ISSN:0975-9646

Distributed Cache Modernizing for the Multi-Source Transmitting Protocol

G.Thilipkumar [1] , Dr. R. Balasubramanian [2]

[1] Research Scholar, [2] Research Guide,
[1,2] J.J. College of Arts and Science, Bharathidasan University, Trichy.

Abstract-On-demand routing protocols use route caches to make routing decisions. Due to mobility, cached routes easily become stale. To address the cache staleness issue, prior work in DSR used heuristics with ad hoc parameters to predict the lifetime of a link or a route. However, heuristics cannot accurately estimate timeouts because topology changes are unpredictable. In this paper, we propose proactively disseminating the broken link information to the nodes that have that link in their caches. We define a new cache structure called a cache table and present a distributed cache update algorithm. Each node maintains in its cache table the information necessary for cache updates. When a link failure is detected, the algorithm notifies all reachable nodes that have cached the link in a distributed manner. The algorithm does not use any ad hoc parameters, thus making route caches fully adaptive to topology changes. We show that the algorithm outperforms DSR with path caches and with *Link-MaxLife,* an adaptive timeout mechanism for link caches. We conclude that proactive cache updating is key to the adaptation of on-demand routing protocols to mobility.

Keywords: **Mobile ad hoc networks, On-demand routing protocols, Mobility, Distributed cache Modernizing.**

1. INTRODUCTION

In a mobile ad hoc network, nodes move arbitrarily. Mobility presents a fundamental challenge to routing protocols. Routing protocols for ad hoc networks can be classified into two major types: proactive and on-demand. Proactive protocols attempt to maintain up-to-date routing information to all nodes by periodically disseminating topology updates throughout the network. In contrast, ondemand protocols attempt to discover a route only when a route is needed. To reduce the overhead and the latency of initiating a route discovery for each packet, on-demand routing protocols use route caches. Due to mobility, cached routes easily become stale. Using stale routes causes packet losses, and increases latency and overhead. In this paper, we investigate how to make on-demand routing protocols adapt quickly to topology changes. This problem is important because such protocols use route caches to make routing decisions; it is challenging because topology changes are frequent.

To address the cache staleness issue in DSR (the Dynamic Source Routing protocol) [6], [8], prior work [4], [11], [9] used adaptive timeout mechanisms. Such mechanisms use heuristics with ad hoc parameters to predict the lifetime of a link or a route. However, a predetermined choice of ad hoc parameters for certain scenarios may not work well for others, and scenarios in the real world are different from those used in simulations. Moreover, heuristics cannot accurately estimate timeouts because topology changes are unpredictable. As a result, either valid routes will be removed or stale routes will be kept in caches.

To evict stale routes faster, DSR with path caches uses a small cache size. However, as traffic load or network size increases, small caches will cause route re-discoveries, because more routes need to be stored, but small caches cannot hold all useful routes. If the cache size is set large, more stale routes will stay in caches because FIFO replacement becomes less effective. It was shown that path caches with unlimited size perform much worse than caches with limited size, due to the large amount of ROUTE ERRORS caused by the use of stale routes [4].

In this paper, we propose proactively disseminating the broken link information to the nodes that have that link in their caches. Proactive cache updating is key to making route caches adapt quickly to topology changes. It is also important to inform only the nodes that have cached a broken link to avoid unnecessary overhead. Thus, when a link failure is detected, our goal is to notify all reachable nodes that have cached the link about the link failure.

We define a new cache structure called a cache table to maintain the information necessary for cache updates. A cache table has no capacity limit; its size increases as new routes are discovered and decreases as stale routes are removed. Each node maintains in its cache table two types of information for each route. The first type of information is how well routing information is synchronized among nodes on a route: whether a link has been cached in only upstream nodes, or in both upstream and downstream nodes, and neither. The second type of information is which neighbor has learned which links through a ROUTE REPLY. Thus, for each link in a node's cache, the node knows which neighbor nodes have cached that link. Therefore, topology propagation state, the information necessary and sufficient to remove stale routes, is kept in a distributed manner.

