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

July 30, 1998

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 characteristics, e.g., the congestion on the link. Our algorithm is destination-initiated: the 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 recomputes 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.

1 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 behavior). 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 [12] 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 the positive results of [12], dynamic metrics are not widely used in today's Internet. The technical reasons for this choice 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 [6]. 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. This time staggered route update for different destinations in the network reduces route oscillations. Traditional algorithms like Link State (LS) and Distance Vector (DV) with dynamic metrics recompute paths between all nodes in the network 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. Since route recomputations in Scout are independently controlled by the destinations, route updates to different destinations are usually staggered in time, 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 of shortest paths in a network. This is because in Scout, each destination independently computes paths to itself and doesn't share path computations with other nodes.

To obtain the benefits of Scout while keeping the routing overheads low, we integrate Scout with two traditional routing methods: the LS and the DV algorithms. The integration is based on selective use of the Scout algorithm. First, we use Scout only to recalculate routes based on dynamic link costs. Second, we use Scout only to update paths 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. We use the LS or DV algorithm with 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.

By using Scout with dynamic metrics to recalculate and improve the quality of paths to important destinations, and relying on traditional algorithms with static cost metrics to maintain other routes, we are able to reap 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. We provide motivation for our work in Section 2. In Section 3, we present Scout, our selective route recomputation algorithm that takes advantage of traffic locality in wide-area networks. Next we describe its integration with the Distance Vector algorithm in Section 3.2 and Link State in Section 4. In Section 5 we present experimental data showing the behavior of these algorithms. We conclude in Section 7.

2 Motivation

Our work is motivated by two beliefs. The first is that dynamic metric routing has great potential for increasing network performance measured in terms of packet drops and packet delays. Second, network traffic has extremely high destination locality: i.e., a high percentage of network traffic is destined for a few nodes. In the following subsections, we provide some evidence for these beliefs in a large wide area network, the Internet.

2.1 Routing with Dynamic Link Metrics

Adaptive routing algorithms such as Link State and Distance Vector use static link cost metrics to calculate shortest paths between pairs of nodes in a network. By static cost metric we mean that a link's cost is based on its static properties, such as latency and bandwidth. These algorithms recompute paths only when component (node/link) failure or recovery is observed. However, to maximize network performance, routing algorithms also need to take into account the amount of traffic on a link. This is important because the performance of a path depends not only on the static properties of each link on that path, but also on the link's dynamic properties such as congestion (i.e., queuing delay and packet drops). Using dynamic link metrics to calculate the "current" shortest path allows routing algorithms to disperse congestive areas by splitting traffic streams that cause congestion. This directly results in shorter packet delay and reduction in packet loss.

Versions of Link State and Distance Vector algorithms have been developed to use dynamic metrics. The primary mode of path recalculation is through *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 trigger threshold. If so, the router initiates a network wide shortest path recomputation.

The dynamic cost of a link can be calculated in various ways. The most recent work was done by Khanna and Zinky in [12] who developed the "revised" ARPANET routing metric in 1987. In their metric, a link's cost depends on its latency, bandwidth, and current utilization. Khanna and Zinky conducted experiments of their dynamic metric using a Link State routing algorithm on the actual ARPANET. They showed that dynamic metrics were able to achieve significantly better network performance over static metrics. In addition, they showed that their new metric, compared to an existing dynamic metric function, provided higher route stability and lower amounts of routing oscillations, even under high network loads.

However, Khanna and Zinky mention that their new dynamic link metric does not solve all route stability problems. In particular, they observed that some route recomputations do not disperse traffic, but simply shift them to another part of the network. When traffic is shifted to new sets of links, it cause congestions there and triggers another recomputation. Khanna and Zinky attributed these oscillations to route heredity which is endemic to single path routing methods. As we indicated in the introduction, a major reason for these shifts not considered by Khanna and Zinky is that current routing algorithms compute shortest paths between all pairs of nodes simultaneously.

Despite the heredity problem, the study by Khanna and Zinky showed that their dynamic metric, compared to the existing dynamic metric, was able to improve the 1987 ARPANET performance by a 46% reduction in round-trip delay, 18% increase in throughput, 19% decrease in routing updates, and a dramatic decrease in packet drops. We expect that with the growth of the Internet, in the amount of traffic and network size, the benefit of dynamic metric routing will be greater.

2.2 Internet Traffic Locality

A key factor exploited by our hybrid Scout routing algorithm is the destination locality of network traffic. We analyzed packet level TCP traces from DEC and LBL (for more information about these traces, refer to [1]). These traces contain an hour's worth of wide-area traffic between the traffic source (DEC or LBL) and the rest of the world. All the traces we analyzed show a high degree of destination locality. For all the traces we analyzed, at least 90% of all traffic (in bytes) were sent to 10% of the destinations. The ratio of traffic in packets show a similar property (a few were just below 90%). Figure 1 shows a cumulative plot of the largest trace we analyzed. The trace is called dec-pkt-4 and contains around 4 million TCP packet traces. The shape of this figure is typical of the other traces we analyzed.

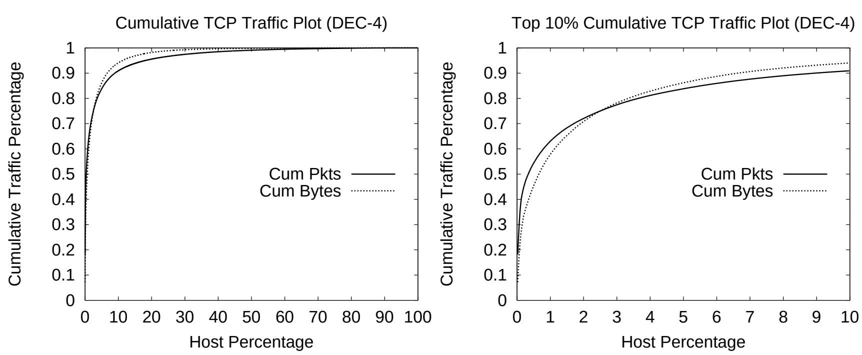

Figure 1: Traffic locality in dec-pkt-4 TCP packet traces. The left graph shows the cumulative distribution of traffic to destinations. The right graph shows distributions of the the top 10%. Note that the top 1% of the destinations receive around 60% of the traffic.

