A New Approach to Routing With Dynamic Metrics

Johnny Chen Peter Druschel Devika Subramanian
Department of Computer Science
Rice University
Houston, TX 77005
Email: {jc,druschel,devika}@cs.rice.edu
Phone: (713) 527-4664, Fax: (713) 285-5930

Abstract—

We present a new routing algorithm to compute paths within a network using dynamic link metrics. Dynamic link metrics are cost metrics that depend on a link's dynamic state, e.g., the congestion on the link. Our algorithm is destination-initiated: a destination initiates a global path computation to itself using dynamic link metrics. All other destinations that do not initiate this dynamic metric computation use paths that are calculated and maintained by a traditional routing algorithm using static link metrics. Analysis of Internet packet traces show that a high percentage of network traffic is destined for a small number of networks. Because our algorithm is destination-initiated, it achieves maximum performance at minimum cost when it only computes dynamic metric paths to these selected "hot" destination networks. This selective approach to route recomputation reduces many of the problems (principally route oscillations) associated with calculating all routes simultaneously. We compare the routing efficiency and end-to-end performance of our algorithm against those of traditional algorithms using dynamic link metrics. The results of our experiments show that our algorithm can provide higher network performance at a significantly lower routing cost under conditions that arise in real networks. The effectiveness of the algorithm stems from the independent, time-staggered recomputation of important paths using dynamic metrics, allowing for splits in congested traffic that cannot be made by traditional routing algorithms.

I. Introduction

Network routing algorithms used today calculate least cost (shortest) paths between nodes. The cost of a path is the sum of the cost of all links on that path. The cost of a link can be static (a function of the link) or dynamic (a function of the link and its dynamic state). The benefit of routing using dynamic cost metrics is that paths can be recomputed based on prevailing traffic patterns to reduce congestion, packet delay, and packet drops.

Khanna and Zinky [11] showed that the performance, measured in packet drops, of a routing algorithm using a particular class of dynamic metrics on the ARPANET, significantly outperformed the same routing algorithm using static cost metrics. This dynamic cost metric, called the *revised ARPANET routing metric*, also reduced routing instabilities and oscillations compared to the previous metric based purely on link delays. Despite their positive results, dynamic metrics are not widely used in today's Internet. The technical reasons are that 1) the amount of routing updates with dynamic metrics are hard to control *a priori* because routing updates are dependent on network traffic, and 2) routing using dynamic metrics can cause routing oscillations, though Khanna and Zinky's dynamic metric lessens the problem.

In this paper, we present a new routing algorithm that uses dynamic metrics and that to a large extent, overcomes the limitations cited above. The algorithm uses Scout, a destination-initiated shortest path computation technique that we have developed [5]. A destination node using the Scout algorithm initiates a path computation from every node in the network to itself at periodic intervals. The period between route recomputations is controlled by the initiating node. The routing overhead of the algorithm is a function of this period and is independent of the dynamic conditions in the network. Therefore, Scout has the property that its routing overhead is predictable and under the control of the initiating nodes in the network.

Since each initiating node takes charge of its route recomputations, the updating of routes to each initiating node is uncorrelated. In contrast, traditional algorithms like Link State (LS) and Distance Vector (DV) that use dynamic metrics recompute paths between all nodes in the network quasi-simultaneously. This simultaneous computation has the unfortunate tendency of shifting *congestion causing traffic* from a currently congested area to a currently uncongested one which then becomes the next congested area, triggering route recomputations over again. Because route recomputations in Scout are independently controlled by the destinations, route updates to different destinations are uncorrelated and usually staggered in time, thus allowing congested traffic to be split, significantly reducing route oscillations caused by shifting all traffic in a congested zone at the same time.

However, this flexibility and autonomy in route recomputation comes at a price. Scout is not as efficient as algorithms like the DV at computing all-pairs shortest paths. This is because Scout destinations independently compute paths to themselves and do not share path computations with other nodes.

To obtain the benefits of Scout while keeping the routing overheads low, we integrate Scout with LS and DV routing algorithms. The three main ideas of this hybrid are as follows: First, Scout is used only to recalculate routes based on dynamic link costs. Second, Scout updates paths only to a small number of "hot" destination nodes in the network. Analysis of Internet packet traces show traffic locality patterns that support this choice: almost 90% of all Internet traffic is destined to 10% of the networks, and the top 1% destinations receive more than 50% of all network traffic, both in bytes and packets (see Figure 1). Third, the LS or DV component of the hybrid uses static cost metrics to recalculate routes in response to topology changing events (e.g., link failure and recovery), as well as routes to non-hot destinations.

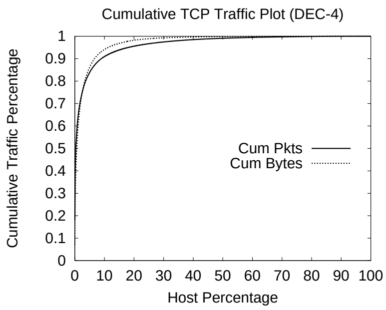


Fig. 1. Traffic locality in dec-pkt-4 TCP packet traces from [1]. The graph shows the cumulative distribution of network traffic to destinations. The trace contained an hour's worth of wide-area traffic from DEC's corporate intranet to the rest of the Internet. The destination locality in this trace is typical of other traces we analyzed.

By using Scout with dynamic metrics to improve path quality to important destinations and relying on traditional algorithms with static cost metrics to maintain other routes, the hybrid algorithm reaps the benefits of routing with dynamic metrics without paying a high, unpredictable routing overhead and without incurring significant routing instabilities.

The rest of the paper is organized as follows. In Section II, we present Scout, a selective route recomputation algorithm that takes advantage of traffic locality in wide-area networks. In the same section, the integration of Scout with the Distance Vector routing algorithm is described. We present experimental data showing the behavior of these algorithms in Section III, discuss related work in Section IV, and conclude in Section V.

II. Our Approach

One of the key ideas of our approach is that we compute routes based on dynamic metrics only for a minority of destinations, namely destinations that receive a majority of the traffic. Due to the high destination locality observed in Internet traffic, being able to calculate dynamic metric paths to these *hot destinations* has several advantages. First, we hope that calculating paths to a few destinations is cheaper than calculating paths to all destinations. Second, not calculating shortest paths to all destinations simultaneously reduces the route heredity problem [11] and therefore reduces route oscillation. In addition, because these destinations receive a high percentage of routing traffic (say 50+%), rerouting these paths enables our algorithm to shift enough traffic to alleviate areas of congestion.

