# Active Route Cache Optimization for Ad hoc Networks

Nikhil I. Panchal and Nael B. Abu-Ghazaleh
Computer System Research Laboratory
CS Dept., Binghamton University
Binghamton, NY 13902–6000
{nikhil,nael}@cs.binghamton.edu

*Abstract*—**In mobile Ad hoc networks, the topology of the network is constantly changing as nodes move in and out of each others range. Routing must maintain required paths in such an environment with an acceptable overhead. On demand routing protocols maintain paths only when they are requested by flooding the request through the network at great cost. Caches are used to reduce the frequency of flooding. We quantify the benefits and drawbacks of maintaining caches (vs. doing a full search every time). We show that caching can have undesirable side effects including inefficiencies due to stale paths, and the use of low quality paths even when significantly shorter paths become available. Based on these observations, we investigate optimizations to the caching of the Dynamic Source Routing (DSR) algorithm including path verification, periodic scoped searches to discover shorter paths, and selective pruning of cached paths. With active cache management we have a new class of Ad Hoc routing algorithms that is proactive but only on paths that are in use. We believe that these techniques are general and can be applied to other on-demand routing protocols.**

## I. Introduction

Ad hoc networks [1] are networks formed entirely from wireless nodes; the nodes must cooperate in determining the network topology and forwarding traffic destined to other nodes [2]. They have important applications including military communication, search and rescue operations, sensor networks and are likely to play an important role in a future Internet that is increasingly wireless at the periphery. In ad hoc networks, routing is an especially difficult problem because the topology of the network is constantly changing as nodes move in and out of each other's range. Furthermore, the low bandwidth available to the nodes places a premium on minimizing the overhead required by the routing protocols.

Ad Hoc routing protocols fall into two categories: (1) Table-driven (proactive); and (2) On-demand (reactive). Table-driven protocols attempt to maintain consistent, up-to-date routing information among all nodes in the network. Thus, they require periodic route-update messages to propagate throughout the network. On the other hand, On-demand protocols initiate a route request flood whenever there is a path is needed between a pair of nodes. The advantage of the table-driven approach is that routes to any destination are always available without the overhead of a route discovery. In contrast, in On-demand routing, the source must wait until a route has been discovered, but the overhead is significantly less than Table-driven algorithms where many of the updates are for unused paths. A large number of routing protocols have been suggested including proactive (e.g., [3], [4], [5], [6], [7]), reactive (e.g., [8], [9], [10]) and hybrids (e.g., [11]). On demand protocols provide better routing performance (and potentially lower overhead) for networks were mobility is frequent [12], [13].

The cost of the route request (a network wide flood) is significant and can adversely affect the performance of the network. Furthermore, in the presence of multiple communicating hosts, several floods can be initiated at the same time causing congestion and hot-spotting. Ad hoc routing protocols rely on two primary techniques to reduce the cost of route requests: (i) caching: every route request can return multiple paths to the destination. Furthermore, it is sometimes possible for the node to learn additional paths (for example, by observing traffic from other nodes). When a path is broken, the node can switch to another path in its cache if one is available, reducing the frequency of floods; and (ii) flood localization: it is possible to restrict the range of the flood by stopping it from going down unpromising areas of the network. The restriction can be based on location information [14] or the previous path [15].

In this paper, we consider cache optimizations to on-demand routing protocols. While relying on cached paths can reduce the frequency of route request floods, it can lead to inefficiencies in the following two ways:

1. Stale cache paths: by the time a path is needed, it may

Submitted, Infocom 2002

no longer be there. This can cause significant lost time while data packets are sent down this bad path until the path is detected after a number of MAC layer timeouts. The effect is even worse for TCP as congestion avoidance mechanisms are activated. This is a classic cache coherence problem; the cache does not get invalidated when the original path expires.

2. Low quality cache paths: by the time they are needed the paths in the cache are based on an earlier snapshot of the network. Meanwhile, better quality/shorter paths may have become available. However, such paths are not discovered until all the paths currently in the cache expire. The length of the path can significantly affect the end-to-end bandwidth and latency between the communicating nodes [16].

Recently, Hu and Johnson investigated the effect of the caching structure and strategies on the DSR on-demand routing protocol performance [17]. They investigated the effect of the cache size and organization (path-based or link-based) on the network performance under different scenarios. In addition, they explored the effect of timing out old cache information (only for the link-based cache organization). Marina and Das [18] investigated additional optimizations that appear promising. More specifically, they use wider notification of failure to notify all nodes of a relevant failure (instead of only the source node to the dropped packet) to reduce the number of stale paths. They also study the use of negative caches to maintain a list of recently broken links and ignore any received paths that include these links. In this paper, we investigate additional optimizations to cache management. More specifically, we investigate (i) validation of cached paths so that stale paths are expired quickly; (ii) elimination of bad quality paths based on several criteria; and (iii) using a restricted short path search periodically to discover any shorter paths that might have become available. Although DSR is used to study the effect of these optimizations, we believe that they generalize to other routing algorithms as well.

The remainder of this paper is organized as follows. Section II overviews the DSR algorithm and analyzes the effect of caching in more detail. Section III describes the suggested optimizations. Section IV presents an experimental evaluation of the proposed mechanism. Finally, Section V presents some concluding remarks.

