Bypass Routing: An On-Demand Local Recovery Protocol for Ad Hoc Networks

Cigdem Sengul
University of Illinois Urbana-Champaign
e-mail: sengul@uiuc.edu

Robin Kravets
University of Illinois Urbana-Champaign
e-mail: rhk@cs.uiuc.edu

***Abstract*— On-demand routing protocols for ad hoc networks reduce the cost of routing in high mobility environments. However, route discovery in on-demand routing is typically performed via network-wide flooding, which consumes a substantial amount of bandwidth. In this paper, we present *bypass routing*, a local recovery protocol that aims to reduce the frequency of route request floods triggered by broken routes. Specifically, when a broken link is detected, a node patches the affected route using local information, which is acquired on-demand, and thereby bypasses the broken link. We implemented SLR (Source Routing with Local Recovery) as a prototype of our approach. Simulation studies show that SLR achieves efficient and effective local recovery while maintaining acceptable overhead.**

***Index Terms*— Ad hoc networks, routing, local recovery.**

I. Introduction

An ad hoc network is formed by a group of wireless devices without depending on any infrastructure. Each node communicates directly with its neighbors and functions as a router that forwards packets for nodes that are not within transmission range of the sender. Maintaining communication in ad hoc networks requires effective routing mechanisms in the presence of dynamic topology, which may cause route failures and require discovery of new routes. Therefore, the challenge to routing in dynamic ad hoc networks stems from the need to maintain routes while minimizing overhead from such maintenance.

Routing protocols for ad hoc networks can be categorized as proactive or reactive (on-demand) based on when routes are discovered. Proactive protocols [1] maintain up-to-date routing information regardless of the presence of traffic, and so consume valuable resources such as bandwidth and power even if the network is idle. On-demand routing protocols have been shown to reduce routing overhead in high mobility environments by only maintaining actively used routes [2], [3]. Although on-demand routing protocols only initiate route discovery when a route is needed, such discovery is typically performed via network-wide flooding. Since flooding consumes a substantial amount of bandwidth, it is essential to reduce the frequency of route discoveries, and so network flooding.

To overcome performance problems from frequent route discovery attempts, hybrid protocols incorporate both reactive and proactive protocol characteristics [4]. Although hybrid protocols do not waste resources by flooding the network for each route request, it is difficult to balance the cost of exchanging routing information periodically (i.e., proactivity) and network-wide flooding for route discovery (i.e., reactivity) [5]. Other protocols reduce the frequency of flooding by allowing a relay node to initiate a limited route discovery in the event of a route failure [6], [7] or employ local error recovery mechanisms [8], [9]. However, protocols using either limited broadcast or local error recovery have focused on reducing packet drops and not on utilizing the bandwidth efficiently during route recovery. Multipath routing protocols cache multiple routes to a destination in a single route discovery [10], [11], [12]. However, in the presence of mobility, multipath protocols incur additional packet drops and delay due to their dependency on potentially stale routes from caches. Based on these observations, our goal is to design an efficient on-demand local recovery protocol that reduces route request floods due to route failures without incurring any overhead from proactive exchanges.

The contribution of our research is *bypass routing*, which effectively localizes reaction to route failures using on-demand local recovery and a novel cache invalidation mechanism. Essentially, *bypass recovery* uses link-state information to find a patch between one of the neighbors and a node along the route to the destination, bypassing the link that caused the broken route. Different than other route repair techniques [8], our approach acquires link-state information of the neighborhood of a broken link on-demand. Therefore, bypass recovery ties local recovery to up-to-date information about a node's neighborhood, and hence increases the chance of recovering a broken route compared to using potentially stale routes (or link state information) from caches. The key benefit of bypass recovery comes from this localization of route re-

covery to the one hop neighborhood of the broken link, which reduces the impact of route recovery on the rest of the network. Additionally, the on-demand nature of bypass recovery ensures no extra overhead is incurred during normal operation of the network. Finally, an improved route invalidation mechanism that closely ties route validity to up-to-date neighbor information is integrated into route caching, enabling the early identification of broken routes. Through the localization of recovery and the improved caching, bypass routing is an effective and efficient approach to recovery from route failures. Results of extensive simulations based on our prototype SLR (Source Routing with Local Recovery) show that SLR enables fast recovery of broken routes, increasing the packet delivery ratio while maintaining acceptable overhead.

