. The success ratio is higher than without using the learning strategy, namely avoiding previously known bad routes helps. What makes the results even better is the fact that we manage to reduce the message overhead in comparison to the the basic *detr* version. This is because the high overhead is generated from walking along paths that lead us nowhere. We ran this simulation both in a static network (Fig. 18 and Fig. 19) and a dynamic (see Fig. 20) and Fig. 21) network, where the link weights can change, and found that for both the success ratio was higher with learning, and the overhead lower.

3) Results of combining 'good' and 'bad' learning: The last strategy that was used was combining information on both successful and unsuccessful paths. This scheme success ratio (Fig. 22) was higher than the other strategies, but the overhead (Fig. 23) was also higher. This happens because the overhead when we use information of good paths is very high.

VIII. Analytical Results

We aim to show that choosing a high degree node for the detour is a superior strategy to choosing a random node. The problem of node breakdown in a network is a percolation problem [9], [4]. The detour will succeed if both routes from the source to the detour node and from that node to the destination are not blocked as a result of the breakdowns.

The probabilities of both routes not being blocked are correlated and also depend upon the fact that the original

Fig. 20. Success ratio for *memory-detr* saving **bad** paths vs. regular *detr*. (dyn=200)

Fig. 21. Overhead for *memory-detr* saving **bad** paths vs. regular *detr* (dyn=200)

Fig. 22. Success ratio for *memory-detr* saving both bad and good data

Fig. 23. Overhead for *memory-detr* saving both bad and good data

route is blocked. In order to simplify the problem we will present the proof of a weaker theorem which ignores edge weights and correlations.

The model we use for the network is the random configuration model [1]: For each node, i, choose a degree, k_i, where the number of nodes having a degree k is given by the degree sequence $n(k)$ (which can be Poisson, Power-Law, or any other). Now create a list containing k_i copies of each node i, and then choose a random matching on this list. Each pair of matched nodes are connected through an edge. Under certain conditions (in particular, a restriction on the upper cutoff of the sequence) this model can be shown to lead to each network with a given degree sequence with the same probability [1].

We will begin with the following technical Lemma:

Lemma 8.1: If a series, $P(l)$ satisfies $\sum_{l=1}^{L} P(l) \geq 0$ for all $L \geq 1$, then it also satisfies $\sum_{l=1}^{L} P(l)q^l \geq 0$ for all $0 \leq q \leq 1$.

Proof: We prove the Lemma by induction over L. For $L = 1$ it is clear that if $P_1 \geq 0$ then $P_1 q \geq 0$. Assume now that the induction hypothesis is correct for $L = L_0$, i.e. for all P_l satisfying $\sum_{l=1}^{L_0} P(l) \geq 0$, $\sum_{l=1}^{L_0} P(l)q^l \geq 0$. If $P(L_0 + 1) \geq 0$ clearly $\sum_{l=1}^{L_0+1} P(l)q^l \geq 0$. If $P(L_0+1) < 0$ define the following sequence $Q(l) = P(l)$ for all $l < L_0$ and $Q(L_0) = P(L_0) + P(L_0+1)$. Now, since $P(L_0+1) < 0$, $P(L_0 + 1)q^{L_0+1} \geq P(L_0 + 1)q^{L_0}$. Therefore, $\sum_{l=1}^{L_0+1} P(l)q^l \geq \sum_{l=1}^{L_0+1} Q(l)q^l$. But, by the induction hypothesis $Q(l)$, which is of length L_0 satisfies $\sum_{l=1}^{L_0+1} Q(l)q^l \geq 0$. Thus, $P(l)$ also satisfies this property. ■

We now turn to show that the probability of success increases with the degree of the detour node. Notice that in the proof we assume that the graph is connected with high probability. This is true for Barabasi-Albert networks, and also for the configuration model if the

minimum node degree is at least 3 [14]. The proof is also possible when the minimum degree is lower, with some technical difficulties stemming from the conditioning on the paths remaining connected.

Theorem 1: In a randomly connected network, when links break down with probability p, the probability of the original shortest path between nodes a and b not to be blocked is an increasing function of the degree of node a, k_a.

Proof: The probability for a path of length l to be intact is q^l, where $q \equiv 1-p$ is the probability that a single link is functional.