This analysis confirms our intuition of network traffic locality, both in local area and in wide area networks. For example, in a local area network, most of the traffic is to the network file server. And in wide area networks, most traffic appears to be destined for a few hot Web sites or to the major Internet service provider networks.

3 Our Approach

One of the key ideas of our approach is that we compute routes based on dynamic metric 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 and therefore reduces route oscillation. On the other hand, 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 take advantage of the traffic locality observed in networks today. This algorithm 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. Scout 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. In this way, the hybrid algorithms get the best of both worlds: DV and LS are efficient at computing paths between nodes in a relatively static environment[1] and Scout is efficient at selective path computation. These selected nodes should be ones that receive a large portion of the network traffic.

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 algorithms that integrate Scout with Distance Vector and Link State algorithms. We conclude by summarizing the benefits of this integration.

3.1 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. The forwarding table is used as follows: whenever a router receives a packet, the router forwards the packet to

[1] By this we mean an environment where link cost metric changes are infrequent.

the next-hop neighbor specified by the router's forwarding table for the packet's destination. This process of forwarding continues through the network until the packet reaches its destination. These router forwarding tables are updated by Scout.

Scout's basic mechanism of route computation is through message flooding. With Scout, nodes in the network periodically send a *Scout message* to all their neighboring nodes. 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, $C'_R = C_R + Cost(P \to Q)$. C'_R represents the cost a message will take if it is sent by P via Q to R.

In the first broadcast interval (ie. 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 from 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[2].

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. In other words, if P waited for a Scout from its designated neighbor Q in the previous BI but never received a Scout, instead of waiting for Q, or any other neighbor in the current round, P immediately floods the first Scout it sees in the current broadcast interval.

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. Our decision was motivated by the observation that propagating more recent, perhaps sub-optimal, information is more useful than trying to wait for the best information which entails the risk of not propagating any information at all.

As an example, consider node A sending Scouts in its first broadcast interval (sequence number 0). A trace of how Scouts propagate through the network is shown in Figure 2. Node A sends a Scout of the form $[A, 0, 0]$, denoting the Scout originates from A, with cost 0, sequence number 0.

[2]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.

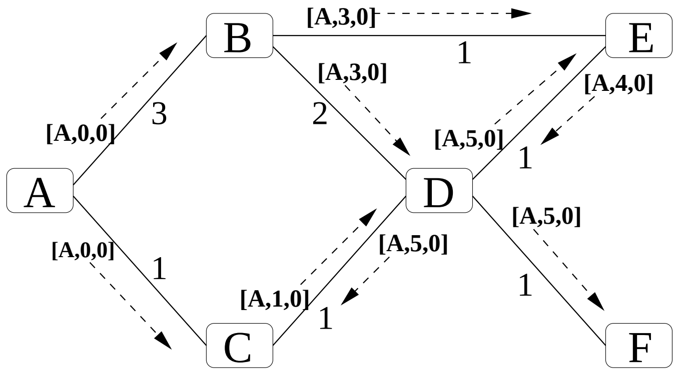

Figure 2: A Scout Example. Node A Sends Scouts for the first time. In this network, boxes denote routers and edges links. The number beside each link represents the cost of that link.

In this example, node A's Scout forwarded by B reaches D before the one forwarded by C. Since D has no previous information of A's Scout, D sends the Scout ($[A, 5, 0]$) to all its neighbors, after adjusting the Scout cost. Notice that D learns the optimal path to A when it receives the Scout ($[A, 1, 0]$) from C. However, D does not propagate this information in the current BI. Instead, C becomes D's designated neighbor (and next hop to A). In the first BI, the only node that did not get the optimal cost to A is node F, even though it has the optimal route. That is, because of hop-by-hop forwarding, F's packets destined to A will traverse the path (F, D, C, A) with cost 3 even though F thinks it's optimal path cost is 6.

In the next BI, D will wait for C's scout before sending its Scouts. The trace is given in Figure 3. Notice that D sends the Scout from C. The algorithm converges in two rounds and every node has the least cost path to A.

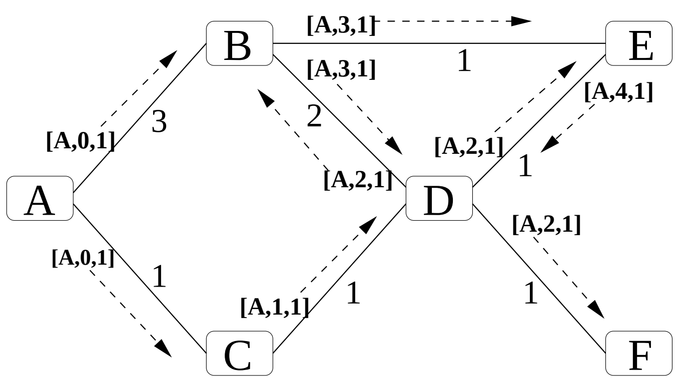

Figure 3: A Scout Example. A sends Scout in the second BI

There are several things to note about the Scout algorithm. First, a node only forwards one Scout per BI. This means that the number of Scouts forwarded in one BI is $O(L)$, where L is the number of links[3]. 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). In practice, we have simulated

[3] 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 $2L$.

Scouts on networks with more than 100 nodes and found that it always converges in at most 3 BIs, even after multiple node/link failures. 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,the Scout's route updates are completely controlled by the Scout generating node and not by network changes. Therefore Scout routing costs are easily controlled. Finally, Scouts require very little router computation. This is in contrast to LS and DV where an update may affect entire forwarding tables. We prove the correctness of the Scout algorithm and its convergence in a technical report [6].

3.2 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.

3.2.1 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 [4, 9] algorithm, and is the basis of many practical routing algorithms in current use [14, 3, 16].

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.

The task of the distributed DV algorithm is to continually update the forwarding table in each router to ensure that packets travel on the shortest path to their destination. Routers construct their forwarding tables by exchanging path information with their neighbors in the form of *distance vector packets* (DVP). Distance vector packets are excerpts of a router's forwarding table. A DVP carries the identity of the originating router, and a list of distance vectors of the form (*dst_addr*, *cost*), taken from the originating router's forwarding table.