We design a distributed algorithm that uses the information kept by each node to achieve distributed cache updating. When a link failure is detected, the algorithm notifies selected neighborhood nodes about the broken link: the closest upstream and/or downstream nodes on each route containing the broken link, and the neighbors that learned the link through ROUTE REPLIES. When a node receives a notification, the algorithm notifies selected neighbors. Thus, the broken link information will be quickly propagated to all reachable nodes that need to be notified.

Our algorithm has the following desirable properties:

- Distributed: The algorithm uses only local information and communicates with neighborhood nodes; therefore, it is scalable with network size.
- Adaptive: The algorithm notifies only the nodes that have cached a broken link to update their caches; therefore, cache update overhead is minimized.
- Proactive on-demand: Proactive cache updating is triggered on-demand, without periodic behavior.
- Without ad hoc mechanisms: The algorithm does not use any ad hoc parameters, thus making route caches fully adaptive to topology changes.

Each node gathers the information about which node learns which link through forwarding packets, not through promiscuous mode, which is an optimization for DSR [10]. To handle situations where promiscuous mode is used, we combine our algorithm and the secondary cache used in DSR with path caches, without any modification to the algorithm.

We evaluate the algorithm with and without promiscuous mode through detailed simulations. We show that, under non-promiscuous mode, the algorithm outperforms DSR with path caches by up to 19% and DSR with Link-MaxLife [4] by up to 41% in packet delivery ratio. Under promiscuous mode, the algorithm improves packet delivery ratio by up to 7% for both caching strategies and reduces latency by up to 27% for DSR with path caches and 49% for DSR with Link-MaxLife.

Our contributions are threefold. First, we addressed the cache updating issue of on-demand routing protocols. Second, we show that proactive cache updating is more efficient than adaptive timeout mechanisms. Finally, we conclude that proactive cache updating is key to the adaptation of ondemand routing protocols to mobility.

2. Related Work

Maltz et al. [10] were the first to study the cache performance of DSR. They found that the majority of ROUTE REPLIES are based on cached routes, and only 59% of ROUTE REPLIES carry correct routes. They also observed that even ROUTE REPLIES from the target are not 100% correct, since routes may break while a ROUTE REPLY is sent back to the source node. They concluded that efficient route maintenance is critical for all routing protocols with route caches.

Holland and Vaidya [3] showed that stale routes degrade TCP performance. They observed that TCP experiences repeated route failures due to the inability of a TCP sender's routing protocol to quickly recognize and remove stale routes from its cache. This problem is complicated by allowing nodes to respond to route discovery requests with routes from their caches, because they often responds with stale routes. Perkins et al. [14] showed the impact of stale routes on DSR.

Hu and Johnson [4] studied the design choices for cache structure, cache capacity, and cache timeout. They proposed several adaptive timeout mechanisms for link caches. In Link-MaxLife [4], the timeout of a link is chosen according to a stability table in which a node records its perceived stability of each other node. A node chooses the shortest-length path that has the longest expected lifetime. When a link is used, the stability metric for both endpoints is incremented by the amount of time since the link was last used, multiplied by some factor. When a link is observed to break, the stability metric for both endpoints is multiplicatively decreased by a different factor. Link-MaxLife was shown to outperform other adaptive timeout mechanisms.

AODV (the Ad hoc On-demand Distance Vector routing protocol) [12], [13] uses a precursor list for ROUTE ERROR reporting. For each route table entry, a node maintains a list of precursors that may be forwarding packets on this route. The list of precursors contains those neighboring nodes to which a ROUTE REPLY was generated or forwarded. These precursors will receive notifications from the node when the next hop link is detected as broken. Each time a route table entry is used, its lifetime is updated to be the current time plus a fixed parameter. When a route table entry is expired, the precursor list associated with the entry will be removed. The precursor list is designed with a similar goal as our ReplyRecord, but there are two main differences between precursors and our mechanism. First, we do not use timeouts for cache table entries and ReplyRecord entries. Second, precursors keep track of the nodes recently using some route; once a route table entry is expired, precursors that have not used or did not recently use that route will not be tracked. In contrast, our mechanism completely keeps track of topology propagation state in a distributed manner.