## II. Motivation – Analysis of Caching Effect

The Dynamic Source Routing (DSR) Protocol [8] for Ad hoc networks is an on-demand routing protocol that uses source routing. Using DSR, when an IP packet is to be sent from a source A to a destination B, A checks if it knows a path to B. If it does, the packet is sent along this path. However, if no path is known, a route request is generated. The route request is a network wide flood that propagates through the network until it is received by B. Whenever B receives a route request packet, it reverses the path and sends a route-reply packet back to A. As A receives route replies, it learns paths to B and is able to communicate with it. When the mobility of the nodes causes any link along the path to break, the break is detected by the a node along this link as it tries to send the packet across it unsuccessfully. More specifically, after a number of MAC layer failure retries/timeouts, the node decides that the link is broken and sends a notification to the source of this fact.

Without a route cache, every packet to be sent will trigger a route request flood. This is obviously an unacceptable solution because of the resulting overhead of the flood and the latency experienced by each packet. At a minimum, the route cache must hold the "active" path to each node with which the node is communicating. When the source receives notification of path failure, it must find a different route to the destination for subsequent packets. If only the active path was available in the cache route, a new route request flood is needed. However, since multiple routes are generally discovered by each flood, they can be stored in the route cache for use when a path breaks. In addition, the node can collect other path information from the packets it is forwarding or by promiscuously listening to traffic exchanged by its neighbors [19].

While using caches reduces the overhead of route requests, it can lead to inefficient operation for the following two reasons:

- Stale Paths: by the time a cached path is needed, it might be invalid (due to node movement). Using a stale path can add significant delay until the path is discovered broken. If many paths in the cache are stale, several stale paths may need to be followed before a valid path is found (or a route request is initiated).
- Low Quality Paths: routing protocols attempt to select the shortest path to the destination. The information available to the source is based on the information collected during the last route request. Thus, if a shorter path becomes available, it is not discovered until the next flood is initiated. As paths in the cache fail, the node will use the next path in the cache in order of path length. It is conceivable that it will be using long paths when significantly shorter paths are available in the network.

Both of these problems are instances of the cache coherence problem. More specifically, when a route request is initiated, the node learns information about the state of the network and caches that information. With time, node mobility invalidates some of the cache information by making

some of the paths invalid and introducing new paths that are not available in the cache.

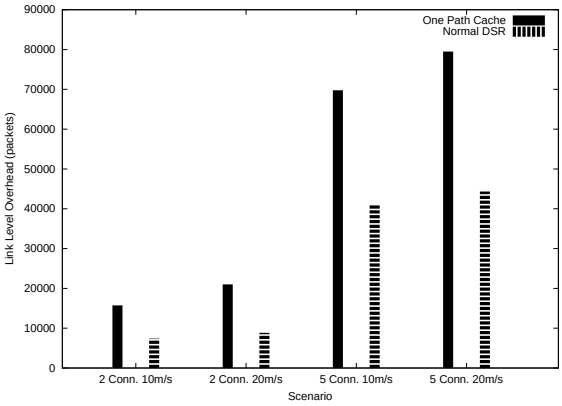


Fig. 1. Overhead

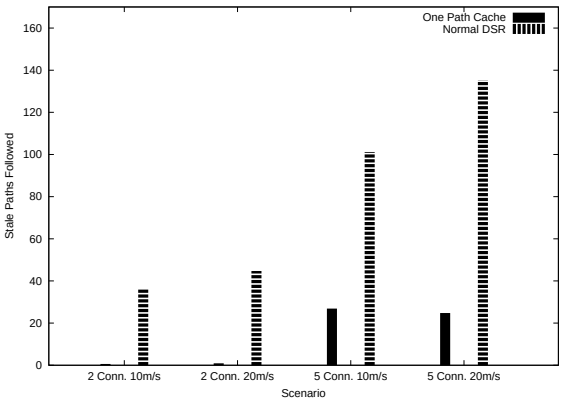


Fig. 2. Number of Stale Paths

Figure 1 shows the overhead incurred by normal DSR[1] vs. DSR with only one cache entry for every path. Thus, in the second case, every path failure requires a route request flood. It is obvious that the use of the cache significantly reduces the routing overhead.

On the other hand Figure 2 shows the number of bad cache paths followed by each implementation. Whereas the single entry implementation gets fresh paths (since they are always obtained in response to a full flood), the full cache version often uses bad paths. This in turn causes packet delay while these paths are followed and the path break is detected. Similarly, Figure 3 shows the average path length used by the two implementations. Here, we see that length is significantly shorter for the one cache

[1] DSR has many configurable options. For these results, cache replies were disabled since they perform so poorly [19]. The results are for 35 nodes in a 670x670 area using CBR traffic with ten 1024 byte packets per second; more detailed description of the experimental setup is provided in Section IV.

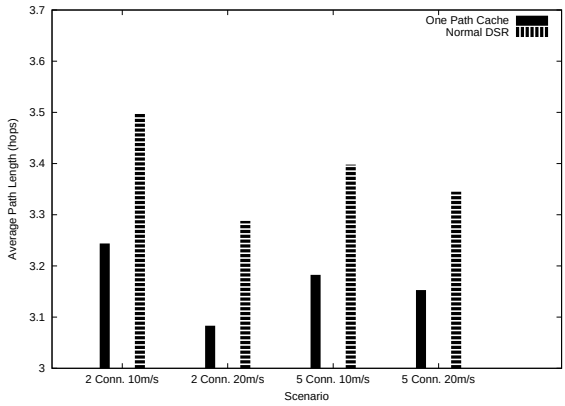


Fig. 3. Average Path Length

entry case because of the frequency the searches for new paths returns an updated snapshot of the best available path in the network. In comparison, the full cache version uses the paths available in its cache; these can be significantly longer than the best available path. A more detailed analysis of these effects under different cache organizations can be found elsewhere [12], [13], [19].