Our solution to dynamic metric routing is based on two key concepts. First, we developed a routing algorithm, Scout, that is able to selectively update paths to individual destinations. The second key idea is to integrate Scout with traditional DV and LS algorithms. This hybrid provides base static routes to a majority of destinations using DV/LS and provides dynamic metric routes to selected destinations using Scout.

Scout is a host-initiated, selective route calculation algorithm that calculates least cost paths to individual nodes (or networks). By host-initiated, we mean that the path calculation process to a node (or network) is initiated by that node (or gateway router to the network). Scout is selective because only paths to that destination are affected, and it is efficient at computing routes to individual destinations. However, Scout does not aggregate path computation, and therefore is not as efficient as LS and DV when computing paths to all nodes. This deficiency motivated us to integrate Scout with LS and DV.

The hybrid of Scout with DV and LS routing algorithms uses the traditional routing algorithm to calculate paths between nodes using static metrics, while Scout continually refines paths to selected nodes using dynamic metrics.

The remainder of this section is organized as follows. We first give a brief description of the basic Scout algorithm and show how it computes least cost paths between nodes. We then describe the hybrid algorithm that integrates Scout with the Distance Vector algorithm. We conclude by summarizing the benefits of this integration.

A. The Scout Algorithm

In the Scout routing algorithm, each router maintains a forwarding table indicating for each destination, the next-hop neighbor on the known least cost path to that destination.

Scout's basic mechanism of route computation is message flooding. With Scout, destinations periodically flood a *Scout* message throughout the network. Let R be the node initiating the Scout message. The period between two consecutive floodings of Scout messages from R is called the *broadcast interval* (BI). A Scout $[R, C_R, x]$ contains the originating node's address, R, and the cost C_R to reach R, initially zero, and an increasing sequence number x. When a node P receives a Scout message $[R, C_R, x]$ from its neighbor Q, P first checks whether the sequence number x is valid. If not, the Scout is discarded. Otherwise, P modifies the Scout's cost C_R to include the cost of sending a message from P to Q, $C'_R = C_R + Cost(P \rightarrow Q)$. C'_R represents the cost a message will take if it is sent by P via Q to R.

In the first broadcast interval (i.e., when P has no record of receiving a Scout from R), P forwards the Scout [R,C'_R, x] immediately after receiving the first Scout message, initiated by R, to all neighbors except Q, the neighbor which sent P the Scout. Node P might receive more of R's Scouts in the same BI, indicating different paths and path costs to R. P remembers only the least cost Scout to R and adjusts its forwarding table to reflect the best known path, but P does not forward these Scout messages. For each node, the next-hop to R is always the neighbor that provided the least cost Scout in the current BI. In the next and subsequent broadcast intervals of R, P waits to receive a Scout message from its *designated neighbor* before flooding. We define node P's *designated neighbor* to node R as the neighbor of P that provided the least cost Scout to R in the previous broadcast interval. When P receives the

Scout from this designated neighbor, P forwards the least cost Scout received in the current BI (i.e., with the current sequence number) to R to all neighbors except the one from which the least cost Scout was received[1].

In the presence of node or link failures, a node P might never receive a Scout from its designated neighbor because the designated neighbor may no longer be connected. We account for this by requiring P to flood the first Scout message from R if P did not flood any of R's Scout messages in the previous BI.

We do not allow P to wait for a neighbor in the current BI, in particular, the designated neighbor, because if there are multiple failures, waiting for the best information might cause cascading waits.

There are five distinctive features of the Scout routing algorithm. First, the number of Scouts forwarded in one BI is $O(L)$, where L is the number of links[2]. Second, the worst-case number of BIs needed for Scout to converge is proportional to the longest least cost path (with respect to the originating node). However, this worst case requires pathological conditions. In practice, we have simulated Scout on networks with more than 100 nodes and have not observed cases where convergence required more than three BIs. Moreover, after a link failure, an alternate (not necessarily least-cost) path is guaranteed to be computed after at most two BIs, if one exists.

Third, Scouts are small and fixed size, and can therefore be piggybacked on a hop-by-hop basis onto data packets, largely defraying their cost to the network. Fourth, Scout's route updates are completely controlled by the Scout generating node and not by network changes; therefore Scout routing costs are easily controlled. Fifth, Scouts require very little router computation. This is in contrast to LS and DV where an update may affect entire forwarding tables. Detailed proofs of correctness and convergence are in our technical report [5].

B. Scout – Distance Vector

In this section, we describe the integration of the Scout algorithm with the Distance Vector (DV) routing algorithm. We first review the basic DV algorithm and then describe the hybrid algorithm.

B.1 The Distance Vector Algorithm

Distance Vector is a distributed adaptive algorithm that computes shortest paths between all pairs of nodes in a dynamic network. It is based on a centralized method known as the Bellman-Ford [2], [8] algorithm, and is the basis of many practical routing algorithms in current use [12], [1], [14].

Routers running DV compute a forwarding table indicating the best path to all known destinations. Each entry in a DV forwarding table contains three elements: the destination address, the next-hop neighbor on the known least-cost path to the destination, and the cost of the known least-cost path to the destination. Each router constructs its forwarding tables by exchanging path information with their neighbors in the form of *distance vector packets* (DVP). DVP's are excerpts of a router's forwarding table and indicate the shortest path the router has to other destinations.

Routers periodically exchange DVP's with all their neighbors or in response link/node failures or recoveries. It can be shown that after a bounded number of DVP exchanges following a topology or link cost change, all routers' forwarding tables will contain values reflecting the shortest paths to all destinations [12], [1], [14].

B.2 Scout-DV Hybrid

In the Scout-DV hybrid, the DV component is responsible for computing the least cost paths using static link costs (which are the base cost of each link). The algorithm also propagates link failures and recoveries and changes router forwarding tables in the usual way. The Scout algorithm is integrated with only a few small changes described below. Scout modifies the DV forwarding tables (the next-hop and cost fields) and uses dynamic link metrics.