3. The Dynamic Source Routing Protocol

A. Overview of DSR

DSR consists of two on-demand mechanisms: Route Discovery and Route Maintenance. When a source node wants to send packets to a destination to which it does not have a route, it initiates a Route Discovery by broadcasting a ROUTE REQUEST. The node receiving a ROUTE REQUEST checks whether it has a route to the destination in its cache. If it has, it sends a ROUTE REPLY to the source including a source route, which is the concatenation of the source route in the ROUTE REQUEST and the cached route. If the node does not have a cached route to the destination, it adds its address to the source route and rebroadcasts the ROUTE REQUEST. When the destination receives the ROUTE REQUEST, it sends a ROUTE REPLY containing the source route to the source. Each node forwarding a ROUTE REPLY stores the route starting from itself to the destination. When the source receives the ROUTE REPLY, it caches the source route.

In Route Maintenance, the node forwarding a packet is responsible for confirming that the packet has been successfully received by the next hop. If no acknowledgement is received after the maximum number of retransmissions, the forwarding node sends a ROUTE ERROR to the source, indicating the broken link. Each node forwarding the ROUTE ERROR removes from its cache the routes containing the broken link.

B. Route Caching in DSR

DSR uses path caches [1] or link caches [4]. In a path cache, a node stores each route starting from itself to another node. In a link cache, a node adds a link to a topology graph, which represents the node's view of the network topology. Links obtained from different routes can form new routes. Thus, link caches provide more routing information than path caches.

A node learns routes through forwarding ROUTE REPLIES and data packets, or by overhearing packets when promiscuous mode is used [10]. DSR does not cache the source route accumulated in a ROUTE REQUEST, since ROUTE REQUESTS are broadcast packets and thus links discovered may not be bi-directional [8]. Due to the same reason, when a node forwards a ROUTE REPLY, it caches only the links that have been confirmed by the MAC layer to be bi-directional [8], which are the downstream links starting from the node to a destination. When forwarding a data packet, a node caches the upstream links as a separate route. After initiating a Route Discovery, a source node may learn many routes returned either by intermediate nodes or by the destination; it will cache all those routes. Thus, DSR aggressively caches and uses routing information.

Besides Route Maintenance, DSR uses two mechanisms to remove stale routes. First, a source node piggybacks on the next ROUTE REQUEST the last broken link information, which is called a GRATUITOUS ROUTE ERROR. Although this optimization helps remove stale routes from more caches, GRATUITOUS ROUTE ERRORS are not able to reach all nodes whose caches contain the broken link, because some ROUTE REQUESTS will not be further propagated due to the use of responding to ROUTE REQUESTS with cached routes. Second, DSR uses heuristics: a small cache size with FIFO replacement for path caches and adaptive timeout mechanisms for link caches [4], where link timeouts are chosen based on observed link usages and breakages.

4. The Distributed Cache Modernizing Algorithm

In this section, we first describe the cache staleness issue. We then give the definition of a cache table and present two algorithms used to maintain the information for cache updates. Finally, we describe our distributed cache update algorithm in detail.

A. Problem Statement

On-demand Route Maintenance results in delayed awareness of mobility, because a node is not notified when a cached route breaks until it uses the route to send packets. We classify a cached route into three types:

- pre-active, if a route has not been used;
- active, if a route is being used;
- post-active, if a route was used before but now is not.

It is not necessary to detect whether a route is active or post-active, but these terms help clarify the cache staleness issue. Stale pre-active and post-active routes will not be detected until they are used. Due to the use of responding to ROUTE REQUESTS with cached routes, stale routes may be quickly propagated to the caches of other nodes. Thus, pre-active and post-active routes are important sources of cache staleness.

We show an example of the cache staleness issue. In Figure 1, assume that route ABCDE is active, route FGCDH is post-active, and route IGCDJ is pre-active. Thus, node C has cached both the upstream and the downstream links for the active and post-active routes, but only the downstream links, CDJ, for the pre-active route. When forwarding a packet for source A, node C detects that link CD is broken. It removes stale routes from its cache and sends a ROUTE ERROR to node A. However, the downstream nodes, D and E, will not know about the broken link. Moreover, node C does not know that other nodes also have cached the broken link, including all the nodes on the post-active route, F, G, D, and H, and the upstream nodes on the pre-active route, I and G.