The rest of the paper is organized as follows. In Section II, we give a brief overview of proactive and reactive protocols and discuss existing approaches for local recovery in ad hoc networks. In Section III, we present bypass routing and describe our prototype SLR in Section IV. We demonstrate the effectiveness of bypass recovery in the context of SLR via simulation in Section V and further compare the results from the evaluation of SLR to the published results of AODV-BR (Ad Hoc On-Demand Distance Vector - Backup Routing) [13] and NSR (Neighborhood-aware Source Routing) [8] in Section VI. Section VII concludes with future work.

II. Local Recovery in Ad Hoc Networks

Routing protocols for ad hoc networks can be categorized into three classes: *Proactive*, *Reactive* and *Hybrid*. In general, protocols differ based on how they handle *Route Discovery* and *Route Maintenance*. Route discovery sets up initial routes or searches for new routes when the old ones break. Route maintenance manages accurate information about existing routes and also supports error recovery when a broken route is detected.

In proactive protocols, routing information is exchanged among neighbors periodically or each time a change occurs in network topology (i.e., route discovery and route maintenance are continuously performed). While protocols in this category have the advantage that routes are immediately available when requested, they incur high control overhead. Although hybrid protocols aim to reduce the control overhead by balancing proactivity and reactivity, all current hybrid protocols rely on proactively acquiring at least one-hop neighborhood information for all nodes. In comparison, on-demand routing protocols reduce routing overhead by tying route discovery to network communication [2], [3]. On-demand protocols initiate a route discovery only when a new route is needed for initial route set-up or due to a broken route. Such route discovery is achieved by flooding the network, which causes high routing overhead and interference with ongoing traffic. If a broken route can be repaired, route discovery from the source, which causes the network-wide flood, is no longer necessary. However, it is important to ensure that any route recovery technique costs less in terms of control overhead and delay than a new route set-up via flooding.

Several protocols implement solutions to the flooding problem in on-demand routing by providing more efficient route recovery mechanisms. These protocols can be categorized into three main classes:

- *Limited broadcast:* Route discovery is initiated by relay nodes. The broadcast range is limited and does not flood the whole network [6], [7], [14].
- *Multipath routing:* Multiple routes are discovered and cached in a single route discovery [11], [10], [12], [13].
- *Local error recovery:* Route errors are handled at a relay node instead of relying on end-to-end error recovery at the sender [9], [8].

The rest of the section discusses route recovery protocols in these three classes in more detail.

A. Limited Broadcast Approaches

AODV [6] and ABR [7] provide route recovery by allowing relay nodes to initiate a search to replace a failed route. In AODV, if a node is no further than *maximum repair* hops away from the destination, it attempts to repair the route by broadcasting a route request with a limited time-to-live. ABR (Associativity-Based Routing) uses a similar technique but repairs broken routes with new routes that tend to be more long-lived by basing route decisions on a measure of next hop mobility. However, both methods are too bandwidth consuming, since even with a limited scope, flooding can deliver the request messages to a large number of nodes, leading to high routing overhead. Additionally, routes are repaired based on route replies from caches, and therefore recovery may not be successful if the caches contain stale routes. Other mechanisms that try to localize flooding to a limited region of the network either require location information (e.g., GPS) [14] or use a heuristic-based approach that requires fine tuning of parameters based on current network characteristics to determine the query region [15].

B. Multipath Routing Approaches

Multipath routing [10], [11], [12] discovers and caches multiple routes with a single route discovery. When a broken route is detected, it is expected that other routes are

available from the cache and a new route discovery due to a broken route is only needed when all cached routes to a destination break. Although multipath routing reduces the number of route discovery attempts, it may not be effective in the presence of mobility, incurring additional packet drops and delay. For instance, if a significant amount of time has passed between route discovery and route recovery, it is likely that the cached routes are invalid due to topology changes. Without any mechanism to keep the caches up-to-date, a route discovery attempt may be inevitable.

C. Local Error Recovery Mechanisms

Local error recovery provides more robust route recovery during route failures in mobile environments by allowing a relay node to repair a broken route. While local recovery can extend the lifetime of a route in traditional routing protocols, it can also extend the lifetimes of individual routes when multiple routes are used for load balancing [16], [11], [17].

In AODV-BR [13], nodes snoop route reply messages to create alternate next hops to a destination. When a node detects a broken route, it broadcasts the packet to its neighbors, which forward it to the destination (if they have an entry in their cache). However, the alternate routes may be stale since they are only populated during route discovery. A similar approach is used in WAR (Witness-Aided Routing) [9] that designates nodes in the neighborhood of an ongoing communication as witness nodes, which buffer overheard packets and deliver the packets themselves to the next hop if a failure occurs. This requires witness nodes to maintain state and dedicate storage for communication in which they are not involved. Additionally, both AODV-BR and WAR may cause unnecessary overhead by delivering the same packet to the destination several times.