Given the behavior at these two end points of the caching spectrum, we would like to develop intelligent optimizations to the caching organization that deliver the advantages of caching (low routing overhead), without suffering the full extent of its drawbacks (large number of stale paths, and using low quality paths). These optimizations are described in Section III.

## III. Suggested Cache Optimizations

Optimizing the operation of the routing caches requires taking measures to reduce the effect of the invalid/missing cache information. In order to verify the validity of the paths in cache, we propose cache validation – sending a unicast packet along the path to discover whether it is still valid. Furthermore, we analyze different criteria for deciding which paths to validate and when to validate them. In order to discover any better paths that have become available since the last search (this is especially necessary once we are down the the bad quality paths in the cache), we propose scoped route searches that look for paths that are shorter than the best currently available path. Also, in order to avoid using low-quality or low-confidence paths, we consider selectively pruning the available paths in the cache. The remainder of this section describes these techniques in more detail.

### A. Path Validation

Having multiple paths available in the cache is desirable to minimize the frequency of the route request floods and

to provide seamless switching to a new path in the event that the current path breaks. However, as time passes, the paths in the cache can become invalid. Using invalid paths can cause significant latency while data is sent down the broken path until the break is discovered. One solution is to expire cache entries if they are old (e.g., Maltz et al [19] explored the effect of expiring cached information in link organized route caches). However, it is possible to expire valid paths if this approach is used. Wide error notification as suggested by Marina and Das [18] also targets eliminating stale paths but requires state information to be maintained at intermediate nodes (contrary to the philosophy of DSR) and generates many error packets for every hop failure even when a notification is for a path that will not be needed. Also, Marina and Das studied negative caches to protect from stale paths in transit that get overtaken by the path failure packet; the possibilities of this optimization are also mentioned in the DSR RFC [8]. Negative caches would also benefit our implementation (with path validation) since we do not protect against false paths in transit.

Path validation is an active mechanism for verifying path validity. More specifically, a paths is validated by sending a test routing packet to the destination. If the destination receives the packet, it replies to it (another alternative would be to simply discard it). However, if the path is broken, the test packet will be dropped and a route failure warning will be sent to the source. The source can then drop this stale path from its cache. Thus, the technique requires no support from intermediate nodes and generates overhead only for paths that the source is interested in validating.

Path validation is a mechanism that can be integrated into a more effective cache management policy. This policy has to determine what paths to validate (all the available paths, or a subset; how to determine this subset), and when to validate them (when the path breaks; or when they reach a certain age). If there are a large number of cached paths, validating all of them at the same time can lead to significant overhead and hot-spotting in the network, especially when considering that these paths can be close to each other. Thus, validation should be applied judiciously.

It is interesting to note that path validation is similar to localized queries as suggested by Castaneda and Das [15]. Castaneda and Das observed that paths to a destination tend to be similar to recently existing paths and use this information to limit the extent of a route request search to within $k$ hops around the path that broke. Path validation can be thought of as a localized query with $k = 0$.

### B. Scoped Route Searches

Path validation assists in discovering information in the cache that is no longer valid. However, there is no mechanism to discover new routes in the network that have become available since the last search was applied. This can lead to using low quality paths when shorter paths are available in the network (as was observed in Figure 3). To limit the effect of this problem, we use periodic scoped route searches. These searches are similar to floods, but are allowed to propagate only $k$ hops around the source. The value of $k$ must be smaller or equal to the best currently available path. Thus, the scoped request can discover any shorter paths that have appeared in the network since the last full search.

Even though the scoped route searches are not a full network search, they are expensive and should be used judiciously. We must balance the frequency of the searches against the overhead. In our implementation, we use a timer that is set after a route request (scoped or full) to trigger the scoped search after a prespecified time period. We investigate the effect of this period on performance.

Scoped path searches borrow on the idea of the scoped updates used in the Fisheye Routing Protocol (FSR) [6]. In FSR, the frequency of the updates is scoped such that nodes that are close to each other exchange link information more frequently than nodes that are far apart. In a sense, scoped searches represent an on-demand version of FSR where the scope is controlled by the best currently available path.

### C. Path Pruning

Many routes can be found by a route request process; both short, high quality paths, and circuitous long paths will be returned. The node goes through the list of paths in its cache in order of path length, eventually reaching and attempting to use these low quality paths. In this optimization, we seek to prune out low quality paths from the cache to avoid using them or even validating them. While other studies have expire older cache paths [17], [18], we take a risk-reward based approach to expiring paths.

We would like the paths in the cache to be high quality (short, and perhaps uncongested) and reasonably fresh. Such paths are worth using or validating. Path pruning can help limit the amount of paths to be validated by removing the low quality paths. We investigate criteria based on path length, number of available paths, and the age of the path. We note that longer paths, in addition to being of lower quality, have a shorter expected life time since a larger number of hops causes a higher number of potential path failure points.