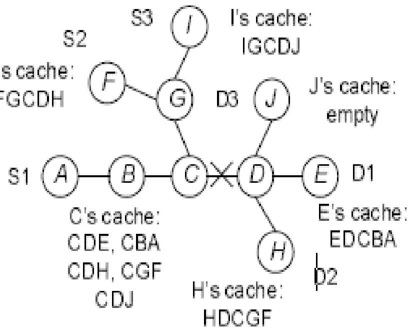

Fig.1. An Example of Routing Caching in DSR.

B. Assumption

Promiscuous mode [10] disables the network interface's address filtering function and thus causes a protocol to receive all packets overheard by the interface. Since it is impossible to know which neighbor overhears which link, we do not maintain such information in a cache table. To handle promiscuous mode, we use a secondary cache to store overhead routes, without any modification to the cache update algorithm.

C. The Definition of a Cache Table

It was shown that no single cache size provides the best performance for all mobility scenarios [4]. Thus, we design a cache table that has no capacity limit. Without capacity limit allows DSR to store all discovered routes and thus reduces route discoveries. The cache size increases as new routes are discovered and decreases as stale routes are removed.

There are four fields in a cache table entry:

- Route: It stores the links starting from the current node to a destination or from a source to a destination.
- SourceDestination: It is the source and destination pair.
- DataPackets: It records whether the current node has forwarded 0, 1, or 2 data packets. It is 0 initially, incremented to 1 when the node forwards the first data packet, and incremented to 2 when it forwards the second data packet.
- ReplyRecord: This field may contain multiple entries and has no capacity limit. A ReplyRecord entry has two fields: the neighbor to which a ROUTE REPLY is forwarded and the route starting from the current node to a destination. A ReplyRecord entry will be removed in two cases: when the second field contains a broken

link, and when the concatenation of the two fields is a sub-route of the source route, which starts from the previous node in the source route to the destination of the data packet.

5. Performance Evaluation

A. Evaluation Methodology

We compared our algorithm called DSR-Update to DSR with path caches and with Link-MaxLife under both promiscuous and non-promiscuous mode. When promiscuous mode (also called tapping) was not used, we did not use GRATUITOUS ROUTE REPLIES since it relies on this mode. For DSRUpdate without promiscuous mode, we did not use GRATUITOUS ROUTE ERRORS, since we wanted to use the algorithm as the only mechanism to remove stale routes. When promiscuous mode was used, we used all optimizations for the three caching strategies.

B. Simulation Results

1) Packet Delivery Ratio: Figure 13 (a)–(c) show packet delivery ratio. Without promiscuous mode, DSR-Update outperforms DSR with path caches by up to 19% and Link-MaxLife by up to 41%. The improvement increases as mobility, traffic load, or network size increases. As mobility increases, more routes will become stale; therefore, the advantages of fast cache updating become more significant. As traffic load increases, stale routes will adversely affect more traffic sources; proactive cache updating reduces packet losses from more sources. Proactive cache updating is also important for large networks, because as network size increases, more nodes will cache stale routes.

2) Packet Delivery Latency: Figure 14 shows packet delivery latency. Without promiscuous mode, DSR-Update reduces latency by up to 54% of DSR with path caches. Since detecting link failures is the dominant factor of delivery latency, the reduction in latency further demonstrates the effectiveness of the algorithm. Moreover, the reduction increases as mobility, traffic load, or network size increases, because quick removing stale routes reduces link failure detections by multiple flows.

6. Conclusions

In this paper, we presented the first work that proactively updates route caches in an adaptive manner. We defined a new cache structure called a cache table to maintain the information necessary for cache updates. We presented a distributed cache update algorithm that uses the local information kept by each node to notify all reachable nodes that have cached a broken link. The algorithm enables DSR to adapt quickly to topology changes.

