Ensuring Cache Freshness in On-Demand Ad Hoc Network Routing Protocols

Yih-Chun Hu
Carnegie Mellon University
Pittsburgh, PA, USA
yihchun@cs.cmu.edu

David B. Johnson
Rice University
Houston, TX, USA
dbj@cs.rice.edu

ABSTRACT

In a wireless ad hoc network, nodes cooperate to forward packets for each other over possibly multi-hop paths, to allow nodes not within direct wireless transmission range to communicate. Many routing protocols have been proposed for the ad hoc network environment, several of which operate on-demand and utilize a *route cache* listing links that this node has learned. In such protocols, aggressive caching of overheard routes can significantly improve performance; in particular, overhead can be reduced by leveraging information received in packets overheard or forwarded from other nodes, including other routing packets and the source routes on other data packets. Unfortunately, such information sharing can substantially increase the risk of cache cross-pollution, since stale routing information in one node's cache, representing a link that no longer exists, can easily be added into the caches of other nodes. Even when a node has actually learned that a link no longer exists, it is still possible for that node to again hear the stale information. In this paper, we present a new mechanism which we call *epoch numbers*, to reduce this problem of cache staleness, by preventing the re-learning of stale knowledge of a link after having earlier heard that the link has broken. Our scheme does not rely on ad hoc mechanisms such as short-lived negative caching; rather, we allow a node having heard both of a broken link and a discovery of the same link to sequence the two events in order to determine whether the link break or the link discovery occurred before the other.

Categories and Subject Descriptors

C.2.1 [**Computer- Communication Networks**]: Network Architecture and Design—wireless communication, packet-switching networks; C.2.2 [**Computer- Communication Networks**]: Network Protocols—routing protocols; E.1 [**Data Structures**]: Distributed Data Structures

General Terms

Algorithms, Design, Reliability

Keywords

DSR, Dynamic Source Routing, ad hoc networks, epoch numbers, route cache, theory, bounded latency

1. INTRODUCTION

In a wireless ad hoc network, mobile nodes cooperate by forwarding packets for each other to allow communication between nodes not otherwise in direct wireless transmission range. Ad hoc networks do not require any infrastructure such as base stations or access points, and do not require any centralized administration or control; the network is entirely self-organizing between the peer mobile nodes that form the network. Ad hoc networks hold promise in applications where no existing network infrastructure exists, or where the existing infrastructure cannot be used for reasons such as security, usage cost, or insufficient resources. For example, workers providing emergency relief following a natural disaster might communicate using an ad hoc network, as might soldiers in a military operation. Routing within such networks is a challenging problem, due to factors such as node mobility, limited wireless transmission range, wireless transmission interference, and changes in the wireless propagation environment.

Many proposed ad hoc network routing protocols, including DSR [10], LAR [12], SSA [2], and ABR [13], operate on-demand and use a *route cache* to choose routes; these protocols also use source routing, such that each node maintains a cache of all routes that it has previously discovered or overheard in other packets, and the sender of a packet chooses a route for each packet it wishes to send using routes from its route cache, possibly combining routing information learned in different ways over time. This use of caching can substantially reduce the overhead of the routing protocol and also reduces the latency in delivering data packets when a cached route is already available.

However, routing *cache staleness* presents a serious challenge to such protocols. Caches represent learned portions of the network topology, but a cache entry may become invalid due to changes such as two nodes moving out of wireless transmission range of each other; a node is not notified when one of its cache entries becomes invalid, unless the node actually attempts to use the cache entry in routing a packet it sends. Although a periodic routing protocol such as a link-state or distance-vector routing protocol could distribute updated information in a somewhat timely manner, periodic protocols have been shown to have higher overhead in a number of studies [1, 7], and periodic protocols still take some amount of time to detect a link failure and to distribute this information.

When a node uses information from its route cache that was learned from overheard packets, the cache staleness problem is compounded, since stale information could circulate in the network indefinitely. For example, one node may use some stale information to route a packet that it sends, allowing a number of nodes to overhear that packet and to also cache that stale routing information. If any node that overheard the use of the route
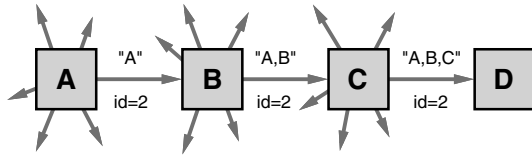
Figure 1: An example of DSR Route Discovery. Node A is discovering a route to node D. Each node forwards the ROUTE REQUEST from A, adding its own address to the list in the packet; the combination of the initiator address (A), the target address (D), and the request identifier (2) assigned by node A uniquely identifies this Route Discovery.

does not subsequently overhear the corresponding route breakage notification, that node will be left with a stale link in its route cache, which it may later use in routing its own packets.