Suppose now that there exist two nodes a_1 and a_2, having degrees k_1 and k_2 and distances from node b l_1 and l_2, respectively. If $l_1 = l_2$ then $q^{l_1} = q^{l_2}$ and no difference in the probability of path functionality exists.

Suppose now that $l_1 \neq l_2$. Assuming (without loss of generality) that $l_1 < l_2$, and that the shortest path between a_1 and b is achieved through a_1's neighbor c. Since the matching of edges was random, any switching between two links gives an equally probable graph. Therefore, the ratio between graphs where a_1 is a neighbor of c and $d(a_1,b) = l_1 < d(a_2,b)$ and between cases where c is a_2's neighbor and $d(a_1,b) > l_1 = d(a_2,b)$ is k_1/k_2. Similar arguments apply when the minimum distance is achieved through more than one neighbor, and, in fact then a higher ratio is achieved.

Thus, if $k_1 > k_2$, it follows that $P(d(a_1,b) < l) > P(d(a_2,b) < l)$ for any l and b. By applying Lemma 8.1 to $P(d(a_1,b) < l) - P(d(a_2,b) < l)$ it follows that $\sum_l P(d(a_1,b) = l)q^l > \sum_l P(d(a_2,b) = l)q^l$ ¿From which follows that $E(q^{d(a,b)})$ is an increasing function of a's degree, as asserted. ■

A recent paper [21] studies the behavior of distances between nodes in scale free networks. The results proven there may be used to estimate the probability of success of the suggested scheme, and the dependence of this probability on the degree of the chosen detour node. The main result of [21] (see also [10]) states that the expected distance, d between two randomly chosen nodes in a scale free network, with degreequence $n(k) \sim k^{-\tau}$ is with high probability $D = 2\left\lfloor \frac{\log\log N}{\log(\tau-2)} \right\rfloor + c$, for some (random) constant c. Therefore, the probability for a single random detour to be available is $p = q^{2\left\lfloor \frac{\log\log N}{\log(\tau-2)} \right\rfloor + c}$. As can be seen, this changes very slowly with N, and is expected to give a reasonable probability of success even for very large networks.

To estimate the dependence of the success probability on the degree we first prove the following Lemma.

Lemma 8.2: Let v_1 and v_2 be two nodes in a graph, and let B_1 and B_2 be two non intersecting balls around nodes v_1 and v_2 respectively. If the number of edges emanating from both balls is equal, then given a node v_3, outside the balls, its probability to be at a distance l of the balls satisfies $P[d(B_1,v_3) = l] = P[d(B_2,v_3) = l]$ for every l.

Proof: For every configuration where the distances are $d(v_3,B_1) = l_1$ and $d(v_3,B_2) = l_2$ there exists an equaly likely configuration with the connections of the edges emanating of B_1 swapped with the connections of the edges emanating from B_2 where the distances are switched. The Lemma follows. ■

We now turn to estimate the dependence of the probability of the detour success on the degree. According to [21], the asymptotic behavior of the number of nodes, Z_d at a distance d from a random node is $\log Z_d \approx c(\tau-2)^{-d}$ for $d < D/2$, and some c (depending on the details of the degree sequence). For a high degree node with degree k, the number of edges emanating from a ball of radius 0 around it is approximately the same as the number of edges emanating from a ball of radius d around a random node, where $\log k = c(\tau-2)^{-d}$, or $d = \frac{\log\log k - \log c}{|\log(\tau-2)|}$. Therefore, the ratio between the probability of success for a random site and for a site of high degree k is $P_{\text{rand}}/P_k = q^d = q^{\frac{\log\log k - \log c}{|\log(\tau-2)|}}$. As an estimation of the probability of success for a single detour, assuming high degree and independence of the paths between the source and the detour and the destination and the detour (which should be viewed as an approximation) it is expected that the probability of success will be proportional to $q^{-2\frac{\log\log k}{|\log(\tau-2)|}}$ (where the squaring is due to the effect on both the source and destination routes).

IX. Conclusion and Future Work

We have shown that by using deflection routing with recent knowledge regarding the network structure we reduce the overhead of management traffic. In particular we showed that deflecting the traffic from congested areas gives better results than adapting QMRP to this scenario. We also showed that using learning techniques for this problem improved the results. We tested our learning strategies on a dynamic network and found that avoiding bad paths is a better strategy than re-using good ones. We found that if we can afford to pay a higher overhead combing both strategies lead us to a better success ratio.