DSR [18] potentially caches multiple routes to a destination and provides a route salvaging option that enables relay nodes to recover from route failures locally by searching for alternate routes in their caches. However, nodes immediately send a route error back to the source. Therefore, salvaging in DSR does not achieve any reduction in the frequency of route discoveries. A recently proposed protocol, CHAMP (Caching and Multipath) [19], uses a salvaging algorithm where all nodes temporarily cache packets before forwarding. When a node receives a route error and if the failed packet still exists in its *packet cache*, the node sends the packet with an alternate route from its route cache. Therefore, local recovery is achieved by incurring additional storage overhead in relay nodes. Furthermore, trying to recover from the failure at all upstream nodes using stale route cache information may incur additional delay and packet overhead.

NSR [8] incorporates proactivity into DSR by maintaining two-hop neighborhood state for all nodes via HELLO, route request and reply messages and uses this link-state information to enable relay nodes to repair broken routes. Specifically, HELLO messages contain each node's neighbors, which provides a way of obtaining two-hop neighborhood state that cannot be acquired by snooping. However, HELLO messages incur additional overhead even when the network is stable and hence, are very expensive.

Although each of these local error recovery mechanisms has limitations, we believe localization of recovery is necessary for the scalability of ad hoc routing protocols. Next, we present design guidelines for local recovery protocols to achieve effective local recovery.

D. Local Recovery Protocol Design Guidelines

The goal of any local recovery mechanism should be to repair broken routes in a way that reduces control overhead and chooses valid routes. Therefore, a routing algorithm with local recovery should possess the following characteristics to enable efficient route repair:

- Repair with cached routes when available
- Repair with local error recovery when cached routes are not available
- Repair all active routes affected by broken links
- Utilize bandwidth efficiently

Utilization of both route caches and local error recovery mechanisms is essential for providing robust recovery in ad hoc networks. The benefit of using route caches is two-fold. First, when a link failure occurs, an alternate route may be immediately available. Second, using route caches can provide reduction in the control overhead that is required to repair a route. However, in high mobility environments route caches may contain stale routes. Therefore, route recovery should not entirely rely on route caches and local error recovery mechanisms should take over when route caches do not provide useful information.

Although the main concern of local recovery is to repair a broken route as fast as possible, it is important to alleviate the effects of a broken link on future transmissions by repairing all routes that actively use the broken link. Finally, a local recovery mechanism should use bandwidth efficiently by incurring minimum overhead.

To achieve efficient and effective route recovery, a local recovery protocol should follow these guidelines. However, none of the protocols discussed in Section II-C fully recognize these characteristics. For example, NSR, which

is the closest approach to our work, uses local error recovery for route repairs. However, recovery is achieved at the expense of efficient bandwidth utilization (due to HELLO messages). To this end, we propose bypass routing, which is a novel approach that incorporates these guidelines for effective route recovery.

III. Bypass Routing

Bypass routing performs on-demand route recovery utilizing both route caches and local error recovery. Essentially, to recover from a route failure, a node first salvages a route by searching its route cache for an alternate route to the destination (if the node caches multiple routes). If a route exists, the node patches the broken route with the alternate route. If the node is not able to repair the route from its route cache, it initiates bypass recovery by querying its neighbors to see if they have a link to any nodes on the downstream route to the destination (e.g., the next hop, or all downstream nodes in case of source routes). As replies arrive, the node repairs the routes affected by the link failure with the received connectivity information. When those packets reach the destination, the new route information is added to an enhanced route error packet and sent back to the source to inform it about the broken link and successful route change.

A. An Illustration of Bypass Routing

Although bypass routing is not limited to any routing protocol, the specifics of the mechanism depend on the characteristics of the underlying on-demand routing protocol. An illustration of bypass routing integrated into a source routing protocol is shown in Figure 1. Initially, the flow from Node S to Node D uses the route "S-L-M-N-P-R-D". When the link between Node M and Node N breaks, Node M detects the failure and attempts to patch the route by using an alternate route from its cache to destination D. When Node M finds a route without loops, Node M salvages the packet with the "S-L-M-T-U-V-D" route. Figure 1 also illustrates an example of bypass recovery. Again, Node S initially uses the route "S-L-M-N-P-R-D". When the link between Node P and Node R breaks and Node P does not have an alternate route in its cache to the destination, Node P triggers a local query to its neighbors. The neighbors reply if they have active links to any of the downstream nodes on the broken route. In Figure 1, Node X reports its connectivity with Node R to Node P. Node P patches the route accordingly and the packet is first forwarded to Node X and then to Node R to reach the destination.