In this paper, we present a new mechanism, which we call *epoch numbers*, for reducing the amount of stale information in each node's route cache. We achieve this improved cache management by preventing a node from re-learning stale information about a link after having earlier heard that this link has broken. Our scheme does not rely on ad hoc mechanisms such as short-lived negative caching of routing information; rather, we allow a node, having heard that a link has broken and having heard a discovery of the same link, to sequence the two events and determine whether the link break or the link discovery occurred more recently than the other.

First, in Section 2 of this paper, we summarize the operation of the Dynamic Source Routing protocol (DSR), as an example ad hoc network routing protocol, in which we describe our work. Section 3 describes our approach using epoch numbers, Section 4 provides an analysis of the properties our approach achieves, and Section 5 presents conclusions.

2. OVERVIEW OF THE DYNAMIC SOURCE ROUTING PROTOCOL (DSR)

This section provides an overview of the Dynamic Source Routing protocol (DSR) [8, 9, 10] as an example ad hoc network routing protocol, on which we base our development of epoch numbers for ensuring routing cache freshness. DSR is one of a number of routing protocols proposed within the Mobile Ad Hoc Networks (MANET) Working Group of the Internet Engineering Task Force (IETF) [6], the principal protocol standards development body for the Internet. Similar techniques for solving cache staleness by sequencing link detection and breakage information could be applied to other ad hoc network routing protocols.

The operation of DSR is based on *source routing*, in that the sender of a packet determines the complete sequence of hops to be used as the route for that packet to its destination. In the basic version of DSR, the source route for each packet is represented in the header of the packet.

DSR divides the routing problem in two parts, *Route Discovery* and *Route Maintenance*, both of which operate *entirely* on-demand. In Route Discovery, a node actively searches through the network to find a route to an intended destination node. While using a route to send packets to the destination, Route Maintenance is the process by which the sending node determines if the route has broken, for example because two nodes along the route have moved out of wireless transmission range of each other.

A node that has a packet to send to some destination searches its *route cache* for a route to that destination. If no cached route is found, the sending node initiates Route Discovery by locally broadcasting a ROUTE REQUEST packet containing the destination node address (known as the *target* of the Route Discovery), a list (initially empty) of nodes traversed by this ROUTE REQUEST, and a *request identifier* from this source node. The request identifier, the address of this source node (known as the *initiator* of the Route Discovery), and the target address together uniquely identify this Route Discovery. An example of DSR Route Discovery is shown in Figure 1.

A node receiving a ROUTE REQUEST checks to see if it has previously forwarded a REQUEST from this Discovery by examining the IP Source Address, target address, and request identifier. If it has recently seen this identifier, or if its own address is already present in the list of nodes traversed by this REQUEST, the node silently drops the packet. Otherwise, the node appends its address to the node list in the REQUEST and forwards the REQUEST. When a REQUEST reaches the target node or a node with a route to the target in its route cache, this node returns a ROUTE REPLY to the initiator of the ROUTE REQUEST. The ROUTE REPLY contains a copy of the node list from the REQUEST, and can be delivered to the initiator node by reversing the node list, by using a route back to the initiator from its own route cache, or by "piggybacking" the REPLY on a new ROUTE REQUEST targeting the original initiator. When the initiator of the request receives the ROUTE REPLY, it adds the newly acquired routing information from the REPLY to its route cache for future use.

In Route Maintenance, a node forwarding a packet for a source attempts to confirm that the packet successfully reached the next hop in the route. A node can make this confirmation using a link-layer acknowledgement (such as is provided in IEEE 802.11 [5]), a passive acknowledgement [11], or by means of a network-layer acknowledgement. A packet is possibly retransmitted if it is sent over an unreliable Medium Access Control (MAC) layer, although it should not be retransmitted if retransmission has already been attempted at the MAC layer. If a packet is not acknowledged as described above, the forwarding node assumes that the next-hop destination is unreachable over this link, and sends a ROUTE ERROR to the source of the packet, indicating the broken link. A node receiving a ROUTE ERROR removes that link from its route cache. An example of DSR Route Maintenance is shown in Figure 2.