Because the DV component of the hybrid algorithm computes static shortest paths between all destinations, the Scout component of the algorithm can take advantage of this. The only modification to Scout is the following:

- If the Scout algorithm does not have any information on the designated neighbor, the default is to use the next-hop neighbor computed by the DV algorithm as the designated neighbor.

This feature is used in two places: 1) when a node generates its first Scout message and 2) during the first Scout broadcast after a link failure/recovery (see below).

The two modifications to the DV algorithm are:

- When a router detects a link/node recovery, it exchanges forwarding tables with the router in the usual DV manner, propagating the Scout paths as if they were computed by DV. The receiving router performs the standard operations on the received DVP and recomputes the shortest paths for both Scout and non-Scout sending destinations.
- When a router detects a link/node failure, it follows the standard DV procedure and sends to all its neighbors a DVP containing infinity cost for all destinations using the failed link (including Scout sending destinations). This process nullifies all DV and Scout paths that use the failed link. Again, the standard DV recomputation mechanism will compute static metric shortest paths (if they exist) to all affected destinations.

To ensure that a straggling Scout message from the previous BI does not re-advertise a path through the failed link, the Scout-DV hybrid algorithm increments the Scout sequence counter for Scout calculating destinations that were affected by the failure. Notice this does not alter the Scout algorithm behavior in the next BI because the sequence number will be current.

[1] In general, with a dynamic network, the least cost Scout may not come from the designated neighbor, since link costs may change between broadcast intervals.

[2] It is not exactly L because it may happen that two neighboring nodes send Scouts to each other simultaneously. The actual number is between L and $2L$.

In the hybrid algorithm, the non-selected destinations obtain the same static least cost path as with the normal static metric DV algorithm. The responsiveness to link/node failure and recovery is just as efficient in the hybrid as in the traditional DV algorithm. The difference is that the selected Scout generating destinations have the paths to them constantly refined according to the dynamic changes in link costs. A correctness proof for our Scout-DV algorithm is included in our technical report [4].

C. Scout – Link State

We have also designed and implemented an integration of Scout with the Link State algorithm, called hybrid Scout-LS. Functionally, it is similar to the Scout-DV hybrid: the Link State component computes static paths between all nodes while the Scout component computes paths based on dynamic link metrics to destinations that generate Scouts. Due to space constraints, the hybrid LS-Scout algorithm is described only in our technical report [4].

D. Algorithm Comparisons

One of the major distinctions between hybrid-Scout (i.e. Scout-DV or Scout-LS) and the conventional LS and DV algorithms is that the efficiency and effectiveness of Scout depends on the number of Scout generating destinations and how much traffic those destinations receive. In contrast, the effectiveness of LS and DV with dynamic metrics depends on the degree of congestion experienced in the network. Scout's routing costs are independent of network traffic and depend only on the BI and the number of Scout generating destinations.

As stated earlier, in order for hybrid-Scout to effectively reroute congestion, the amount of traffic received by the Scout generating nodes must be "significant". In addition, in order for Scout to be efficient, the number of hot destinations must be "small". We quantify these two conditions in Section III and show that these conditions are typical of real networks.

The main features of the hybrid-Scout algorithms are:

1. Hybrid-Scout can selectively upgrade a destination's path calculation using dynamic metrics. This is done by simply having the destination generate Scout messages.
2. Hybrid-Scout reduces route oscillation when compared to LS/DV with dynamic metrics. This is accomplished by only rerouting selected destinations based on dynamic link metrics and by calculating new routes for those selected destinations in a time staggered manner. These phenomena are experimentally shown in Section III.
3. In hybrid-Scout, the desired route quality for a Scout generating node can be determined independently. That is, hybrid-Scout can easily provide different path qualities to different destinations, depending on their importance and traffic usage. This adjustment is done solely on the part of the Scout generating node by changing its Scout BI.
4. Hybrid-Scout's routing traffic is independent of network traffic. Therefore, Scout routing traffic can be easily controlled by limiting the number of Scout generating nodes and their minimal BIs. LS and DV routing costs must be bounded using link metric trigger thresholds and hold-downs [9].

In this paper, we assume that the set of "hot" destinations is statically known and configured to send Scouts. In the Internet, for instance, ISPs and popular Web sites are likely the appropriate destinations, and this set of sites changes only gradually over time. Should static configuration prove too inflexible, traffic monitoring can be used to maintain the set of hot destinations dynamically.

III. Simulation Results

In this section, we present extensive simulation results to evaluate the proposed hybrid Scout algorithms and to compare their behavior with that of the plain DV and LS routing algorithms using dynamic metrics. The purpose of our experiments is to determine whether the hybrid-Scout algorithms deliver better overall network performance at lower costs than DV and LS with dynamic metrics.

To identify the performance tradeoffs and scalability of hybrid-Scout versus dynamic metric LS and DV algorithms, we generate experimental topologies where we systematically vary the size of the network. While network topology is a key parameter determining performance of all four algorithms, we have chosen to study scaling effects in the context of a single class of topologies: viz., Internet-like topologies. We make this choice to not only simplify our experimental analysis but also to make the results relevant to present-day networks.

Since hybrid-Scout recalculates routes based on dynamic metrics only to a set of "hot" destinations (Scout generating nodes), a natural question that arises is how the routing cost varies as the number of hot destinations is increased and as the skew in the traffic distribution to these destinations is increased. Is the improvement in route quality worth the extra routing cost? We answer these questions by studying the performance of the considered algorithms under varying traffic distributions; in particular the number of hot destinations and the percentage of traffic directed to them. A related question is whether the improvement in route quality to hot destinations comes at the expense of those to non-Scout generating destinations. We call traffic destined for Scout generating nodes (hot destinations) *foreground traffic* and traffic destined for other nodes *background traffic*.

Our experimental analysis consists of studying the effects of (1) network size, (2) foreground traffic distribution, and (3) background traffic quantity on routing cost and overall route quality delivered by the different algorithms.

A. Simulation Environment

The simulation environment is based on the "x-netsim" package from the University of Arizona [3], an execution-driven, packet-level network simulator. We implemented four routing protocols in our simulation testbed: the two hybrid-Scout routing algorithms, a Distance Vector protocol similar to the Internet RIP protocol [9][3] and a Link State protocol similar to the Internet OSPF [6] routing protocol. In this section, we present results only for the hybrid DV-Scout algorithm because the behavior of the two hybrid algorithms using dynamic metrics are the same. The only difference is that the hybrid LS-Scout messages are 8 bytes larger than those used in hybrid DV-Scout.