Let C_d^r be the least path cost to destination d recorded in r's forwarding table, let N_d^r be the next hop neighbor on the least cost path to d recorded in r's forwarding table, and let c_{rs} be the cost of the link from r to s. It is assumed that the path cost metric is additive, that is, the cost C of a path $(x_1, \ldots, x_n)$ is $C = c_{x_1x_2} + c_{x_2x_3} + \ldots + c_{x_{n-1}x_n}$. Initially, the forwarding tables in each router r are initialized as follows:

$$C_r^r = 0;\ \forall s : s \neq r, C_s^r = \infty$$

When a router r receives a DVP from a neighbor s, r updates its forwarding table as follows. For all destinations d in s's DVP,

$$if\ (C_d^s + c_{rs} < C_d^r\ or\ N_d^r = s)\ then\ C_d^r = C_d^s + c_{rs}$$

where $N_d^r = s$. Notice the cost added to C_d^s is c_{rs}, the cost from r to s. This addition reflects the correct path cost from r to d through s.

Routers exchange distance vector packets with all their neighbors periodically, 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.

3.2.2 Scout-DV Hybrid

The Scout-DV hybrid algorithm is described in this section. In this 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 always computes a static shortest path to all destinations, the Scout component of the algorithm can take advantage of this. The only modification to the Scout algorithm is

> 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 only modifications to the DV algorithm are

1. When a router detects a link recovery, it exchanges forwarding tables with the router in the usual manner, propagating the Scout cost as if they were computed by DV. The receiving router computes the shortest paths by adding the received DVP costs to the static cost of transmitting on the recovered link. The designated neighbor for Scout entries are readjusted to the newly computed next-hop for their respective entries.

2. When a router detects a link failure, it sends to all its neighbors a DVP containing infinity cost for all destinations using the failed link, including the destinations being calculated by Scout (this is the normal DV failure procedure). This process nullifies all DV and Scout paths that use the failed link and recomputes new paths (if they exists) to the affected destinations using the traditional DV calculation mechanism. Again the designated neighbor is readjusted.

   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 destination 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.

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 are just as efficient in the hybrid as in the traditional DV algorithm. In addition, the selected

Scout generating destinations have the paths to them constantly refined according to the dynamic changes in link cost.

3.2.3 Correctness of Hybrid DV

In the hybrid-DV Scout algorithm, the DV and Scout components interact for optimization and not correctness. That is, if we completely disallow the Scout component to use DV information and vice versa, the hybrid routing algorithm will still converage on the shortest dynamic metric path for Scout generating destinations and shorest static metric paths for the others. The DV and Scout components share information to increase the speed of convergence.

The correctness of hybrid-DV Scout is that in steady state, static link shortest paths are calculated to non-Scout generating destinations and dynamic link shortest paths are calculated to Scout generating destinations. To prove the correctness of the Scout-DV interactions, we first distinguish destinations that generate Scouts versus destinations that do not. For those that do not, their paths are updated only by the DV component. Therefore their paths' correctness and convergence will be the same as in the pure DV algorithm. The correctness of DV has been proven in [4, 9].

With respect to the destinations that are updated by the Scout component, their paths are also updated by the DV component. We need to prove that the interaction of Scout and DV components on these paths are correct in the following sense:

1. Scout's usage of DV's static shortest next-hop neighbor as a default designated neighbor does not cause Scout to diverage on its shortest path computation for that destination.

2. During link failure, the DV component nullifies all Scout calculated paths and if DV calculates any alternate paths, that they are valid.

3. During link recovery, the DV component calculate valid paths.

Proof of 1: In the pure Scout algorithm, if a router receives a Scout in which it does not have any previous information, the router will treat the neighbor that sends the first Scout as its designated neighbor. With the optimization, the hybrid Scout will use the DV's next-hop neighbor as its designated neighbor for that destination. Notice that in the subsequent broadcast interval, the Scout component will use the designated neighbor it has computed from its previous round. Thus this optimization can only affect Scout behavior for the first BI (hopefully making Scout converage faster); therefore does not affect the correctness (ie. eventual convergence) of Scout to calculate shortest dynamic metric paths to its initiating node.

The proof of Scout convergence is irrespective of its default designated neighbor [6]. Thus, if we assume that the default designated neighbor for Scout is always the DV calculated static shortest path, the correctness of the Scout algorithms is still preserved.

Proof of 2 and 3: Whenever a router using the pure DV algorithm detects that a link has failed, it nullifies forwarding table entries to destinations whose next-hop is through the failed link. In addition, the router sends a DVP to all its neighbors indicating that it can no longer reach the list of nullified destinations. This starts a network wide path cancelation and recalculation to the affected destinations.

The complication with hybrid-DV Scout is that the DV component is nullifying paths calculated by Scout and later recomputing paths to those destinations. The question is whether this interaction causes erronous routing by DV or Scout. Here, we prove that the DV nullification and recalculation will produce the right paths.

The crux of the proof is that Scout's encoding of paths has exactly the same properties as DV's encoding of paths. Because they have the same properties, Scout's paths, when computed by the DV component, will have the same property as paths that were originally calculated by the DV component. Thus the correctness proof of DV's calculation using Scout's paths is the same as those for DV.

From the Bellman equations [4], the three necessary and sufficient conditions of paths calculated by DV to a destination are the following:

1. If path from node x_0 to node x_n is calculated to be $(x_0, \ldots, x_n)$, then node x_i's forwarding table entry has x_{i+1} as its next-hop to x_n, $0 \leq i < n$.

2. The cost x_{i+1} has to x_n is less than x_i's cost to x_n, $0 \leq i < n$.

3. The information of this path flowed from x_n to x_0 via the reverse path $(x_n, \ldots, x_0)$.

Using these three conditions of DV calculated paths, the DV link failure/recovery procedure is guaranteed to recompute the shortest path to any destination.

The Scout calculated paths also have these properties (the is from the correctness of Scout [6]), in addition to ones concerning sequence numbering. However, since the DV component does not make computation decisions based on Scout sequence numbers, this does not affect DV's computational behavior.