A number of optimizations to the basic DSR protocol have been proposed [9, 10]. In this paper, we describe only those optimizations that are affected by the changes we make to the protocol. One example of such an optimization is *packet salvaging*. When a node forwarding a packet fails to receive confirmation that the packet has been received by the next-hop destination, in addition to sending a ROUTE ERRORto the source of the packet, the node may attempt to use an alternate route to the destination, if it knows of one. Specifically, the node searches its own route cache for a route to

Figure 2: An example of DSR Route Maintenance. When forwarding a packet from node A to node D, node C detects that the next link along the source route (the link from C to D) is broken and returns a ROUTE ERROR to node A, the original sender of the packet.

the destination; if it finds one, then it *salvages* the packet by replacing the existing source route for the packet with the new route from its route cache. To prevent the possibility of infinite looping of a packet, each source route includes a *salvage count*, indicating how many times the packet has been salvaged in this way. Packets with salvage count larger than some predetermined value cannot be salvaged again.

Another optimization allows nodes in DSR to learn new information by overhearing the source routes on packets sent by other nodes. This optimization has been shown to significantly improve the performance of DSR [15]. Unfortunately, this optimization also provides a means of further spreading invalid cache information, which can result in decreased routing performance [15, 3].

In this paper, we focus on reducing the invalid cache information spread through overhearing of invalid links in routing packets and in source routes. In particular, we focus on those invalid links that have already been found to be broken. Other work has focused on reducing the number of links in the cache that entered the cache while the link was valid but have become invalid through network mobility [3], and also on preventing the discovery of links that are soon to expire [2, 4, 13]. Some previous work has used ad hoc techniques, such as limited-lifetime negative cache information, to reduce the chance of spreading incorrect information [16], or through multipath routing, to increase the probability of successful delivery in the presence of failed links [18, 17]. In this work, however, we focus on the cause of cache corruption, and we identify one piece of information that can eliminate much of the problem. We then propose how that information can be added to an ad hoc network routing protocol using entirely "soft state" such that the loss of the state, for example due to node failure, does not harm the correct operation of the protocol.

3. PROPOSED APPROACH

3.1. Idealized Epoch Numbers

Our approach to reducing the amount of invalid information in the cache is to have enough information to correctly sequence new link discovery and link breakage information. When new link discovery and link breakage information can be correctly sequenced, stale information cannot re-enter the cache after it has been deleted. For example, if a link is down, and a node has discovered that it is down, another node's stale cache entry can be determined to be stale.

First, we will describe an idealized version of *epoch numbers*, which we use to perform this sequencing. Our idealized model does not take into account the finite size of integer representations. In Section 3.2, we present a solution to integer wraparound based on what we call *generation numbers*, which provide a compact representation of how long a piece of information has persisted in route caches throughout the network, without the need for synchronized clocks or relatively large timestamp information.

To sequence conflicting link status information, each node maintains an *epoch number* for itself, which it associates with each link to a newly detected neighbor and with each new link break message that this node itself sends. For example, if during node A's epoch number i, node A discovers that node B is a neighbor, then node A associates the link $A \to B$ with the epoch number i (independent of the epoch number at node B). Each route cache includes both positive (discovered link) information and negative (broken link) information, together with the associated epoch number of each link; each packet containing routing information also includes the associated epoch number of the information from the node's cache.
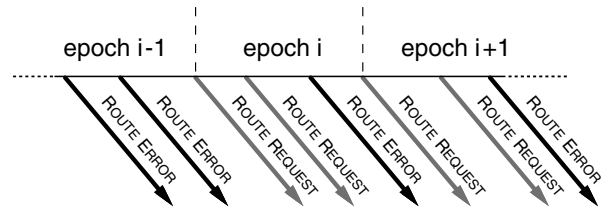

Figure 3: Advancement of the epoch number at a node. Each time a node first originates or forwards a ROUTE REQUEST after it has originated a ROUTE ERROR, the node increments its epoch number. The figure shows the epoch number of a single node, with time progressing from left to right.