Dynamic metric LS and DV were implemented using trigger thresholds. As a router continually updates its out-going link's costs, it observes whether the difference between any of its link's current cost and the link's cost that was last advertised exceeds the threshold. If so, the router initiates a network wide shortest path recomputation. Trigger thresholds are expressed as *trigger percentages*. A trigger percentage of 50% means that a link's cost has to increase/decrease at least 50% of its dynamic range in order to trigger a routing update. The lower the trigger percentage, the more sensitive the algorithm is to link cost changes, the higher the path quality, but also the higher the routing cost (traffic).

The LS and DV protocols are optimized such that route changes are propagated only in response to a triggered update. In the hybrid-Scout algorithm, the DV/LS component only calculates routes based on static metrics (i.e., it does not trigger on dynamic link cost changes) and only the Scout component computes paths using dynamic metrics. Because static metric paths are calculated once at the beginning of a simulation for all four algorithms, we remove the cost of the initial static route calculation for all algorithms in our experiment. Thus, all routing costs presented are due to changes in dynamic metrics.

The performance of the different routing algorithms is measured by the average packet delay. Packet delay is the elapsed time between the sending of the packet and its reception. We construct each experiment such that packet drops do not occur; therefore, the average packet delay reflects the amount of queuing that packets experience, which indicates the degree of network congestion along a packet's path.

The queue size for each router's outgoing link is 100 packets long. A router samples the length of its queues every 10ms. A link's current average queue length is calculated from the average of its instantaneous queue length and the previous running average. The dynamic metric cost function is linear from the static cost (average queue length of 0%) to 3 times the static cost (average queue length of 100%). All links have the same static link cost, and all three algorithms use the same static link cost, dynamic link-cost calculation mechanism, and link-cost function.

B. A Forest Topology

The purpose of the first experiment is to show the basic behavior and scalability of each algorithm. We run DV, LS, and hybrid DV-Scout algorithms on the topology shown in Figure 2. The picture on the left shows a backbone topology consisting of 6 highly connected nodes. Three of the backbone routers are attached to a tree-like topology, shown on the right. The connections are denoted by a dotted triangle attached to a backbone router. The backbone links have three times the capacity of tree links and all links have equal static costs. There are also cross links between an interior router in each tree and the corresponding router in the neighboring tree, which is not shown in the figure. Four hosts are attached to each tree router and one host is attached to each backbone router. The network has a total of 111 hosts. For the purpose of the routing protocols, each host represents a separate destination. Leaf nodes transmit a stream of 10,000 packets to a host attached to a backbone router at 75% link capacity. Backbone routers with attached hosts that receive packets generate Scouts (i.e., they are computing dynamic metric paths to themselves).

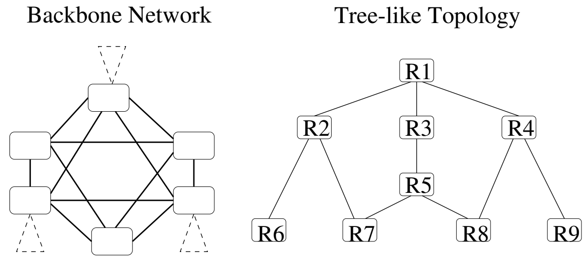


Fig. 2. The left figure shows the backbone network. The right figure shows the tree networks attached to the backbone routers, indicated by the dotted triangles. Boxes represent routers and edges represent links between routers.

B.1 Routing Behavior

Before examining the simulation results for the complete topology, we use one "tree" of our forest topology to demonstrate the routing behavior of each algorithm. For illustrative purposes, assume that $R1$ is the hot destination.

In this example, as the leaf nodes send their packets to $R1$, the queues for links $(R2, R1)$ and $(R4, R1)$ will begin to fill up. As they accumulate, the costs for those links increase. Once the cost of those links exceeds twice the link's base value, the shortest path from $R7$ and $R8$ to $R1$ shifts to the alternate path via $R3$, while $R6$ and $R9$'s packets still travel their primary path. After a while, the queue on $R5$ will also start to accumulate, and the traffic will shift back to the primary path. This shifting process continues until the transfers end.

The speed at which this path recalculation occurs depends on the sensitivity and granularity at which each algorithm is calculating the shortest path to $R1$. For LS and DV, this is parameterized by their trigger percentages: the lower the percentage, the faster the algorithm will trigger update messages upon link cost changes, and the faster it responds to congestion. For hybrid-Scout, it is controlled by its broadcast interval BI: the lower the BI, the better the calculated paths reflect the current link costs.

B.2 Routing Performance Versus Cost

We now present simulation results for the topology shown in Figure 2. In this experiment, we vary the number of hot des-

[3] Our implementation includes the "split horizon" and "poison reverse" heuristics.

tinations by having leaf nodes transfer data to varying numbers of hosts attached to backbone routers. The traffic is uniformly distributed among these "hot" destinations. There are potentially two levels where routing based on dynamic metric can increase performance. The first is within each "tree" (as illustrated above), the second is in the backbone network, where multiple alternate paths exist.

The results are shown in Figure 3. The leftmost graph shows average packet delay of DV and LS with dynamic metrics as a function of the trigger percentage, and the middle graph shows hybrid-Scout's performance as a function of the broadcast interval. The routing cost incurred by each algorithm are shown in the rightmost graph of Figure 3 (DV and LS) and the leftmost graph of Figure 4 (hybrid-Scout). The label x-node denotes the number of distinct destinations to which the leaf nodes are sending packets. The amount of traffic injected into the network for the various number of destinations is fixed; only the distribution of the traffic differs.

The performance and cost graphs show that hybrid-Scout achieves performance comparable to LS and DV at much lower routing cost. For example, the performance of LS/DV at 50% trigger percentage is comparable to hybrid-Scout's performance at a BI of 100ms and one hot destination. However, the routing cost of hybrid-Scout compared to LS and DV is approximately 15 and 40 times less, respectively. At five host destinations, hybrid-Scout, DV and LS achieve comparable performance at 100ms, 25%, and 50% trigger percentage; in this scenario, hybrid-Scout uses approximately 3 and 15 times less routing resources than LS and DV, respectively.