## IV. Experimental Study

An extended version of UCB/LBNL network simulator (NS-2 [20]) was used for the experimental study. NS-2 is a discrete event simulator that was developed as part of the VINT. The extensions implemented by the CMU Monarch project [21] enable it to simulate mobile nodes connected by wireless network interfaces. The include a simple wireless propagation model using IEEE 802.11. The NS-2 DSR protocol cache management implementation was extended to add the cache optimizations outlined in Section III. We simulated a case with 35 nodes in a 670 by 670 square area. The nodes randomly pick a point and start moving towards it. When they reach it they pick a new point (zero pause time). We simulated Constant Bit Rate (CBR) traffic with each source sending 10 packets per second; each packet is 1Kbytes. We simulated scenarios with 2 senders and 5 senders for two mobility points: 10 m/s and 20/ms. At each scenario, 10 random choices of the senders and receivers are chosen and the results for them averaged.

In the initial implementation, we decided to trigger the path validation on a route error. More specifically, when a route error is received, we switch to the next path in the cache as well as send validation packets concurrently to all the available (cached) paths to the destination. Paths which are being validated are marked so that additional validation packets for them are not sent. Thus, all the stale paths are expired concurrently, rather than sequentially per conventional DSR. However, we noticed that a large number of validation packets was still being generated. Since these packets follow similar paths, they can easily cause hot-spotting in the network even though their overhead is significantly smaller than a route-request flood. In order to reduce this overhead, we investigated the following two criteria:

- Prune paths older than a time threshold, and validate the rest
- Prune paths longer than a given length, and validate the rest (with an upper limit on the number of concurrent validations). The length is determined relative to the shortest path seen since the last route request. Thus, capping the length at 1 means check paths of at most 1 hop longer than the shortest path seen since the last search and prune any longer paths.

Figure 4 shows the overhead of the routing protocol in terms of link-level transmissions. These packets include the route-request and reply packets, route errors and the validation packets. The implementation with one cache entry needs a route request flood whenever the path it has breaks; therefore, the overhead for that implementation is much higher than that of the other implementations. The pruned implementations trade off how aggressive the route requests are performed vs. the quality of the cache paths used. Therefore, as the threshold is increased, we see the overhead decrease (since it allows more caching). As can be seen in Figure 5, the number of stale paths followed was significantly reduced using cache validation. A stale path is a path which is not successful in delivering even a single data packet (excluding those that are discovered by the validation packets). Note that the single cache entry implementation hardly follows any stale paths since the path it uses is always freshly obtained in response to a route request.

From the overhead and stale path results, it appears that validation is a success – it significantly reduces the number of stale paths with a modest increase in the overhead. The packet delivery ratio (the accepted measure of routing protocol efficiency) with validation is uniformly better than that of DSR. Packet latency remains roughly the same; this is perhaps due to the CBR traffic generation model which has pause times between packet generation. Since we measure latency as the time between sending a packet and the reception of the next, a dropped packet suffers also a portion of the pause time.

The results also show that the single cache entry implementation has better packet delivery ratio and latency (but at a much lower overhead). We note that this single cache entry model has not been investigated in previous routing cache work which compare only to baseline DSR; it appears to perform very well for the small network scenario we are using. However, the overhead for this technique increases exponentially with the size of the network and its performance will likely drop off as the network size increases. We are currently conducting simulations to verify this conjecture.

After further analysis, we discovered that the triggering mechanism for the validation check (after a route error is received), restricts the utility of validation. More specifically, data is sent down the first cache path without validating that path, resulting in lost data packets and significant delays (up to 0.8 seconds in some cases) while that path break is detected. We are currently exploring alternative triggers for cache validation such as periodic validation in the background to establish high confidence in the cached routes by the time they are needed. We hope to present results for this implementation for the final version of this paper.

Figure 8 show the average path length used by the different implementations. While pruning helps in reducing the path length used, it does not help in detecting new, shorter