Because Scout's paths look exactly like DV computed paths, whenever the DV component updates its forwarding tables, it cannot distinguish paths calculated by Scout. Thus Scout calculated paths will be altered in the same way as paths calculated by the DV component. This implies that DV's alteration of Scout paths after link/node failure or recovery has the same correctness guarantees as DV's alteration of DV computed paths. Thus, DV is guaranteed to nullify all invalid Scout paths after a link failure and is also guaranteed to recompute valid paths to Scout generating nodes on link recoveries and failures.

4 Scout – Link State

In this section, we describe an integration of Scout with the Link State (LS) routing algorithm. The link state algorithm, like DV, is widely used and well understood. However, unlike DV, it relies on a centralized route calculation algorithm. A topology broadcast mechanism ensures that each router knows the current state of the entire network (i.e., topology and link cost), and routes are calculated in a centralized manner in each router. We begin with a description of the LS algorithm and then integrate it with Scout.

4.1 The Link State Algorithm

The Link State algorithm is the basis of many widely used routing algorithms. Protocols based on this algorithm are used in the Internet and in ATM networks [13, 2, 15].

In the Link State routing algorithm, every router periodically broadcasts (via flooding) its local connectivity. This information is contained in a link state packet (LSP) and consists of the router's ID, a list of its neighbor's IDs, and the cost of the connecting links. After a round of broadcasts by each router, every router has information about the entire network's topology. Each router then performs a shortest-path spanning tree computation rooted at itself. From the computation of this tree, every router knows the shortest path to all destinations in the network. This knowledge is encoded in the router's forwarding table. A LS forwarding table is a list of tuples of the form (*dst_addr*, *next-hop*), where *dst_addr* is the address of the destination and *next-hop* is the neighboring router on the shortest path to the destination.

Routers forward packets in the same way as in the DV protocol. Packets are tagged with a path ID consisting only of their destination address. Routers forward a packet by looking up their forwarding table using the packet's destination address and forwarding the packet to the specified next-hop neighbor.

4.2 Scout-LS Hybrid

The Scout-LS hybrid, like the DV hybrid, requires very little modification to either algorithms. The LS component maintains routes using the static link costs, recalculating routes only when link failures and recoveries are detected. The Scout component modifies the LS forwarding tables and uses dynamic metrics. Like the DV hybrid, Scout's only modification is that it takes advantage of the static shortest path computed by LS as its designated neighbor if it has no prior information. The only complication is their interaction when LS detects a link failure, as explained below.

The Link State algorithm does not compute routes by exchanging path information; therefore, when LS recompute routes, it cannot automatically correct Scout routes as in Scout-DV. This causes complications, especially when LS detects a link failure.

For example, if LS detects a link failure and recalculates its routes, a router is unable to determine whether a path calculated by Scout traverses the failed link, thus it is cannot correct the path if necessary. There are three basic approaches to deal with this problem:

1. Ignore the Scout calculated routes. This means a Scout path could potentially be invalid until the next Scout BI. This is a reasonable solution in networks where link/node failures are detected via "Hello" messages and the failure detection delay is on the order of the Scout BI.

2. Invalidate all Scout paths and use the static LS every time a recomputation occurs. This is undesirable because it results in drastic route changes throughout the network.

3. Invalidate only Scout paths that traverse the failed link/node. However, to do this accurately, a Scout needs to maintain a variable length field that lists all nodes/links the Scout has

traversed. This can dramatically decrease the efficiency of Scout.

Our approach is to selectively and conservatively delete Scout routes while maintaining fixed Scout sizes. The solution is as follows: we add a 64-bit field to hybrid-LS Scout (initially zeros) and require that each router identifies itself with a 64-bit ID. This ID does not have to be unique. Every time a router forwards a Scout, it bit-ORs its ID with the 64-bit Scout field. A router stores a Scout's 64-bit field if it wishes to use the path advertised by the Scout. Now whenever a link or node fails, the neighboring routers broadcast their link cost change along with their 64-bit ID. Upon receiving this LS broadcast, routers recompute their paths according to the LS algorithm. Then, for each destination calculated by Scout, if the broadcasting router's ID is contained in the 64-bit Scout field, that destination's path (next-hop) is changed to the LS computed static metric shortest path. This containment test is a simple bit-AND operation.

Note that this method always delete paths calculated Scouts that traverse a failed link/node. However, because this method is conservative, it may delete Scout paths that do not traverse the failed component. Our simulation shows that if a router selects its ID by marking randomly two bits in the 64 bit field, the number of false positives is around 10% for Scout paths that traverse 15 nodes. This method provides a reasonable false positive rate while keeping the Scout size constant.

For Link/Node recovery, the LS component only changes the paths for non-Scout calculated paths. We do not invalidate Scouts because those paths are still valid and we can safely rely on the Scouts to discover the newly created paths in the next broadcast interval.

4.3 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. The effectiveness of LS and DV with dynamic metrics, on the other hand, 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 and the number of Scout generating destinations. This allows complete control over Scout's routing cost.

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". And in order for Scout to by efficient, the number of hot destinations must be "low". We quantify these two conditions in Section 5 and argue that these conditions are present in real networks.

In summary, the main features of the hybrid Scout algorithms are as follows:

1. The ability to selectively upgrade a destination's path calculation to use dynamic metrics. This is done in Scout simply by having the destination generate Scout messages.

2. Hybrid Scout is able to split traffic (disperse congestion) better than LS or DV. Scout splits traffic in two ways. First, because shortest paths are not computed for all pairs of destinations simultaneously, route oscillations are less likely. Second, because Scout update paths to

different destinations at uncorrelated times, traffic streams are diverted more gradually. These phenomena are experimentally shown in Section 5.

3. The desired route quality for a Scout generating node can be determined independently. That is, 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 broadcast interval.

4. Scout routing traffic is independent of network traffic, so Scout routing traffic can be easily bounded by limiting the number of Scout generating nodes and their minimal BIs. LS and DV can also be bounded using hold-downs [10], however, the impact on route quality is less clear.

5 Simulation Results

In this section, we present extensive simulation results to evaluate the proposed hybrid Scout-DV and hybrid Scout-LS algorithms, and to compare their behavior with that of the plain DV and LS routing algorithms using dynamic metrics. The purpose of the experiments in this section is to test 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 the LS and DV algorithms with dynamic metrics, 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. By this we mean a network with a highly connected backbone network, to which multiple tree-like subnetworks attach. 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 four 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 none 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 the routing cost and overall route quality delivered by the different algorithms.