To highlight this cost-performance tradeoff, the middle and rightmost graphs in Figure 4 show the routing performance versus cost of each algorithm with 1 and with 5 hot destinations. The x-axis shows the routing cost, and the y-axis shows the performance at that cost. Results are better the closer they are to the origin, indicating good performance at low cost. Compared to LS and DV, hybrid-Scout consistently achieves better performance at lower costs.

B.3 Routing Cost Versus Network Size

The routing cost for hybrid-Scout and LS algorithms are both $O(L)$, where L is the number of links. The size of each Scout packet is constant (8 bytes for DV-Scout, 16 for LS-Scout) whereas the size of LS packets is proportional to the number of outgoing links per router. In addition, one hybrid-Scout broadcast ($O(L)$) recalculates all the current least cost paths to a Scout originating node. In LS, however, if there are several points of congestion (say m points), every router at those points of congestion needs to perform a routing flood to recalculate the least cost path ($O(mL)$).

DV is less scalable than either hybrid-Scout or LS on this type of topology because the size of each routing update is proportional to the number of destinations. The more destinations a network has, the higher the routing cost. With the network used in the experiment, the size of each DV packet is approximately 600bytes, compared to 32bytes for a LS packet.

Since LS and DV compute shortest paths to all destinations simultaneously, they are more likely to cause massive route shifts (many paths simultaneously rerouted to a common set of links). This shifting of many routes may result in congestion, causing additional triggered updates. In hybrid-Scout, the shifting is less severe, because 1) hybrid-Scout does not recalculate dynamic metric paths to all destinations and 2) paths calculations for different destinations are independent and staggered in time.

B.4 Sensitivity to Traffic Distribution

In our experiment, the number of Scout generating nodes is varied from 1 to 5 (approx. 1% to 5% of hosts) to determine the sensitivity of hybrid-Scout's performance and cost to the traffic distribution.

With respect to performance, we see in Figure 3 that hybrid-Scout actually performs better when there are multiple "hot" destinations. This is because Scout calculates least cost paths to different hot destinations at uncorrelated times, which allows hybrid-Scout to split traffic: paths to destinations that would have shared the same set of links if calculated simultaneously may not, because later route calculations take into account the effect of earlier route changes on network load.

The amount of traffic splitting that hybrid-Scout can achieve depends on the difference in path calculation times. If paths to two neighboring destinations are calculated simultaneously (and thus based on the same link costs), the likelihood that their paths will have common links is high. On the other hand, if their paths are calculated at different times and some link costs change in the mean time due to previous route changes, then their paths will have less links in common. This fact explains why hybrid-Scout's traffic splitting is more prominent at higher BIs. At higher BI's, the mean time between calculations to different nodes are higher, therefore link costs are more likely to reflect the effects of previous route changes. As a result, congestion causing traffic is more likely to be split and network performance is improved.

Hybrid-Scout's routing cost is proportional to the number of Scout generating destinations. As seen in Figure 4, the routing cost for five Scout generating nodes (approx. 5% of the destinations) is exactly five times the cost with one Scout generating node.

The performance and cost of LS and DV are largely insensitive to the number of hot destinations[4], since these algorithms calculate shortest paths to all destinations. The cost of updating paths to one hot destination is the same as updating paths to all destinations. This property manifests itself as coinciding lines in the LS/DV cost graph in Figure 3. LS and DV are able to split traffic to a lesser degree because they compute all pairs shortest paths simultaneously, as indicated by the relatively close performance curves in Figure 3.

[4] The performance and cost vary slightly because the traffic and congestion pattern change with the number of destinations.

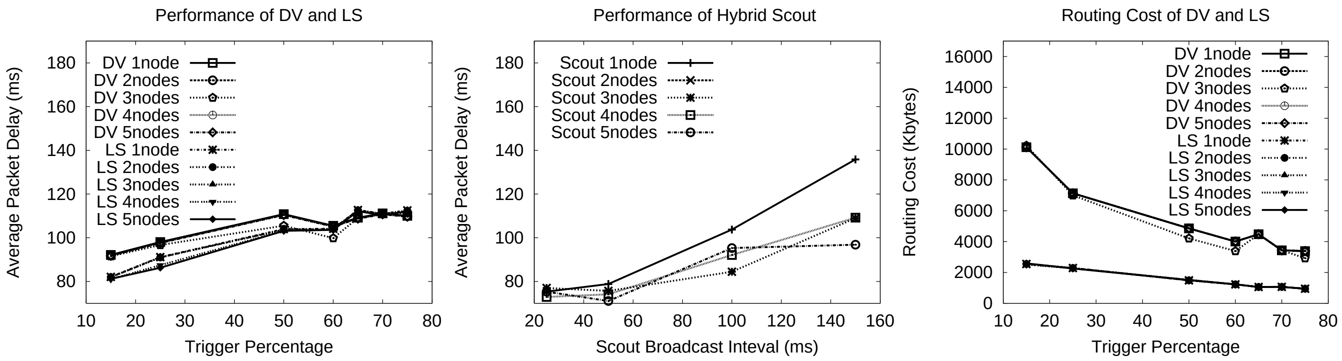


Fig. 3. The performance graphs for DV, LS and hybrid-Scout algorithms and cost graphs for DV and LS (right most graph). In the DV/LS performance graph, the top cluster of curves represent DV performance and the lower LS performance.

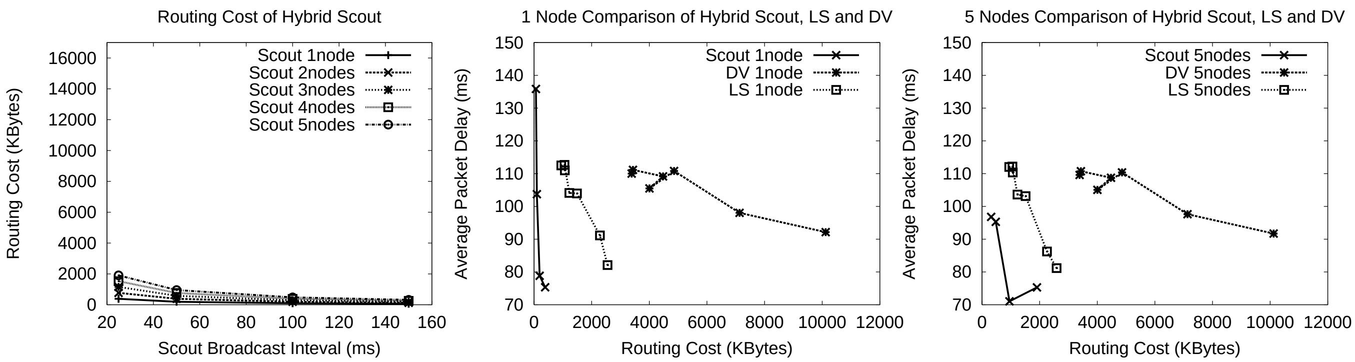


Fig. 4. Cost graph for hybrid-Scout and comparative performance of hybrid-Scout, LS and DV for 1 and 5 hot destinations.