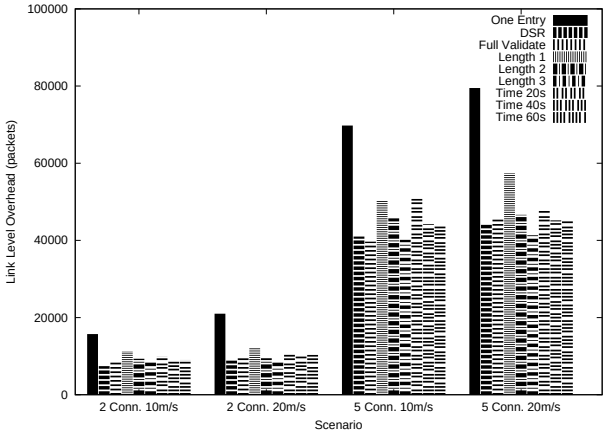


Fig. 4. Overhead Comparison

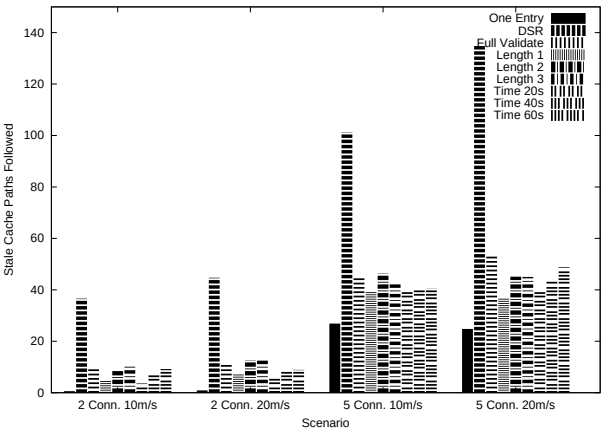


Fig. 5. Stale Paths Followed

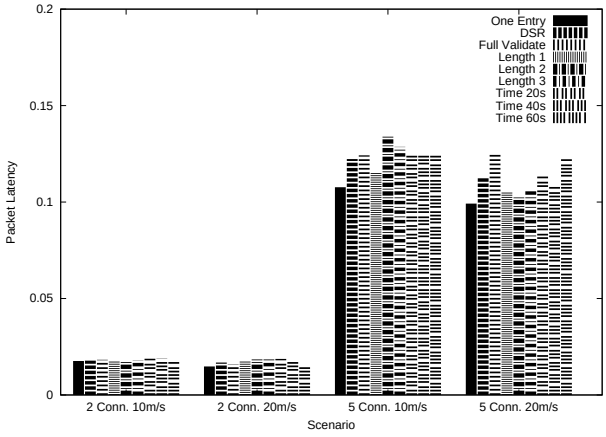


Fig. 6. Average Packet Latency

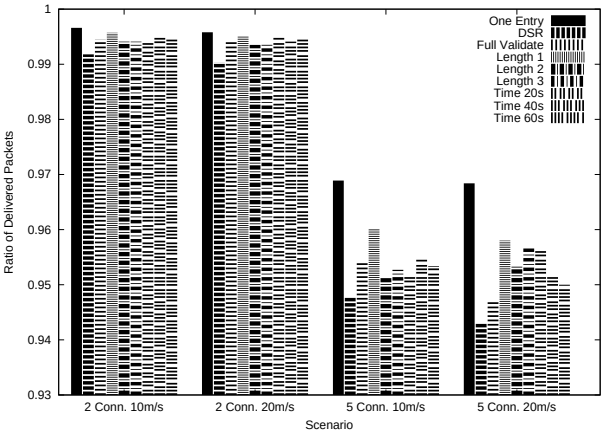


Fig. 7. Packet Delivery Ratio

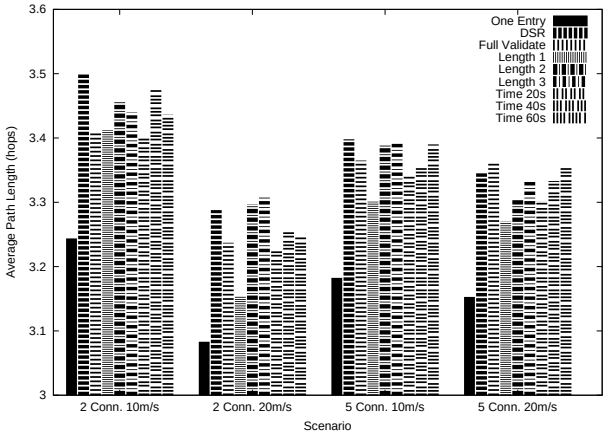


Fig. 8. Average Path Length

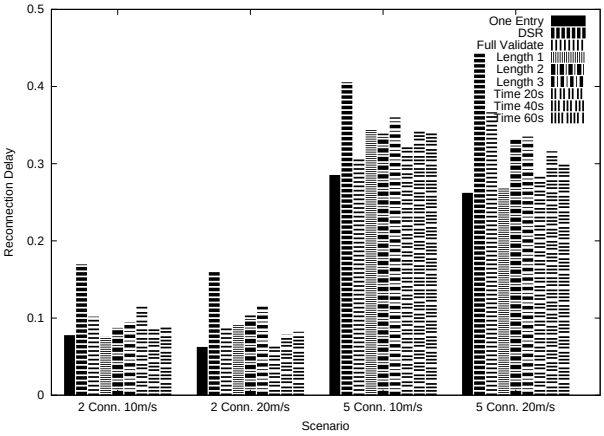


Fig. 9. Reconnection Delay

paths that become available. Figure 9 shows the average reconnection time after a path breaks. Validation significantly reduces this delay relative to regular DSR (since the number of stale paths followed is reduced). However, the delay is still longer than the route request time because of the large cost of following stale paths (this cost is almost completely avoided by the single cache entry implementation).