To evaluate the algorithms according to these criteria, we first run the different algorithms on an Internet like topology to show the different dynamic behaviors of each algorithm and their relative performance and cost. Next, in Sections 5.3 and 5.4 we introduce background traffic to examine the behavior of these algorithms with varying traffic distributions.

5.1 Simulation Environment

The simulation environment is based on the "x-netsim" package from the University of Arizona [5], an execution-driven, packet-level network simulator. The simulator takes as input a description of the network topology, including link characteristics such as bandwidth and propagation delay, and a set of software modules that implement the various protocols running on the routers and hosts of the network. Simulation time advances according to the calculated transmission and propagation delay of packets in the network. Software processing in the routers and hosts are assumed to have zero cost.

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 [10][4] and a Link State protocol similar to the Internet OSPF [7] 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 exactly the same. The only difference is that the hybrid LS-Scout messages are 8 bytes larger than those used in hybrid DV-Scout.

LS and DV use dynamic metrics to calculate paths by transmitting route updates in response to link cost changes. These updates are called *triggered updates*. The triggering sensitivity is parameterized by the *trigger percentage* (we call this trigger percentage instead of trigger threshold because the threshold is parameterized by a percentage). 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 (ie. 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 accurately reflects 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

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

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 cost function.

This function is slightly different then the one proposed in [12], which uses a piece-wise linear function. We choose our function because it provides a more gradual cost increase and gives us full control over the amount of routing updates using trigger percentages (DV and LS) and Broadcast Intervals (Scout). We have conducted experiments using the piece-wise linear function and the relative performance/cost of the three algorithms remain the same. The correlation between routing updates and performance is not as obvious with the piece-wise function because of the non-uniformity in the cost function.

5.2 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 4. 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 a 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).

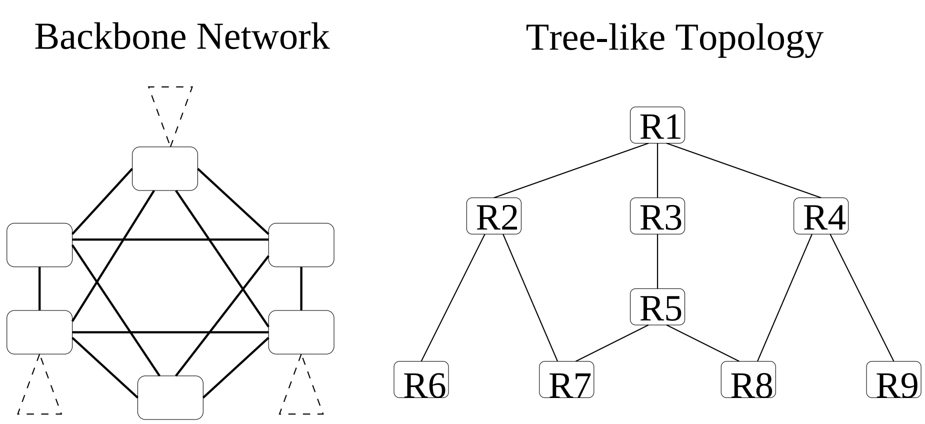

Figure 4: The left graph shows the backbone network. The right shows the tree networks attached to the backbone routers, indicated by the dotted triangles. Boxes represent routers and edges represent links between routers.

5.2.1 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.

5.2.2 Routing Performance Versus Cost

We now present simulation results for the topology shown in Figure 4. In this experiment, we vary the number of hot destinations 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.

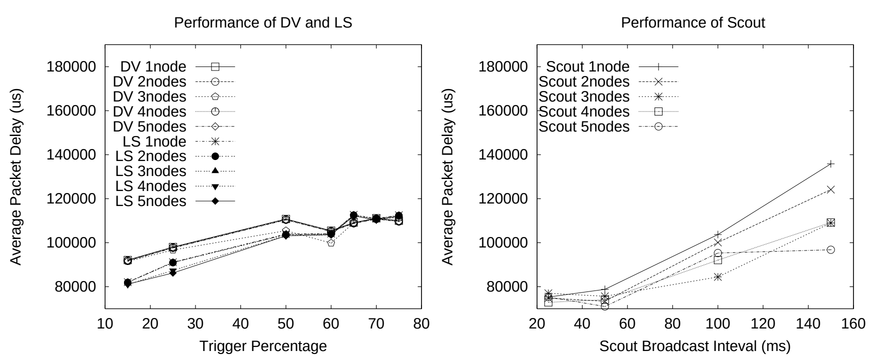

Figure 5: The performance graphs for DV, LS and hybrid-Scout algorithms. In the DV/LS graph, the top cluster of curves represent DV performance and the lower LS performance.

The performance and cost results of this experiment are given in Figures 5 and 6. In the performance graphs in Figure 5, the left graph shows average packet delay of DV and LS with dynamic metrics as a function of the trigger percentage, and the right graph shows hybrid Scout's

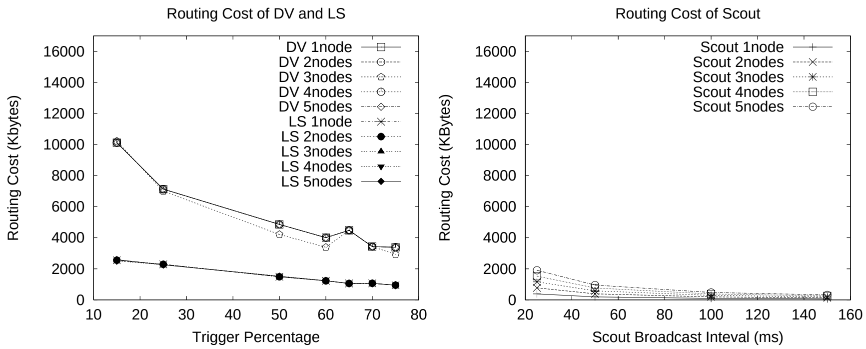

Figure 6: The cost graphs for DV, LS and hybrid-Scout algorithms. In the DV/LS graph, the top cluster of curves represent DV costs and the lower LS costs.

performance as a function of the broadcast interval. Figure 6 shows the routing cost, in Kbytes, of each algorithm. The label x-node denotes the number of distinct destinations that the leaf nodes are sending packets to. Note that the amount of traffic injected into the network for the various number of destinations is fixed; only the distribution of the traffic differs.