C. Background Traffic

The purpose of the next experiment is to test 1) whether the benefits of hybrid-Scout observed in the previous section are maintained in the presence of background traffic, and 2) whether the performance of background traffic suffers with the hybrid-Scout algorithm.

We use the same network topology as in the previous section. In this experiment, the number of hot destinations is fixed at three (i.e., roughly 3% of the destinations are sending Scouts), and the foreground traffic is fixed at 50% tree-link capacity. At this rate, the foreground traffic by itself does not cause any network congestion; any congestion and subsequent rerouting in the network is caused by the additional background traffic.

To add background traffic, each host in the network repeatedly chooses a node at random and sends 100 packets to that node at 25% tree-link capacity. The amount of background traffic is controlled through the frequency of these transmissions. Increasing the background traffic increases the total amount of traffic injected into the network. Background traffic percentage is defined as the ratio between the amount of background traffic versus the total network traffic.

The experiment uses a trigger percentage of 15% for DV and LS and a BI of 25ms for hybrid-Scout. The average packet delay for the foreground and background traffic and the routing cost are shown in Figure 5.

The leftmost graph in Figure 5 shows the packet delay experienced by the foreground traffic. As the overall network traffic increases (marked by the increase in background traffic), packet delay also increases. The results also show that hybrid-Scout's ability to reroute and split traffic allows it to achieve lower packet delay than LS and DV. We conclude that the performance benefits achieved by hybrid-Scout are maintained in the presence of background traffic.

The middle graph shows packet delay experienced by the background traffic in the same simulations. The packet delay for the background traffic is less than the foreground traffic delay because most of the traffic do not encounter congested links. The fact that the background traffic performance of hybrid-Scout is comparable to DV and LS shows that the increase in route performance for selected destinations obtained by hybrid-Scout does not come at the expense of non-selected destinations. The intuition is that by shifting traffic that contributes most to congestion (foreground traffic), the remaining traffic on those congested links are not likely to continue causing congestion. Hence, the network performance increases for non-selected destinations as well. The benefits obtained by hybrid-Scout for foreground traffic also indirectly benefit background traffic, as long as there is "enough" foreground traffic. We address the question of what is enough traffic in the next

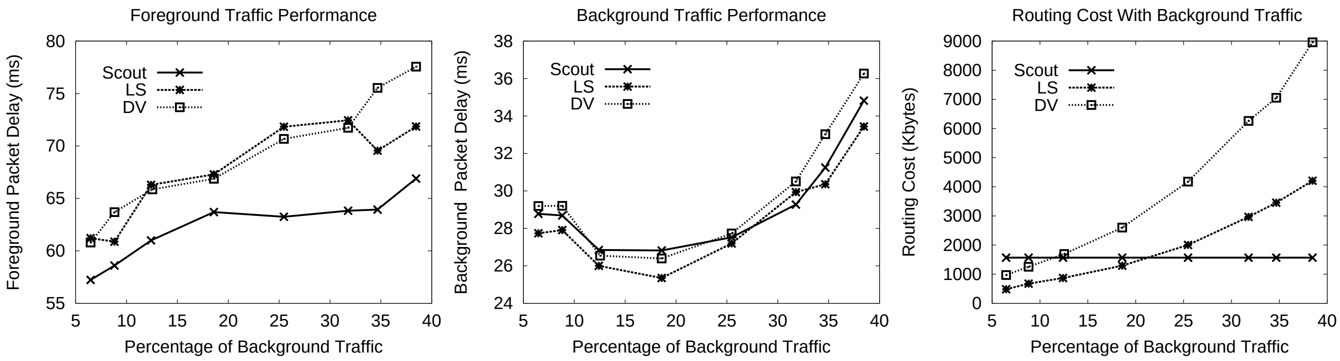


Fig. 5. The foreground and background performance (left two graphs) and the respective routing costs (rightmost graph) in the large network. In all three graphs, the percentage of background traffic represents the fraction of traffic in the network that is destined for non-hot destinations.

experiment.

An interesting feature in the background traffic delay graph is the dip near 12% background traffic. At less than 12%, the background traffic that traverse congested points is not enough to cause much rerouting; thus, packets experience queuing. Above 12%, enough congestion accumulates to cause rerouting, hence the packet delay is decreased because congestion is relieved. Of course, the packet delay again increases with background traffic, because the benefits of rerouting are offset by the increase in traffic. The dip at 12% is not observed in the foreground graph because the congestion that causes rerouting degrades the foreground performance such that there is a net increase in delay as a result of rerouting. The background traffic, on the other hand, always encounters congestion; therefore it can only stand to benefit from rerouting.

The rightmost graph in Figure 5 shows the cost of each routing algorithm in this experiment. The hybrid-Scout algorithm exhibits a constant routing cost with different levels of network traffic. LS and DV's costs increase with the amount of traffic, with DV increasing most rapidly.

The cost graph shows an important characteristic of the LS and DV triggering mechanism: as the network utilization increases (marked by the increase in background traffic percentage), the LS and DV triggered updates also increase. This is undesirable because these updates compete with data packets for link bandwidth and router CPU at a time when links are already heavily utilized and routers are busy forwarding packets. This additional network load increases the probability of severe congestion and packet loss.

The cost graphs also show that LS and DV use less routing resources than hybrid-Scout at low network utilization. Here, only a few links are getting congested, resulting in few triggered updates. However, the advantages of this behavior under low network utilization are not as important as the disadvantages under high utilization levels. Since most links and routers are underutilized at low network utilization, the presence of additional routing traffic does not have a significant adverse impact on network performance.