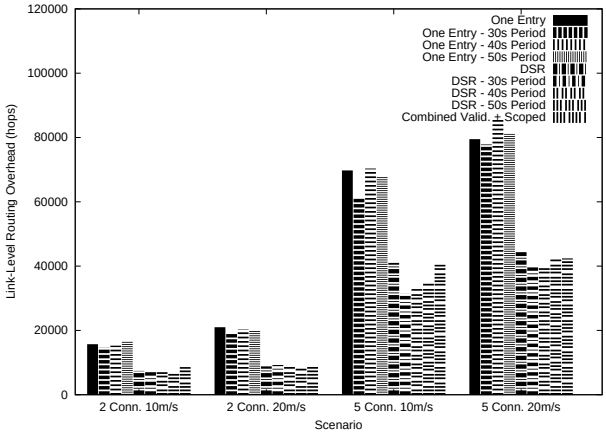


Fig. 10. Scoped Search – Overhead

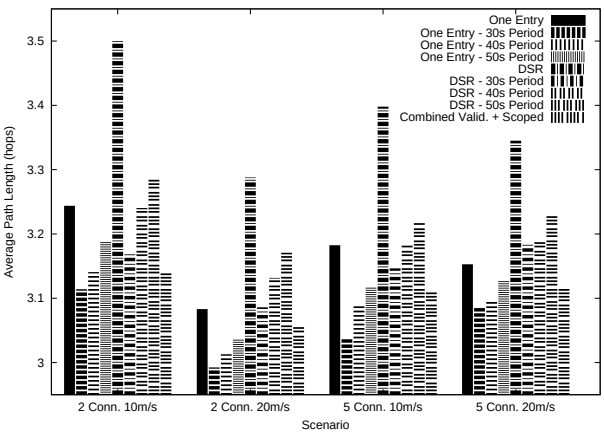


Fig. 11. Scoped Search – Path Length

The second set of experiments implemented the scoped periodic search as described in Section II. We investigated a scoped search after 30, 40, or 50 seconds after the last route-request (or scoped search) have passed. The difference between a scoped search and a route request is that the search looks for paths of length equal to the current path or shorter; thus, if the current path is of length 3, the route request only propagates 3 hops away from the source, limiting its overhead. We investigated adding scoped searches to the single cache entry implementation, and to normal DSR. Figure 10 shows the overhead results. Surprisingly, the overhead was reduced most of the time by doing the voluntary scoped searches. A possible explanation is that the search returns fresh/more stable paths and reduce the frequency of the full searches. Figure 11 shows the average path length obtained using the scoped search. Clearly, the scoped search significantly reduces the average path length.

Note that the last bar on these figures represents the combination of the scoped search (period 30s) and cache validation triggered by route error with Length criteria (ping only entries of best seen length + 2 or shorter). The average path length obtained by this policy was significantly better than the DSR policies, with a comparable overhead.

Figure 12 shows the average packet latency. The scoped search significantly improves latency for the 5 connection scenarios. Figure 13 shows the ratio of sent packets that were successfully received. There was a significant improvement vs. normal DSR especially for the 5 connection case. For the two connection cases, the performance improvement is modest; already, over 99% of the packets were delivered successfully in those cases.

The choice of CBR traffic to study routing level issues has advantages and disadvantages. The advantages include that the traffic generation is independent of the state of the network or previous packets. However, it has some disadvantages including: (i) how to measure performance? and (ii) the pause time between packets generated adds noise to the results and can hide the cost of expensive routing delays that will show up under normal traffic. For these reasons, we also evaluated the performance of the policies under TCP traffic. In addition, whereas CBR does not provide a measure for the throughput/capacity of the network, the TCP traffic provides that. However, TCP suffers extra delays if stale paths are followed because the congestion control mechanisms take effect and slow the transmission rate. Furthermore, TCP benefits from reduction in path length [16].

Figure 14 shows the performance of the different cache management policies under FTP traffic (over TCP). In this experiment, one node sends a continuous stream of TCP packets to the other as fast as it can. The results were averaged over 10 runs (each for a single pair of communicating nodes) at each mobility point. As can be seen in these results, the full cache validation provides better performance (almost 20%) over regular DSR. Length threshold based validation provided a small improvement over the full validation. However, with the scoped search and length based validation, a throughput improvement of almost 60% relative to plain DSR is obtained. Again, the single entry cache performed very well (within 10% of the scoped search) because of the small network size.

## V. Concluding Remarks

In this paper, we investigated active mechanisms for route cache optimization for ad hoc networks. Route caches collect multiple paths to an active destination (from route requests, or other mechanisms) and hold them in case the current path breaks and alternative paths are