In the graphs, we see the effectiveness and efficiency of the hybrid Scout algorithm on this topology. The performance and cost graphs show that hybrid Scout achieves comparable performance to LS and DV at much lower routing cost. For example in Figure 5, the performance of LS/DV 50% trigger percentage is comparable to hybrid-Scout's BI of 100ms (1 hot destination). However, the routing cost of hybrid Scout compared to LS and DV is approximately 15 and 40 times less, respectively. At 5 host destinations, hybrid-Scout,DV and LS achieved comparable performance at 100ms, 25%, and 50% trigger percentage; hybrid-Scout used approximately 3 times and 15 times less routing resources than LS and DV respectively.

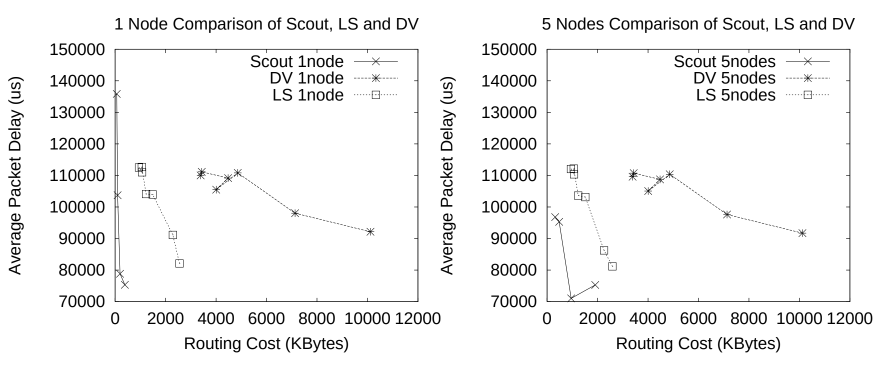

Figure 7: Comparative performance of hybrid-Scout, LS and DV for 1 and 5 hot destinations. The curves in these graphs are taken from graphs in Figures 5 and 6.

Figure 7 shows the cost-performance graphs for each algorithm with 1 and with 5 hot destinations. The results are based the same experiment as in Figures 5 and 6. The x-axis shows the routing cost and the y-axis shows the performance at that cost. An algorithm that is more to the left and bottom of the graph is better because it shows that the algorithm can provide better performance at lower cost. As seen from the left graph showing one hot-destination, hybrid Scout significantly outperforms both DV and LS on this topology. The graph clearly shows that for the same performance, hybrid Scout uses significantly less network resources than both LS and DV. For example, at a packet delay of 105000us, hybrid Scout requires approximately 100Kbytes of routing traffic while LS and DV require approximately 1224 and 4004Kbytes, respectively.

5.2.3 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 to the packet sizes, hybrid Scout uses less routing cost than LS because one 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 this large network, the size of each DV packet is approximately 600 bytes. This is compared to 32 bytes for LS.

In addition, since LS and DV compute shortest path to all destinations simultaneously, they are more likely to cause massive route shifts (many paths simultaneously rerouted to a common set of links). These shifting of many routes may result in future congestion which cause additional future trigger updates. This shifting is less severe in hybrid Scout because 1) hybrid Scout does not recalculate dynamic metric paths to all destinations and 2) paths to different destinations are calculated independently.

Table 1 shows how the routing cost of each algorithm scale with network size. For LS and DV, the trigger percentage was 15% and for hybrid-Scout, there was 1 hot destination with BI of 25ms. The tree topology is the same as the one in Figure 4 and the different numbers of trees as shown in Figure 4.

Tree Size	Hybrid-DV-Scout	Distance Vector	Link State	# of destinations	# of links
1-tree	52 Kbytes	35 Kbytes	60 Kbytes	9	10
2-trees	175 Kbytes	1,785 Kbytes	745 Kbytes	68	22
3-trees	385 Kbytes	10,120 Kbytes	2,550 Kbytes	111	45

Table 1: Table showing the scalability of the three algorithms.

As the table shows, the routing cost scaling characteristics for hybrid-Scout and LS are relatively linear with the network size. LS is higher because the number of congestive points also increases with the increase in network size. The number of hosts attached to each router increased from 1 to 4 between the 1-tree and 2-trees topology. Thus the routing cost for DV increased significantly between the two topologies. This table confirms our analysis of the different scalability characteristics of each algorithm.

5.2.4 Sensitivity to Traffic Distribution

Sensitivity to traffic distribution refers to the performance and cost of hybrid Scout when there are more than one Scout generating node. In our experiment, we range the number from 1 to 5 (approx. 1% to 5% of hosts).

In the hybrid Scout performance graph in Figure 5, we see that the performance of hybrid Scout is actually better when there are multiple hot destinations. This is because Scouts calculate least cost path to different hot destinations at uncorrelated times (as oppose to LS and DV which recalculates simultaneously). This allows hybrid Scout to split traffic: paths to destinations that would have shared the same set of links if calculated simultaneously do not because route calculations to those destinations are done at different times.

The amount of traffic splitting hybrid Scout achieves depends on the difference in path calculation times and how much link costs change between the different calculations. For example, if paths to two neighboring nodes are calculated using the same link costs, the likelihood that many of their paths will coincide is high. On the other hand, if their paths are calculated independently at different times and some links' costs change between those times, then their paths will have less links in common. This is why hybrid Scout traffic splitting is more prominent at higher BI. At higher BI's, the time between calculations to different nodes are also higher, therefore link costs are more likely to be different between two hybrid Scout calculations (for different destinations), so their traffic is better split and network performance is improved.

As expected, the hybrid Scout cost is proportional to the number of Scout generating destinations. As seen in Figure 6, the routing cost for five Scout generating nodes (approx. 5% of the destinations) is exactly five times the cost of one Scout generating node.