Links are sequenced using epoch numbers from the sending side, since it is easier to sequence broken link (ROUTE ERROR) messages at the sending side.

When a node has a link cached that conflicts with a newly heard message containing routing information, the node chooses to trust the information with the higher epoch number. When a link detection and a link breakage have the same epoch number, the link breakage information takes precedence. The epoch number is increased as necessary to maintain proper sequencing, as illustrated in Figure 3. In particular, whenever a node first originates or forwards a ROUTE REQUEST packet after it has originated a ROUTE ERROR, the node increments its own epoch number. Related approaches have been used previously in distributed systems such as for ordering events or states within the system.

As an example of the use of epoch numbers, suppose node A has a link cached from node B to node C with epoch number 1 and uses this link as part of a route. Then node A would only accept a ROUTE ERROR for this link with epoch number greater than or equal to 1. Once A hears such a ROUTE ERROR, for example with epoch number 2, node A will ignore any link from B to C with epoch number less than or equal to the epoch number in the ROUTE ERROR (2, in this example). If A hears such a link, it will not add that link to its cache. This prevents A's cache from being polluted with stale information once it has learned fresher information.

3.2. Provisions for Wraparound

Since epoch numbers are integers with finite length, yet they propagate through the network, we need a mechanism for dealing with the potential for integer wraparound.

First, we limit the rate at which new epoch numbers are used. Since our goal is to limit the rate at which the epoch number space can wrap around, and because of the way in which epoch numbers are compared, the rate limit can be an amortized rate limit over half the epoch number space. For example, we may specify a maximum rate of 128 epoch numbers per second, and if the epoch numbers are 25 bits long, then we are effectively setting a limit of 2^{24} epoch numbers over 2^{24-7} seconds (about 1.5 days).

Second, we need to allow for proper timeouts on epoch information. To do this at a node hearing the information directly, for example in a ROUTE REQUEST or ROUTE REPLY, we simply set a timeout after which the epoch number for that information is no longer valid. We do not require that the link be discarded after the epoch number is invalid; we merely flag the link to indicate that it is older than *all* epoch numbers. This entry can be flagged, for example, by setting the epoch number to 0, and ensuring that nodes do not use 0 as a valid epoch number. At intermediate nodes, we

need a way to know how old an epoch number is, in order to set this timeout correctly. To limit network overhead, we use a conservative approximation based on a *generation number*. We choose a limited generation number space and specify that each node increase the generation number as the information ages. For example, if the epoch number is valid for 131072 seconds, and there are 128 numbers in the generation number space (7 bits), then one generation is consumed every 1024 seconds.

Whenever a packet is sent with a source route, the node inserts into the source route in the packet an epoch and generation number for each link in the source route. The epoch number is chosen to be the epoch number corresponding to the link in the node's Route Cache. To choose the generation number, the node conservatively estimates τ, the time (at this node) of the arrival of this packet at the last node hearing this packet. The node then finds the generation number for that link at that time, increases it by one, and uses that as a generation number for that link in the source route.

For example, suppose a node holds a link for which intends to increment the generation number one second later, and the node sends a packet along a route using that link. If the node estimates the end-to-end delay for this packet to be less than one second, the packet's generation number for that link will be 1 greater than the generation number in the cache. If the node estimates the end-to-end delay as being between one and two seconds, the generation number chosen for the packet will be 2 greater than the generation number in the cache. Whenever the generation number is increased, the node checks if the number has overflowed. If so, the epoch number is set to a special value (such as 0), and nodes hearing this link can cache it only if it does not conflict with a previously held broken link.

The end-to-end delay estimates required by the protocol can be provided by the network, for example using QoS extensions to DSR [14]. It can also be chosen by multiplying a large factor (such as 10) times prior delay measurements to achieve a conservative estimate of the upper bound on end-to-end delay. Finally, in Section 3.4, we present a technique to provide hard guarantees on end-to-end delay in a wireless ad hoc network.

3.3. Reducing the Generation Number

In Section 3.2, we presented a very conservative approach to the use of generation numbers. However, the purpose of generation numbers is simply to allow an epoch number to expire before wraparound. As currently described, this is implemented by wrapping around the generation number before the epoch number could wrap around, based on the maximum rate at which epoch numbers can be consumed. Since reaching the maximum rate should be a rare occurrence, we describe a mechanism for compensating the stored generation numbers when epoch numbers are used at a slower rate.