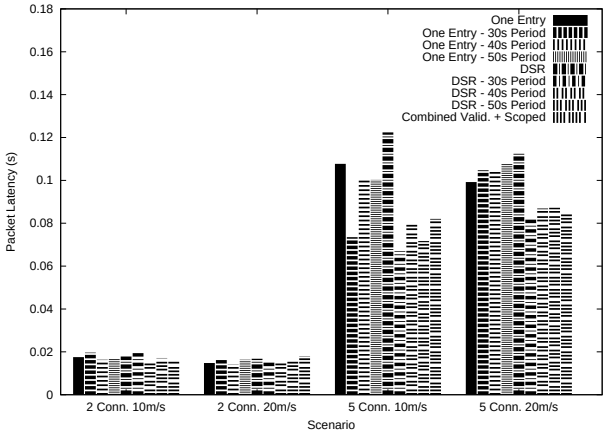


Fig. 12. Scoped Search – Latency

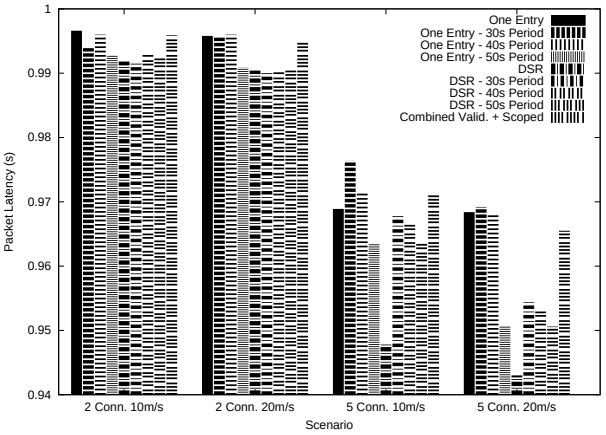


Fig. 13. Scoped Search – Delivery Ratio

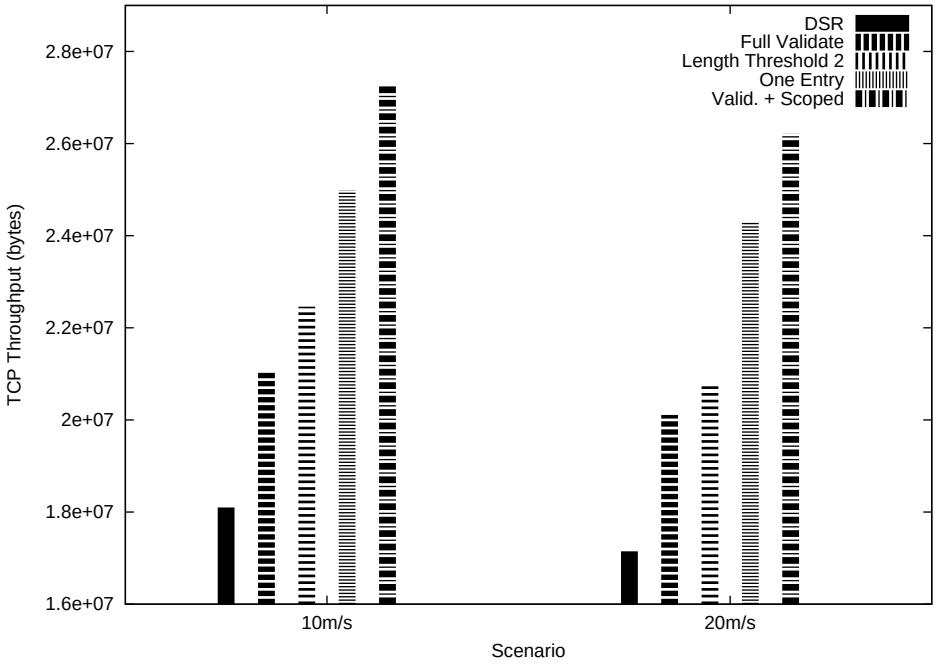


Fig. 14. TCP Throughput Comparison

needed. Route caches can significantly reduce the frequency of route request floods, an expensive operation where a network-wide flood is generated to locate the destination node. However, as time passes, some of the paths held in the cache become invalid; this causes large delays and dropped packets as data is sent down these broken paths. Furthermore, new short paths can appear in the network; they will not be discovered until the next route request is generated. Finally, the paths retained in the cache can be of poor quality (e.g., many hops longer than the best currently available path).

Three mechanisms were suggested to address these undesirable artifacts of caching: (i) cache validation; (ii) scoped route searches; and (iii) path pruning. Cache validation was introduced to reduce the number of stale paths in the cache. More specifically, validation packets are generated to whether a path is still live. If the path is broken, a route error is generated and the route is removed from the cache before it is used to send data. We investigated triggering the concurrent validation of paths to a destination on route errors (with mixed success). Our cache validation policy provided some of the effects that it was designed for (less stale paths, smaller reconnection delay), but provided mixed results for overall packet latency and packet delivery ratio. Further analysis of the traces showed that this could be due to the high probability that the first

backup path is stale (since the validation is triggered by the route error itself). We are currently investigating other triggers for cache validation, including periodic validation "in the background" so that we have high confidence in the "backup" paths by the time they are needed.

Scoped route searches address the problem of newly formed high quality paths that are not discovered until the current supply of paths in the cache are exhausted. Scoped route searches are route requests that are limited to $k$ hops away from the source. By choosing $k$ to be the length of the current best path, only paths of equal or better quality are discovered. This has two desirable effects: (i) better paths than the currently used one are discovered; and (ii) the cache is replenished with high quality fresh paths if they are available. This can potentially reduce the number of full route request floods (we noticed that in the drop in the overhead in some cases). We investigated triggering the scoped searches based on a fixed period (measured from the time of the last flood). Other criteria are possible; for example, it makes sense to take into account the current path quality (if there is a direct connection, no need to look for anything better). Scoped route searches demonstrated significantly better performance (latency, packet delivery ratio, average path length, etc..) relative to regular DSR, especially under high network load. This improvement makes a strong argument for other active local searches technique to optimize optimize the performance of Ad hoc networks under dynamic conditions.

Finally, path pruning is a mechanism where we simply choose to discard some paths. The decision to discard the paths is based on their quality relative to other paths that are available in the cache (or those that have been seen since the last route request). We investigated pruning paths based on age, length, and number available. Path pruning was used in conjunction with cache validation.

We evaluated the use of these mechanisms in active management policies for the DSR cache. The policies were heuristically chosen. The policies are a work in progress; there is definitely room for developing better policies (especially ones that trigger the mechanisms based on a consideration of the cache states, rather than periodically as ours do). Despite this, the results obtained from active management were very promising both for CBR traffic and TCP traffic. We believe that as we continue to optimize the policies, the advantages of active cache management for DSR will be clear.

Surprisingly, the single cache entry implementation performed well in everything other than overhead. This implementation initiates a route request flood whenever the path it has cached fails. Thus, it constantly retrieves the shortest path available in the network at the point when it initiates the search; this path is most often of high quality. It does not suffer from stale paths (only the path in use is cached). The experiments did show that the single cache entry implementation would benefit from scoped searches to discover shorter paths that appear in the network. Intuitively, this approach should suffer because its overhead is too high (a network wide flood); otherwise, there is no need to maintain complex caches. The scenario we studied was small (35 nodes), and the additional overhead generated by the single-cache implementation was not sufficient to counteract its positive effects. We conjecture that this is not a scalable solution: as the network size increases, it is likely that larger caches become beneficial. Under those situations, sophisticated cache management techniques are needed to ensure that only valid and high quality paths are retained.