D. *Fraction of Foreground Traffic*

One of the premises of the hybrid-Scout algorithm is that the amount of traffic received by hot destinations must be "significant enough" for hybrid-Scout to be able to effectively reroute congestion, and that the number of selected hot destination must be "low enough" for hybrid-Scout to be efficient. The following experiment quantifies these conditions.

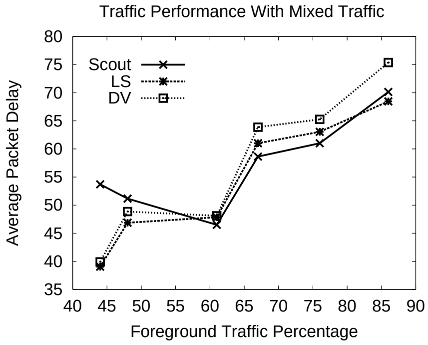
We use the same experimental setup as in the previous experiment (3 hot destinations). However, the total amount of traffic in the network is kept constant while varying the distribution of foreground and background traffic. The performance and cost are shown in Figure 6. Foreground traffic percentage is the ratio of foreground traffic to total network traffic.

The performance graph (left graph) in Figure 6 shows the average packet delay of both the foreground and background traffic. As the foreground traffic percentage increases, more traffic is directed to fewer destinations, causing more packet queuing and higher average packet delay for all destinations. The performance graph also shows that hybrid-Scout is able to achieve performance comparable to LS and DV whenever the foreground traffic accounts for greater than 50% of the network traffic. Notice that hybrid-Scout's performance is significantly worse than LS and DV at 45% foreground traffic. This is because Hybrid-Scout can only reroute paths to hot destinations; therefore it cannot eliminate congestion when these destinations only account for a minority of the traffic.

The cost graph (right graph) in this experiment shows that in our simulated network where 3% of the hot destinations are transmitting Scouts, the routing costs of hybrid-Scout are around 3 to 4 times less than LS and an order of magnitude less than DV.

The routing costs for LS and DV actually increase at 45% and 50% foreground traffic. The reason is that the background traffic is causing minor congestion at many points in the network (as opposed to mainly on paths to the hot destinations when foreground traffic dominates), thus triggering more LS and DV updates.

With the results of this experiment, we are able to answer the two questions posed earlier in this section:



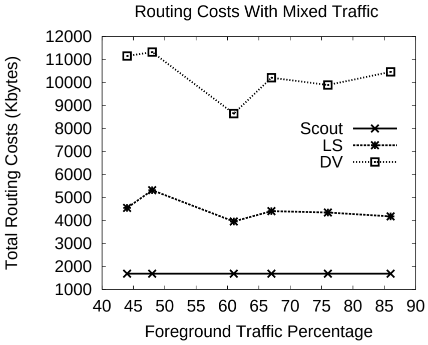


Fig. 6. Network performance and routing cost. The amount of traffic injected is held constant while foreground and background traffic percentages vary.

1. What fraction of total traffic do hot destinations have to receive in order for hybrid-Scout to adequately reroute congestion? The answer from our experiment is at least 50%. The intuition is that if the dynamic routing algorithm is able to control 50% of the traffic, the likelihood that the remaining 50% continues to cause serious congestion is low.
2. How many destinations can generate Scouts for the hybrid-Scout algorithm to be efficient? In our topology, if 10% of the nodes generate Scouts (in a highly utilized network), the cost of hybrid-Scout is comparable to LS. That is, hybrid-Scout is cost effective as long as the number of "hot" destinations is below 10% of the total number of nodes in the network. Note that hybrid-Scout's efficiency also depends on the network utilization. As shown in the previous experiment, at low utilization LS/DV tend to be more efficient, and hybrid-Scout is better at higher utilization levels. However, efficiency at low utilization is not as critical as the efficiency at high utilization levels.

Recall that in our study of Internet traffic locality, 1% of the destinations account for over 50% of the network traffic. This indicates that in the Internet, having the hottest 1% of the destinations generate Scouts will be as effective as LS or DV with dynamic metrics in increasing network performance. However, the cost of hybrid-Scout under these conditions should be approximately an order of magnitude less than LS and 2 orders of magnitude less than DV.

E. Convergence

As stated in Section II, the Scout algorithm (and thus the hybrid-Scout algorithm) converges to least-cost paths in at most l broadcast intervals after link metric changes, where l is the number of hops on the longest calculated path. However, this is a loose upper bound and in our simulations, the algorithm always converges in at most three BIs.

As with other routing algorithms, the path calculation may not converge when the time between link metric changes is smaller than the algorithm's time to convergence. However, even if the algorithm cannot converge due to continuous metric changes, the calculated path are an approximation of the least-cost paths, where the magnitude of the deviation from the least-cost path is related to the magnitude of the link cost changes during the last 2–3 BIs.

At large BIs, the time to converge for Scout may be significantly larger than that of LS and DV. However, it is important to note that unlike Scout, LS and DV must use trigger thresholds and hold-downs [9] to limit the frequency of metric changes and maintain stability. Therefore, even if LS and DV technically converge while Scout does not in a given scenario, the calculated path are in both cases an approximation of the actual shortest paths, since the cost metrics used in the LS/DV calculation do not reflect the actual instantaneous link cost metrics used by Scout. In particular, simulations we have performed show that hold-downs have a similar effect on path quality as the BIs used in Scout. Finally, note that the metrics used in our simulations to determine routing cost and performance take into account effects of route accuracy and stability.

IV. Related Work

One of the earliest testbeds for routing using dynamic link metrics was the ARPANET. The latest revision was done by Khanna and Zinky in [11], who developed the "revised" ARPANET routing metric. This metric replaced the existing "new ARPANET" dynamic link metric and was implemented in the ARPANET in 1987. Without modifying the routing algorithm, Khanna and Zinky was able to achieve better route stability and increase in network performance with their revised dynamic link metric. However, they point out that routing oscillations could still occur. This work is complementary to our own.

Wang and Crowcroft in [15] showed that routing algorithms that compute shortest paths between nodes are inherently susceptible to routing oscillations. Hybrid-Scout alleviates this problem by calculating routes based on dynamic metrics only for selected destinations and in a fashion that staggers route calculations for individual destinations over time.