We denote the number of bits allocated to the epoch number as e, the number of bits allocated to the generation number as g, and denote the maximum rate at which epoch numbers are consumed to be 2^r per second. Using the technique described above, a node increments the generation number for each link in its route cache every $2^{e-r-g-1}$ seconds. Given the conservative maximum rate, the expectation is that after n more epoch numbers are used at the source, the generation number at any node in the network is (at least)

$$\frac{n2^{-r}}{2^{e-r-g-1}} = n2^{g-(e-1)} .$$

If epoch numbers were actually used at a slower rate, it is possible to revise the generation number for earlier epoch numbers. For example, if a node has a link from some node A with epoch number x, and another link from the same node with epoch number $x + n$, the generation number of the epoch x link needs to be no more than the generation number of the epoch $x + n$ link plus

$$\left\lceil n2^{g-(e-1)} \right\rceil .$$

To implement this generation number reduction, when a node hears a new epoch number x (with generation number g) for some other node, the node searches its route cache for links originating from that other node for which the epoch number on that link is less than the new epoch number. For each such link, if that link has epoch number $x - n$ and generation number g', the generation number of that link can be set to

$$\min \left(g', g + \left\lceil n2^{g-(e-1)} \right\rceil \right) .$$

3.4. Hard Latency Guarantees for Networks

In this section, we focus on providing a bounded latency at each hop in an ad hoc wireless network; since DSR uses source routes, a hard bound on per-hop latency also provides a hard bound on end-to-end latency. (Although salvaging breaks this invariant, it can be restored by using the value of the IP Time-To-Live (TTL) field, rather than the length of the source route, to bound the end-to-end latency; furthermore, with epoch numbers, we are concerned only with the lifetime of the source routes themselves.) If there are no explicit mechanisms to provide hard bounds on latency, a packet with an old epoch number can be held in a transmission buffer at some node indefinitely, and when finally transmitted, may appear to be a very fresh epoch number. A node receiving a source route with such an epoch number would then disregard ROUTE ERROR messages on that link for a long time. A number of other ad hoc network routing protocols rely on such bounds in network transmission time; the mechanism described here can be used in conjunction with such protocols to ensure correctness.

To place a hard bound on the time a packet can spend traversing a node, we assume that the communication time between the processor and any network interface at a node is bounded by a known time t_{net}. We also assume that a timer can be set that allows an interrupt handler to be called at the node within a fixed amount of time t_{inter} after the set time, and that each network interface at a node supports a reset operation, which discards any packets contained within that network interface's transmission buffer within a bounded amount of time t_{reset}. Finally, we assume a limited, known maximum decoding time between when a packet finishes arriving at a node and when the node can pass it up to the network layer.

We limit the amount of time a packet can spend in software interface queues by timestamping each packet as it enters the queue and removing any packets that are too old before they can enter a hardware queue. We then limit the amount of time that the packet can spend in a hardware queue by setting a timer and canceling it if the packet is successfully transmitted. If the timer expires, the network interface is reset, limiting the hardware latency to

$$t_{\text{hw}} + 2t_{\text{net}} + t_{\text{inter}} + t_{\text{reset}} ,$$

where t_{hw} is the timer duration set when a packet first enters the hardware interface. Finally, we assume that the nodes can know some transmission latency bound, based on the minimum transmission bit rate and a maximum transmission range. These three factors can be bounded, even in the presence of fail-stop failures.

Many of these assumptions can be relaxed in an ad hoc network if only one network interface at each node is used for routing, and if that network interface maintains an accurate clock. (More precisely, if packets are only forwarded on their incoming network

interface, this algorithm can work.) Each packet passed from the network interface to the processor is stamped with the time read from the clock in the network interface; a packet to be sent can also contain a time after which the network interface should not transmit the packet. To impose a limit on processing time and Medium Access Control (MAC) contention time, the node simply adds the maximum acceptable processing time and contention time to the value received from the network interface and uses that time as the limit. Since the time limit and the receive time are derived from the same clock, any amount of delay in the asynchronous network between the processor and network interface does not affect such a bound. The transmission latency can be bounded as described previously.