These examples show that local repair of broken routes may result in an increase in route lengths. However, sources that are informed of repair information are not forced to use a longer route if they know a shorter route. Specifically, bypass recovery aims to reduce the frequency of route discoveries, while allowing the node to use shorter routes when possible.

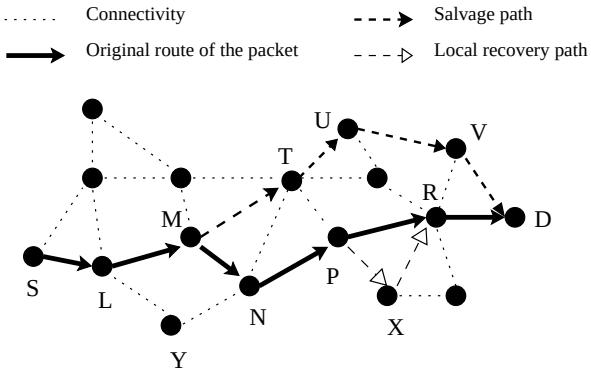

Fig. 1. Error recovery example

B. Operation of Bypass Routing

Bypass routing uses three mechanisms that work together to allow efficient recovery from route failures. A novel *MAC cache* structure is used for determining the state of links to neighboring nodes. A *Route cache* caches recently discovered routes to avoid expensive route discovery. Finally, *Error recovery* includes *route salvaging* and *bypass recovery*.

The remainder of this section explains the MAC cache and error recovery mechanisms. We leave discussing route caches to Section IV, since operation of route caches depends on the underlying routing protocol.

1) MAC Cache: In bypass routing, MAC caches provide connectivity information. To maintain the most recent neighborhood state, a node updates its MAC cache if any communication is heard from any neighbor. On any activity, a neighbor's link status is set to *active*. Cache invalidation is a two stage process. If a *refresh interval* passes without any sign from a neighbor, the neighbor's *link status* is set to *no communication*. Once in the no communication state, if there is still no sign of the neighbor during the *delete interval*, the neighbor is deleted from the MAC cache. The rationale behind this two-stage process is that it provides a second chance for nodes that have not been in active communication recently.

Bypass routing uses MAC caches to enhance route selection. A node searching for a route in its route cache checks if the next hop exists in its MAC cache. A route is considered stale if the next hop is not in the MAC cache. Therefore, MAC caches serve to determine the validity of the routes in route caches. A node uses a route even

if the next hop is in the *no communication* state, thereby utilizing its route cache even when there is little communication in the network. However, a neighbor is deleted from the MAC cache if there is no activity from it within a *delete interval*. Although this does not necessarily mean the link is broken and may result in not using a valid route, the route is reactivated as soon as the node hears from this neighbor. Essentially, if the node does not have any routes to the destination, a one-hop query is performed, which potentially rediscovers all neighbors without flooding the network.

MAC caches are also essential during local recovery, where a node queries its neighbors to see if they have connectivity to any of the downstream nodes on the broken route to the destination. For local recovery, the status of the links should be taken into account and only nodes that have active links to any of the nodes in the query message should reply, since query replies must carry the most recent connectivity information to facilitate effective local recovery.

While bypass routing is not limited to any specific MAC protocol, some knowledge of the underlying MAC protocol is useful to successfully populate the MAC cache. For example, IEEE 802.11 [20] uses RTS (Request-to-send) and CTS (Clear-to-send) to provide a form of channel reservation. After the RTS/CTS exchange, data packet transmission is followed by an ACK. A node listens for RTS/CTS/DATA/ACK packets and updates its MAC cache accordingly. It must be noted that this type of snooping of MAC headers does not incur any additional listening overhead at a node, since a node must try to receive a packet to see if there is a packet being transmitted to it. However, IEEE 802.11 has one behavior that requires additional state to determine the originator of a packet. While RTS and DATA carry both sender and receiver information, CTS and ACK only contain receiver information. When a node overhears a CTS (ACK), the node checks if it is the destination and if it has recently sent the corresponding RTS (DATA). However, without any additional mechanisms, a node that is not the sender of the RTS (DATA) cannot determine the originator of the CTS (ACK). Due to this problem, a node can only cache senders but not receivers on a route. To utilize CTS and ACK packets in bypass routing, all nodes follow the same method as the actual senders of RTS and DATA. Basically, a node that has overheard an RTS (DATA) records the receiver and the sender of the packet. When the node overhears the CTS (ACK), it checks if the recorded sender matches the receiver of the CTS (ACK). If there is a match, the receiver of the RTS (DATA) is a neighbor and should be in the MAC cache. Using this method, MAC cache represents the current neighborhood of a node more accurately (see Figure 2).