We show that, under non-promiscuous mode, the algorithm outperforms DSR with path caches by up to 19% and DSR with Link-MaxLife by up to 41% in packet delivery ratio. It reduces normalized routing overhead by up to 35% for DSR with path caches. Under promiscuous mode, the algorithm improves packet delivery ratio by up to 7% for both caching strategies, and reduces delivery latency by up to 27% for DSR with path caches and 49% for DSR with Link-MaxLife. The improvement demonstrates the benefits of the algorithm. Although the results were obtained under a certain type of mobility and traffic models, we believe that the results apply to other models, as the algorithm quickly removes stale routes no matter how nodes move and which traffic model is used.

The central challenge to routing protocols is how to efficiently handle topology changes. Proactive protocols periodically exchange topology updates among all nodes, incurring significant overhead. On-demand protocols avoid such overhead but face the problem of cache updating. We show that proactive cache updating is more efficient than adaptive timeout mechanisms. Our work combines the advantages of proactive and on-demand protocols: on-demand link failure detection and proactive cache updating. Our solution is applicable to other on-demand routing protocols. We conclude that proactive cache updating is key to the adaptation of on-demand routing protocols to mobility.

References

[1] X. Yu and Z. Kedem. A distributed adaptive cache update algorithm for the Dynamic Source Routing protocol. In Proc. 24th IEEE INFOCOM, March 2013. (An earlier version appeared as NYU CS Technical Report TR2003-842, July 2011.)

[2] The Monarch Project. Mobile networking architectures.http://www.monarch.cs.rice.edu/.

[3] C. Perkins, E. Royer, S. Das, and M. Marina. Performance comparison of two on-demand routing protocols for ad hoc networks. IEEE Personal Communications, 8(1): 16–28, 2010.

[4] C. Perkins, E. Royer, and S. Das. Ad hoc On-demand Distance Vector (AODV) Routing, RFC 3561. http://www.ietf. org/rfc/rfc3561.txt, July 2009.

[5] C. Perkins and E. Royer. Ad hoc on-demand distance vector routing. In Proc. 2nd WMCSA, pp. 90–100, 2005.

[6] M. Marina and S. Das. Performance of routing caching strategies in Dynamic Source Routing. In Proc. 2nd WNMC, pp. 425–432, 2001.

[7] D. Maltz, J. Brooch, J. Jetcheva, and D. Johnson. The effects of on-demand behavior in routing protocols for multi-hop wireless ad hoc networks. IEEE J. on Selected Areas in Communication, 17(8):1439–1453, 1999.

[8] W. Lou and Y. Fang. Predictive caching strategy for on-demand routing protocols in wireless ad hoc networks. WirelessNetworks, 8(6): 671–679, 2002.

[9] D. Johnson, D. Maltz, and Y.-C. Hu. The Dynamic Source Routing for mobile ad hoc networks, IETF Internet Draft. http://www.ietf.org/internet-drafts/draft-ietf-manet-dsr-10.txt, July 2004.

[10] Y.-C. Hu and D. Johnson. Ensuring cache freshness in on-demand ad hoc network routing protocols. In Proc. 2nd POMC, pp. 25–30, 2002.

[11] D. Johnson and D. Maltz. Dynamic Source Routing in ad hoc wireless networks. In Mobile Computing, T. Imielinski and H.Korth, Eds, Ch. 5, pp. 153–181, Kluwer, 1996.

[12] IEEE Computer Society LAN MAN Standards Committee. Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications, IEEE Std 802.11-1997. The IEEE, New York, New York, 1997.

[13] Y.-C. Hu and D. Johnson. Caching strategies in on-demand routing protocols for wireless ad hoc networks. In Proc. 6th ACM MobiCom, pp. 231–242, 2000.

[14] G. Holland and N. Vaidya. Analysis of TCP performance over mobile ad hoc networks. In Proc. 5th ACM MobiCom, pp. 219–230, 1999.

[15] K. Fall and K. Varadhan, Eds. ns notes and documentation. The VINT Project, UC Berkeley, LBL, USC/ISI, and Xerox PARC, 1997.

[16] 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 Proc. 4th ACM MobiCom, pp. 85–97, 1998.