3.5. Proactive ROUTE ERRORS

In general, detecting a broken link is fairly expensive, due to the number of retransmissions required before a node can determine that the link has broken. Also, many different MAC layers use an exponential backoff between consecutive retransmission attempts, resulting in significant latency. To reduce such unnecessary additional link breakage detection latency, a forwarding node can check each link against its cache for freshness. If any link in the source route has an epoch number less than or equal to the epoch number of that broken link in the forwarding node's cache, the forwarding node returns a *gratuitous* ROUTE ERROR to the source of the packet and discards the packet.

4. ANALYSIS

4.1. Network Overhead

To use epoch and generation numbers for all cache entries, we include such numbers for each address in each source route, ROUTE REQUEST, ROUTE REPLY, and ROUTE ERROR packet. As a result, network overhead increases by approximately the ratio between the size of the epoch information and the size of the network address. For example, if a network address is 32 bits, the use of a 7-bit generation number and 25-bit epoch number represents a doubling of this overhead. If epoch numbers are rate-limited to 128 per second, then half of the epoch number space is consumed in 131072 seconds, and each generation number can be used for 1024 seconds. This allows a single link to be overheard as many as 64 generations after when it was learned, even if as much as 17 minutes elapses between each generation (from learning the link to passing along knowledge of the link).

4.2. Storage Overhead

Our epoch number scheme adds a constant factor to the storage overhead in a link-state cache [3], since each link must be represented. Since the amount of information stored per link, including the epoch information, is constant, this additional information is only a constant factor more than the overhead already required by a link-state cache, which is $O(V)$, where V is the number of links.

4.3. Theoretical Analysis

The following two properties hold for a route cache using perfect epoch information (that is, epoch information with unbounded epoch numbers and with no timeout).

First, if a node has heard a ROUTE ERROR for some link, then the node will not add that link back to its route cache as a result of

the node overhearing routing information that was originally discovered before the ROUTE ERROR was sent. A node discovers the existence of a new link to a neighbor node as a result of originating or forwarding a ROUTE REQUEST that is received by that neighbor. When the node originates or forwards this REQUEST, the node begins a new epoch if it has sent any ROUTE ERRORs during its current epoch, and the REQUEST is the first action by the node in this new epoch. The epoch number included in the REQUEST is thus greater than the epoch number included in the node's earlier ROUTE ERROR (that reported that its link to that neighbor had broken). As the ROUTE ERROR information and the knowledge of the new link (from the ROUTE REQUEST) are learned by other nodes, the relative order of the two pieces of information can always be determined correctly by comparing these two epoch numbers from the node that assigned them.

Second, after hearing a ROUTE ERROR for some link and removing that link from its route cache, if the node overhears routing information indicating the existence of that link, the node will add that link to its route cache if that routing information was originally discovered after the ROUTE ERROR was sent. As above, the epoch number included in the ROUTE ERROR and the epoch number associated with the new link (from the ROUTE REQUEST by which it was discovered) are assigned by the same node (the node at the leading edge of that link). As the ROUTE ERROR information and the knowledge of the new link are learned by other nodes, the relative order of the two pieces of information is clear from the order of the two epoch numbers. If the epoch number for the new link is greater than the epoch number for the ROUTE ERROR, then it was discovered by the node at the leading edge of that link after that same node had previously detected the link as broken, indicating that this information is more recent and can be added to the route cache.

When the techniques described in Section 3.3 are not used, certain properties are obtained even when epoch numbers can wrap around, and when generation numbers are used to maintain correctness. In particular, a ROUTE ERROR that is directly heard will not have a higher generation number than any link detected before it, and therefore the epoch number of any directly heard ERROR will not be set to 0 until all the previously learned links (whether directly or indirectly) have also had their epoch number set to 0. As a result, directly heard ERRORs have precedence over all older link discoveries.

5. CONCLUSION

In this paper, we have presented new a solution to the cache staleness problem in on-demand ad hoc network routing protocols such as DSR [10], LAR [12], SSA [2], and ABR [13]. Our solution performs as well as we could hope: in particular, stale information generally cannot override fresh information. We did this without relying on a negative cache, and without significantly increasing the network overhead or changing the basic functionality of the underlying protocol. In fact, our modifications do not directly increase the packet overhead. Our solution works by providing exactly the information needed to sequence discovery of new network links and notification of broken links, thus preventing nodes from re-learning stale cache information.