Another use of dynamic metrics is described in [7]. This work uses signal stability as a dynamic link metric in a wireless, ad hoc mobile network. Here, paths that use stronger signal links are preferred over paths that do not. The dynamic link metrics here do not depend on congestion, but rather on the quality of a link's transmission. The routing algorithm used is a variation of the Distance Vector algorithm.

Johnson and Maltz proposed a host-initiated routing algorithm for mobile networks [10], called Dynamic Source Routing (DSR), similar to Scout. The operating environment is one which the network topology changes so rapidly that maintaining routes to different hosts is infeasible (such as in a mobile or ad hoc network). Their idea is to calculate paths on demand: whenever a node wishes to transfer data to another node, it calculates a path to that node. DSR is similar to Scout in that small messages are flooded to calculate paths. However they differ in several key aspects. First hybrid-Scout maintains paths between all nodes. In an environment such as the Internet, calculating routes whenever a node transmits data is prohibitively expensive. Second, DSR initiates path calculation from the source, as oppose to the destination. Third, Scout is guaranteed to converge on the shortest path, DSR simply computes a feasible path. Fourth, DSR uses source routing, whereas hybrid-Scout uses hop-by-hop forwarding.

Finally, Perkins [13] proposed the Ad Hoc On-Demand Distance Vector (AODV) algorithm: a hybrid of DSR and DV for routing in a mobile, ad hoc network. AODV uses DSR for on-demand route discovery of hop-by-hop routes and uses the DV mechanism for invalidating these routes when link failures are detected. AODV and hybrid-Scout are both hybrid algorithms. However, unlike hybrid-Scout where both the DV/LS and the Scout component compute routes, the DV component in AODV is only used to invalidate routes discovered by DSR.

V. Conclusion

Routing using dynamic metrics has been shown to increase network performance in real networks. However, dynamic metrics are not presently widely used due to the danger of routing instability and high routing costs.

In this paper, we present a new approach to routing using dynamic metrics that promises to overcomes the above limitations. The approach is based on the observation that a high degree of destination locality exists in traffic seen on real networks. Analysis of Internet traffic traces shows that 1% of the hot destinations receive over 50% of the network traffic.

Our proposed algorithm, hybrid-Scout, is able to calculate paths based on dynamic link metrics to selected destinations, while paths to other destinations are calculated using a traditional routing algorithm using static link costs. We performed extensive simulations to answer the question of how effective and efficient hybrid-Scout is at rerouting congestion compared to dynamic metric LS and DV algorithms, and under what conditions. In our simulations, we answer this question using an Internet-like topology consisting of over 100 host networks and around 30 routers. In summary, our simulations on this topology show that:

1. Hybrid-Scout is effective at rerouting congestion if at least 50% of the network traffic is destined to hot destinations (i.e. hybrid-Scout generating destinations). Hybrid-Scout is more efficient than both LS and DV with dynamic metrics if no more than 10% of the network nodes are generating Scouts.
2. Hybrid-Scout is more scalable than LS and DV with dynamic metrics. Achieving comparable network performance (measured in packet delay), hybrid-Scout has substantially less routing cost (in routing message bytes), from 4-5 times to 1-2 orders of magnitude.
3. The selective update mechanisms of hybrid-Scout better splits congestion causing traffic, which reduces route oscillations. This splitting is achieved by 1) only rerouting selected destinations based on dynamic link metrics and 2) by calculating new routes for those selected destinations in a time staggered manner.
4. Hybrid-Scout's routing costs are stable even under high network utilization levels. This ensures that hybrid-Scout does not exacerbate network load during heavy network loads. This is in contrast to LS and DV with dynamic metrics, which tend to increase routing traffic at high network loads, potentially causing instability.

VI. Acknowledgments

We would like to thank Liliana Liu, Zheng Zhu and Peter Perrino for their help with this paper and the anonymous reviewers, whose comments have helped to improve the paper. This research was supported in part by a grant from Southwest Bell Corporation's Technology Resources, Inc.

References

[1] G. Alkin and F. Baker. RIP version 2 MIB extension. RFC 1724, Xylogics, Inc., Cisco Systems, November 1994.

[2] R. Bellman. *Dynamic Prgramming*. Princeton University Press, Princeton, N.J., 1957.

[3] L. S. Brakmo, A. C. Bavier, and L. L. Peterson. x-Sim user's manual, 1996. http://www.cs.arizona.edu/classes/cs525/xsim/xsim.html.

[4] J. Chen, P. Druschel, and D. Subramanian. A new approach to routing with dynamic metrics. Technical report, Rice University. CS-TR 98-321, 1998.

[5] J. Chen, P. Druschel, and D. Subramanian. A simple, practical distributed multi-path routing algorithm. Technical report, Rice University. CS-TR 98-320, 1998.

[6] R. Coltun. OSPF: An Internet routing protocol. *ConneXions*, 3(8):19–25, 1989.

[7] R. Dube, C. D. Rais, K.-Y. Wang, and S. K. Tripathi. Signal stability based adaptive routing (SSA) for ad-hoc mobile networks. *IEEE Personal Communications*, February 1997.

[8] L. Ford and D. Fulkerson. *Flows in Networks*. Princeton University Press, Princeton, N.J., 1962.

[9] C. Hedrick. Routing information protocol, 1988. Internet Request for Comment (RFC) 1058.

[10] D. B. Johnson and D. A. Maltz. Dynamic source routing in ad hoc wireless networks. In Mobile Computing, edited by Tomasz Imielinski and Hank Korth, Chapter 5, pages 153-181, Kluwer Academic Publishers, 1996.

[11] A. Khanna and J. Zinky. The revised ARPANET routing metric. In *Proceedings of SIGCOMM*, pages 45–56, September 1989.

[12] J. McQuillan and D. Walden. The ARPANET design decisions. *Computer Networks*, 1, 1977.

[13] C. Perkins. Ad Hoc On Demand Distance Vector (AODV) routing. Internet-Draft, draft-ietf-manet-aodv-00.txt, November 1997. Work in progress.

[14] Y. Rekhter and T. Li. A Border Gateway Protocol 4 (BGP-4). RFC 1771, T.J. Watson Research Center, IBM Corp., Cisco Systems, March 1995.

[15] Z. Wang and J. Crowcroft. Analysis of shortest-path routing algorithms in a dynamic network environment. In *SIGCOMM Computer Communication Review*, pages 63–71, 1992.