References

[1] B. Bollobás. A probabilistic proof of an asymptotic formula for the number of labelled regular graphs. *Europ. J. Combinatorics*, 1:311–316, 1980.

[2] Justin A. Boyan and Michael L. Littman. Packet routing in dynamically changing networks: A reinforcement learning approach. *Advances in Neural Information Processing Systems*, 6:671–678, 1994.

[3] David Breitgand, Danny Raz, and Yuval Shavitt. The travelling miser problem. In *IEEE Infocom 2002*, New-York, NY, USA, April 2002.

[4] D. S. Callaway, M. E. J. Newman, S. H. Strogatz, and D. J. Watts. Network robustness and fragility: percolation on random graphs. *Phys. Rev. Lett.*, 85:5468 – 5471, 2000.

[5] K. Carlberg and J. Crowcroft. Building Shared Trees Using a One-to-Many Joining Mechanism. *Computer Communication Review*, pages 5–11, January 1997.

[6] Q. Chen, H. Chang, R. Govindan, S. Jamin, S. Shenker, and W. Willinger. The origin of power-laws in internet topologies revisited. In *IEEE Infocom 2002*, New-York, NY, USA, April 2002.

[7] Shigang Chen and Klara Nahrstedt. An Overview of Quality-of-Service Routing for the Next Generation High-Speed Networks: Problems and Solutions. *IEEE Network, Special Issue on Transmission and Distribution of Digital Video*, 12(6):64–79, Nov./Dec. 1998.

[8] Shigang Chen, Klara Nahrstedt, and Yuval Shavitt. A QoS-aware multicast routing protocol. *IEEE Journal on Selected Areas in Communications*, 18(12):2580–2592, December 2000.

[9] Reuven Cohen, Keren Erez, Daniel ben Avraham, and Shlomo Havlin. Resilience of the internet to random breakdowns. *Phys. Rev. Lett.*, 85:4626 – 4628, 2000.

[10] Reuven Cohen and Shlomo Havlin. Scale free networks are ultrasmall. *Phys. Rev. Lett.*, 90:058701, 2003.

[11] Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. On power-law relationships of the internet topology. In *ACM SIGCOMM 1999*, Boston, MA, USA, August/September 1999.

[12] C. Jin, Q. Chen, and S. Jamin. Inet: Internet topology generator. Technical Report CSE-TR-433-00, University of Michigan, EECS dept., 2000. http://topology.eecs.umich.edu.

[13] Jessica Kornblum, Danny Raz, and Yuval Shavitt. The active process interaction with its environment. *Computer Networks*, 36(1):21–34, June 2001.

[14] Milena Mihail, C. Gkantsidis, and Amin Saberi. Conductance and congestion in power law graphs. In *SIGMETRICS 2003*, 2003.

[15] Lata Narayanan. Randomized algorithms on the mesh. In *IPPS/SPDP Workshops*, pages 408–417, 1998.

[16] Srihari Nelakuditi, Zhi-Li Zhang, and Rose P. Tsang. Adaptive proportional routing: A localized qos routing approach. In *INFOCOM 2000*, pages 1566–1575, 2000.

[17] Leonid Peshkin and Virginia Savova. Reinforcement learning for adaptive routing. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, 2002.

[18] Konstantinos Psounis. Active networks: Applications, security, safety, and architectures. *IEEE Communications Surveys*, 2(1), First Quarter 1999. `http://www.comsoc.org/pubs/surveys/1q99issue/psounis.html`.

[19] Ariela Richardson. Heuristic routing in comunication networks. Master's thesis, Bar-Ilan University, Ramat-Gan, Israel, June 2004.

[20] Yuval Shavitt and Eran Shir. DIMES: Let the internet measure itself. *ACM Computer Communications Review*, October 2005.

[21] Remco van der Hofstad, Gerard Hooghiemstra, and Dmitri Znamenski. Distances in random graphs with finite mean and infinite variance degrees. Technical Report math.PR/0502581, arXiv.org, 2005. http://arxiv.org/abs/math.PR/0502581.

[22] S. Yan, M. Faloutsos, and A. Banerjea. QoS-aware multicast routing for the Internet: The design and evaluation of QoSMIC. *IEEE/ACM Transactions on Networking*, 10(1):54–66, February 2002.