In addition, we have also discussed techniques for providing hard latency limits on transmissions in an ad hoc network. Such techniques can also be applied to other protocols that require such hard limits. This approach can also easily be generalized to support nodes having more than one network interface.

REFERENCES

[1] Josh Broch, David A. Maltz, David B. Johnson, Yih-Chun Hu, and Jorjeta Jetcheva. A Performance Comparison of Multi-Hop Wireless Ad Hoc Network Routing Protocols. In *Proceedings of the Fourth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom'98)*, pages 85–97, October 1998.

[2] Rohit Dube, Cynthia D. Rais, Kuang-Yeh Wang, and Satish K. Tripathi. Signal stability-based adaptive routing (SSA) for ad hoc mobile networks. *IEEE Personal Communications*, 4(1):36–45, February 1997.

[3] Yih-Chun Hu and David B. Johnson. Caching Strategies in On-Demand Routing Protocols for Wireless Ad Hoc Networks. In *Proceedings of the Sixth Annual International Conference on Mobile Computing and Networking (MobiCom 2000)*, August 2000.

[4] Yih-Chun Hu and David B. Johnson. Design and Demonstration of Live Audio and Video over Multihop Wireless Ad Hoc Networks. In *Proceedings of the MILCOM 2002 IEEE Military Communications Conference*, October 2002. To appear.

[5] IEEE Computer Society LAN MAN Standards Committee. *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, IEEE Std 802.11-1997*. The Institute of Electrical and Electronics Engineers, New York, New York, 1997.

[6] Internet Engineering Task Force MANET Working Group. Mobile Ad Hoc Networks (MANET) Charter. Available at http://www.ietf.org/html.charters/manet-charter.html.

[7] Per Johansson, Tony Larsson, Nicklas Hedman, Bartosz Mielczarek, and Mikael Degermark. Scenario-based Performance Analysis of Routing Protocols for Mobile Ad-hoc Networks. In *Proceedings of the Fifth Annual International Conference on Mobile Computing and Networking (MobiCom 1999)*, pages 195–206, August 1999.

[8] David B. Johnson. Routing in Ad Hoc Networks of Mobile Hosts. In *Proceedings of the IEEE Workshop on Mobile Computing Systems and Applications (WMCSA'94)*, pages 158–163, December 1994.

[9] David B. Johnson and David 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.

[10] David B. Johnson, David A. Maltz, and Josh Broch. The Dynamic Source Routing Protocol for Multihop Wireless Ad Hoc Networks. In *Ad Hoc Networking*, edited by Charles E. Perkins, chapter 5, pages 139–172. Addison-Wesley, 2001.

[11] John Jubin and Janet D. Tornow. The DARPA Packet Radio Network Protocols. *Proceedings of the IEEE*, 75(1):21–32, January 1987.

[12] Young-Bae Ko and Nitin Vaidya. Location-Aided Routing (LAR) in Mobile Ad Hoc Networks. In *Proceedings of the Fourth Annual International Conference on Mobile Computing and Networking (MobiCom 1998)*, pages 66–75, October 1998.

[13] Sung-Ju Lee, Mario Gerla, and Chai-Keong Toh. A simulation study of table-driven and on-demand routing protocols for mobile ad hoc networks. *IEEE Network*, 13(4):48–54, July 1999.

[14] David A. Maltz. Resource Management in Multi-hop Ad Hoc Networks. Technical Report Technical Report CMU-CS-00-150, CMU School of Computer Science, November 1999.

[15] David A. Maltz, Josh Broch, Jorjeta Jetcheva, and David 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*, 17(8):1439–1453, August 1999.

[16] Mahesh K. Marina and Samir R. Das. Performance of Route Caching Strategies in Dynamic Source Routing. In *Proceedings of the 2nd Wireless Networking and Mobile Computing (WNMC)*, April 2001.

[17] Asis Nasipuri, Robert Castaneda, and Samir R. Das. Performance of Multipath Routing for On-Demand Protocols in Ad Hoc Networks. *ACM/Kluwer Mobile Networks and Applications (MONET) Journal*, 6(4):339–349, 2001.

[18] Aristotelis Tsirigos, Zygmunt J. Haas, and Siamak Tabrizi. Multipath Routing in Mobile Ad Hoc Networks or How to Route in the Presence of Topological Changes. In *Proceedings of the MILCOM 2001 IEEE Military Communications Conference*, October 2001.