Notice that the performance and cost of LS and DV largely are insensitive to the number of hot destinations[5]. This is because these algorithms always calculate shortest path to all destinations. Therefore the cost of updating path to one hot destination is the same as updating paths to all destinations. This property is seen by the coinciding lines in the LS/DV cost graph in Figure 6. With respect to performance, LS and DV are less able to split traffic because they compute all pairs shortest path simultaneously. This lack of traffic splitting ability is shown by the relatively close performance curves of LS and DV in Figure 6.

To see the cost-performance ratios of each algorithm, the right graph in Figure 7 shows the relative ratios for 5 hot destinations. Again, we see that hybrid Scout is able to provide better performance at lower costs than LS and DV, and that LS is more efficient than DV.

In this experiment, all network traffic is destined to the hot destinations. In the next section, we simulate these algorithms' performance with background traffic to examine whether the benefits observed here are also obtained in the presence of other traffic.

[5]The performance and cost vary slightly because the traffic and congestion pattern is different for different numbers of destinations.

5.3 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, we fixed the number of hot destinations to 3 (ie. roughly 3% of the destinations are sending Scouts). The foreground traffic is fixed and is set at 50% tree-link capacity. At this rate, the foreground traffic itself does not cause any network congestion. Therefore any congestion or rerouting in the network is caused by background traffic.

To add background traffic, each host in the network repeatedly chooses another random host in the network and sends 100 packets to it at 25% tree-link capacity. Background traffic is increased by increasing the frequency of these transmissions. By increasing the background traffic, we increase the amount of traffic injected into the network. The background traffic percentage is the ratio between the amount of background traffic versus total network traffic.

The experiment uses a trigger percentage of 15% for DV and LS and a BI of 25ms for hybrid Scout. These parameters were chosen based on their comparable performance. The performance graphs for the foreground and background traffic are given in Figure 8.

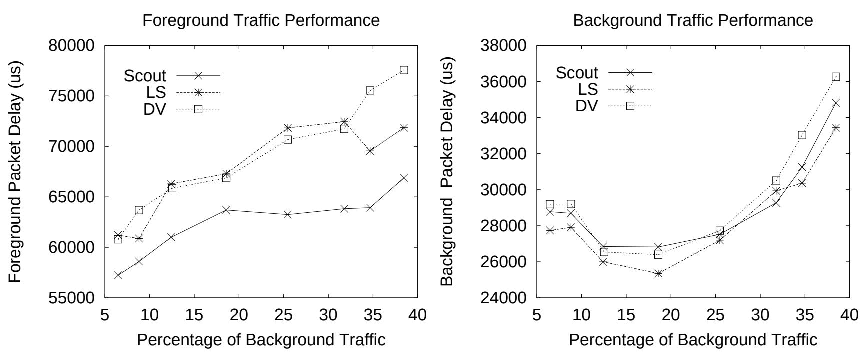

Figure 8: The foreground and background performance in the large network. The percentage of background traffic is the percentage of traffic in the network that is destined for non-hot destinations.

The left graph in Figure 8 shows the foreground traffic's packet delay. Note that as the overall network traffic increases (marked by the increase in background traffic), packet delay also increases. The performance graph also shows that the hybrid Scout's ability to reroute and split traffic is better than LS and DV as it achieves lower packet delays. This means that the performance benefit achieved by hybrid Scout is maintained in the presence of background traffic.

The right graph shows packet delays experienced by the background traffic in the same simulations. The packet delay for the backgound traffic is less than the foreground delays because most of the traffic does not encounter congested links. The background traffic performance of hybrid Scout is comparable to DV and LS. This shows that the increase in route performance for selected destinations obtained by hybrid Scout does not come at the expense of paths to other, 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 unlikely to continue causing congestion. Hence, the network performance increases for non-selected destinations as well. Thus, the benefits obtained by hybrid Scout for foreground traffic also indirectly benefit background traffic. This benefit is present as long as there is “enough” foreground traffic. We address this issue in the next experiment.

An interesting feature in the background traffic graph is the dip near 12% background traffic. The reason for the dip is that with less than 12%, the background traffic that traverse congested points was not enough to cause much rerouting; thus packets experienced queuing. However, above 12%, congestion accumulates enough to warrant rerouting, hence the packet delay is decreased because congestion is relieved. Of course, the packet delay again increases as background traffic increases because the benefits of rerouting are offset by the increase in traffic. The dip at 12% is not observed in the foreground graph. This is because the congestion that caused rerouting degrades the foreground performance such that there is a net loss in performance as a result of rerouting. Note that this net loss does not occur with background traffic on congested paths because the background traffic always encounters congestion. Therefore it can only stand to benefit from rerouting.

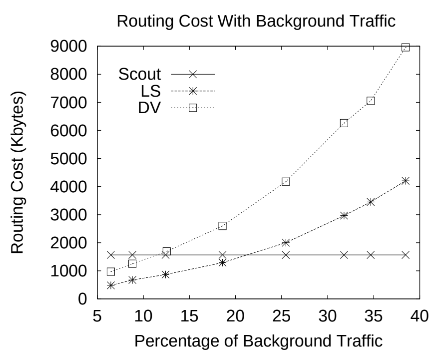

Figure 9: Routing costs for the experiment in Figure 8.

Figure 9 shows the cost of each routing algorithm in this experiment. The cost behavior of each algorithm is obvious here. The hybrid Scout exhibits a constant routing cost with different levels of network traffic where the LS and DV increase with the amount of traffic, with DV increasing most rapidly.

The cost graph shows a very important characteristic of LS and DV triggering mechanisms. That is, as the network utilization increases (marked by the increase in background traffic percentage), the LS and DV trigger updates also increase. This is very undesirable because these updates are directly competing with data packets for link bandwidth and router CPU at the most critical time when links are heavily utilized and routers are busy forwarding packets. This extra increase in network load increases the probability of packet loss and severe congestion.

On the other hand, the cost graphs show that LS and DV use less routing resources than hybrid Scout at low network utilization. This is because only a few links were getting congested and therefore there were few trigger updates. However, the advantages of this low network utilization behavior are not as important as the disadvantages under high utilization levels. This is because at low network utilization, most links and routers are not heavily loaded, therefore the presence of more or less routing traffic does not have a significant impact on network performance.

Note that LS and DV's routing costs can be controlled by hold-downs [16]. However, this comes at the cost of worse routing performance. We expect that with hold-downs, the routing cost can be kept low, but will result in significantly higher packet delay.