Finally, although the policies were implemented with respect to DSR, we believe that they are fundamental to cache management (cache invalidation and cache refresh). Therefore, the ideas should carry over to other routing protocols. On demand protocols with active cache management represent a tradeoff between pure on demand and table driven protocols: they augment on demand protocols with proactive mechanisms but only with respect to currently active paths.

## REFERENCES


[1] Internet Engineering Task Force MANET Working Group, "Mobile ad hoc networks (manet) charter," http://www.ietf.org/html.charters/manet-charter.html.

[2] S. Corson and J. Macker, "Mobile ad hoc networking (manet): Routing protocol performance issues and evaluation considerations," Request for Comments, Internet Engineering Task Force, Jan. 1999, RFC 2501.

[3] C-C Chiang, M. Gerla, and L. Zhang, "Routing in Clustered Multihop, Mobile Wireless Networks with Fading Channel," in *Proceedings of IEEE SICON'97*, Apr. 1997, pp. 197–211.

[4] P. Jacquet, P. Mulethaler, A. Qayyum, A. Laouiti, L. Viennot, and T. Clausen, "Optimized link state routing protocol," Internet Draft, Internet Engineering Task Force, Mar. 2001, http://www.ietf.org/internet-drafts/draft-ietf-manet-olsr-04.txt.

[5] S. Murthy and J.J. Garcia-Luna-Aceves, "An Efficient Routing Protocol for Wireless Networks," *ACM Mobile Networks and Applications Journal*, pp. 183–197, Oct. 1996.

[6] G. Pei and M. Gerla, "Fisheye state routing in mobile ad hoc networks," in *Proceedings of ICC'2000*, 2000, pp. D71–D78, Internet Draft available at: http://www.ietf.org/internet-drafts/draft-ietf-manet-fsr-00.txt.

[7] C. E. Perkins and P. Bhagwat, "Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers," *ACM Computer Communications Review*, vol. 24, no. 4, pp. 234–244, Oct. 1994, SIGCOMM '94 Symposium.

[8] D. Johnson, D. Maltz, Y-C. Hu, and J. Jetcheva, "The dynamic source routing protocol for mobile ad hoc networks," Internet Draft, Internet Engineering Task Force, Mar. 2001, http://www.ietf.org/internet-drafts/draft-ietf-manet-dsr-05.txt.

[9] V. Park and S. Corson, "Temporally-ordered routing algorithm (TORA) version 1 functional specification," Internet Draft, Internet Engineering Task Force, Nov. 2000, http://www.ietf.org/internet-drafts/draft-ietf-manet-tora-spec-03.txt.

[10] C. Perkins, E. Royer, and S. Das, "Ad hoc on-demand distance vector (aodv) routing," Internet Draft, Internet Engineering Task Force, Mar. 2001, http://www.ietf.org/internet-drafts/draft-ietf-manet-aodv-08.txt.

[11] Z. Haas, M. Pearlman, and P. Samar, "Zone routing protocol (zrp)," Internet Draft, Internet Engineering Task Force, Jan. 2001, http://www.ietf.org/internet-drafts/draft-ietf-manet-zone-ierp-00.txt.

[12] J. Broch, D. Maltz, D. Johnson, Y.-C. Hu, and J. Jetcheva, "A performance comparison of multi-hop wireless ad hoc network routing protocols," in *Proceedings of the International Conference on Mobile Computing and Networking (MobiCom'98)*, Oct. 1998.

[13] P. Jacquet and L. Viennot, "Overhead in mobile ad hoc networks," Tech. Rep., Institut National De Recherche en Informatique Automatique (INRIA), June 2000.

[14] Y. Ko and N. H. Vaidya, "Location-Aided Routing (LAR) Mobile Ad Hoc Networks," in *Proceedings of the International Conference on Mobile Computing and Networking (MobiCom'98)*, Oct. 1998.

[15] R. Castaneda and S.R. Das, "Query localization techniques for on-demand routing protocols in ad hoc networks," in *Proceedings of the International Conference on Mobile Computing and Networking (MobiCom'99)*, Aug. 1999.

[16] G. Holland and N. H. Vaidya, "Analysis of TCP performance over mobile ad hoc networks," in *Proceedings of the International Conference on Mobile Computing and Networking (MobiCom'99)*, Aug. 1999.

[17] Y.-C. Hu and D. Johnson, "Caching strategies in on-demand routing protocols for wireless ad hoc networks," in *Proceedings of the International Conference on Mobile Computing and Networks (MobiCom'00)*, Aug. 2000, pp. 231–242.

[18] M. Marina and S. Das, "Performance of route caching strategies in dynamic source routing," in *Proceedings of the 2nd Wireless Networking and Mobile Computing Workshop*, Apr. 2001.

[19] D. A. Maltz, J. Broch, J. Jetcheva, and D. B. Johnson, "The Effects of On-Demand Behavior in Routing Protocols for Multi-Hop Wireless Ad Hoc Networks," *IEEE Journal on Selected Areas in Communications, special issue on mobile and wireless networks*, Aug. 1999.

[20] ," UCB/LBNL/VINT Network Simulator, web-site http://www-mash.CS.Berkeley.EDU/ns.

[21] ," NS-2 with Wireless and Mobility Extensions, available via web-site http://www.monarch.cs.cmu.edu.