We mention that in a realistic implementation of hybrid Scout, a network's Scout generation rate should decrease if the traffic it is receiving decreases. More specifically, we expect that a Scout generating node will have a cap on the maximum number of Scouts it can generate over a period of time, and if traffic destined to it is low, it will reduce the amount of Scouts it generates. This way, the routing traffic is still controlled and the hybrid Scout's performance will be more efficient for a wider range of network traffics.

5.4 Fraction of Foreground Traffic

One of the main premises of the hybrid Scout algorithm is that the amount 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" such that hybrid Scout is efficient. The following experiment quantifies these issues.

For our last simulation, we use the same experimental setup as in the previous experiment (3 hot destinations). The only difference is that we keep the total amount of traffic in the network constant while varying the foreground and background traffic to change the distribution of traffic. The performance and cost graphs are given in Figure 10. Foreground traffic percentage is the ratio of foreground traffic to total network traffic.

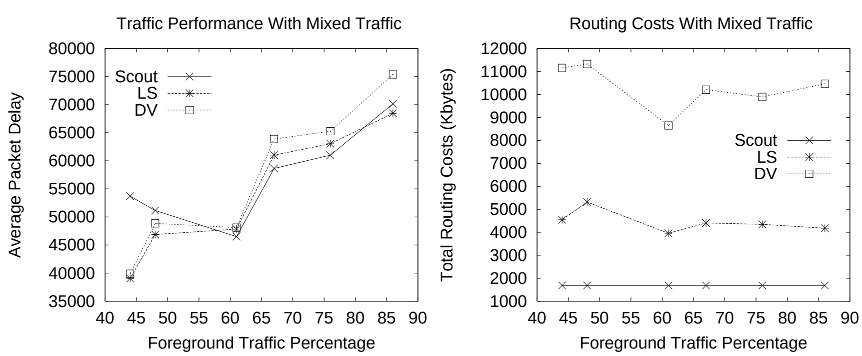

Figure 10: Network performance and routing cost. The amount of traffic injected is held constant while foreground and background traffic percentages vary.

The performance graph (left graph) in Figure 10 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 comparable performance 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 and at low foreground traffic, rerouting only foreground traffic is not significant enough to eliminate congestion.

The cost graph in this experiment shows that in a 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.

Notice that the routing costs for LS and DV actually increased at 45% and 50% foreground traffic. This is because the background traffic was 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 this experiment, we are able to answer the two questions posed earlier in this section:

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 a routing algorithm is able to control 50% of the traffic, the likelihood that the remaining 50% continue to cause serious congestion is low.

2. How many destinations can generate Scout 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 depend on the network utilization. As shown in the previous experiment, at low utilization LS/DV tend to be more efficient and at higher utilization levels, hybrid Scout is better. 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 accounts for over 50% of the network traffic. This indicates that in a network with such destination locality, having 1% of these hot 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 circumstances will be approximately an order of magnitude less than LS and 2 orders of magnitude less than DV.

5.5 Experimental Summary

The efficiency of hybrid Scout over LS and DV with dynamic metrics is derived from the observation that network traffic exhibits high destination locality. Because of this locality, the benefits of dynamic metric routing can be realized by concentrating on providing good paths to those hot destinations: if a high percentage of traffic is rerouted to avoid congestion, then most likely the remaining traffic will also experience less congestion. Selective dynamic routing also has the side-effects of reducing route oscillation and splitting of congested traffic. The obvious question is what type of network and traffic skew are necessary to achieve this benefit.

In our experiment, we showed the differences in routing costs and performance between the hybrid algorithms and LS/DV in an Internet-like topology. From the performance-cost graphs in Figure 7, hybrid Scout is able to provide better performance at lower costs. In these example, 100% of all traffic are destined for hot destinations. We also observed that hybrid Scout was able to split traffic even amongst paths to hot-destinations.

Our last two experiments show the behavior of these algorithms in the presence of background traffic. The first experiment shows that the performance advantages for the foreground traffic achieved by hybrid Scout do not come at the expense of background traffic. Second, the routing cost of hybrid Scout is constant even at high network utilization levels. This is in contrast to LS and DV where their traffic increases significantly at higher network utilization, which can have a detrimental effect on network performance.

The second experiment addresses the issue of foreground traffic percentage. Our experiments show that in the our topology, hybrid Scout's performance is comparable to LS and DV whenever the traffic destined for Scout generating nodes exceed 50%. Furthermore, the cost of hybrid-Scout is much less than either LS or DV when around 3% of the nodes are generating Scouts. We expect that hybrid-Scout will be very effective in networks such at the Internet because the destination locality is very high.

6 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 [12], 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 [17] showed that routing algorithms that compute shortest paths between nodes are inherently susceptible to routing oscillations. Although they can be reduced by using better dynamic link metrics, routing algorithms that calculate all-pairs shortest paths simultaneously cannot avoid route oscillation under certain conditions. 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 [8]. The work uses signal stability as a dynamic link metric in a wireless, ad hoc mobile network. Here, paths that traverse 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.

Johnson and Maltz proposed a host-initiated routing algorithm for mobile networks [11], 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 network). The idea here is to calculate paths to nodes 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 converage on the shortest path, DSR simply computes a feasible path. Fourth, DSR uses source routing, whereas hybrid-Scout uses hop-by-hop forwarding.

7 Conclusion

Routing using dynamic metrics have 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 a high percentage of network traffic is destined for a small percentage of destinations. Of the traces we analyzed, 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 when compared to the conventional LS and DV algorithms used with dynamic metrics, 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 hybrid-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 that use 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, causing instability.

References

[1] Internet traffic archive traces. http://ita.ee.lbl.gov/html/traces.html.

[2] PNNI draft specification. ATM Forum 94-0471R13, 1994.

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

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

[5] 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.

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

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

[8] 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.

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

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

[11] 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.

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

[13] J. McQuillan, I. Richer, and E. C. Rosen. The new routing algorithm for the ARPANET. *IEEE Transactions on Communications*, COM-28(5):711–719, 1980.

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

[15] J. Moy. The OSPF specification. RFC 1131, October 1989.

[16] 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.

[17] 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.