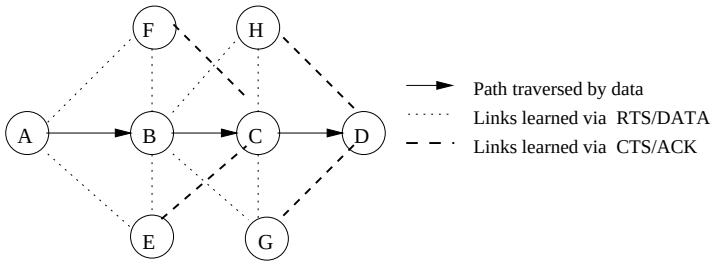

Fig. 2. Link-state information learned from RTS/CTS/DATA/ACK

2) Error Recovery: In bypass routing, error recovery proceeds in three stages : 1) salvaging using route caches, 2) bypass recovery, and 3) error reporting (see Figure 3). In the presence of a route failure, a node initially searches its cache for alternate routes to the destination to salvage a packet and records this recovery attempt in a *fail-record table*. If the salvaged packet arrives at its destination, the destination sends back an enhanced route error message to the source to indicate the salvaged route as an alternate to the broken route. When forwarding this information, the repairing node snoops on the packet to update the fail-record with the repair information. If new packets that use the broken link in their source routes arrive at the repairing node, the node continues salvaging packets using the repair information. However, if no acknowlegment is received from the destination the node should prevent packets from using the recently broken link and send back error messages to source of these packets.

In bypass routing a packet is salvaged only once. This is important to reduce the reliance on route caches for route recovery. Failure of the salvaged route serves as a warning for the node detecting the failure to perform bypass recovery instead of salvaging. Thereby, a node performs bypass recovery if 1) the node does not have an alternate path to perform salvaging, or 2) the packet has already been salvaged. Specifically, the node buffers the failing packet and all packets in the interface queue that need to use the broken link in a *fail-packet buffer*. The bypass recovery attempt is recorded in the *fail-record table* and a list of nodes existing on such soon-to-fail routes is broadcast to one-hop neighbors to see if they are neighbors with any of those nodes. Therefore, a query reply in by-pass recovery can fix multiple active routes.

The nodes that receive the query message search their MAC caches for a neighbor listed in the query. The node includes all such neighbors in its reply. To avoid query reply storms, the nodes use a random backoff algorithm and send a query reply only if they have not overheard an identical query reply. When the querying node receives

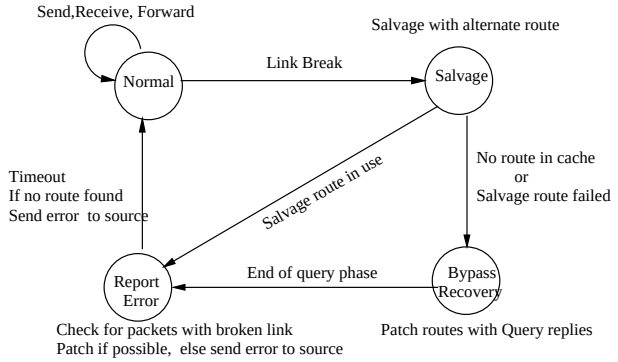

Fig. 3. Protocol state diagram

a reply, it checks its *fail-packet buffer* to repair as many packets with broken routes as possible with new link-state information. For flows that have switched to a new route, one packet is marked to warn the destination of the broken link and the route change. The node also updates its route cache with the new connectivity information.

If a repairing node does not receive query reply or enhanced route error messages before a timeout occurs, a route error message is sent back to source. In the worst case, when alternate routes fail, delaying a route error message delays route recovery. However, a successful local recovery prevents any delay from flooding initiated by the source to find a route to the destination. Bypass routing is optimistic assuming it can repair the route, reducing overhead on success and causing limited delay on failure.

IV. Source Routing with Local Recovery

To demonstrate the feasibility and effectiveness of bypass routing, we implemented Source Routing with Local Recovery (SLR). SLR uses DSR as the underlying protocol, but implements route selection and error recovery based on bypass routing. Bypass routing can also be integrated into other routing protocols such as AODV [6], where the node detecting a route failure can query neighbors if they have connectivity to the other end of the broken link or the destination. Our future plan is to evaluate the effect of bypass routing on other protocols. Next, a brief overview of relevant aspects of DSR to SLR and a detailed discussion on route caches is presented.

A. Overview of SLR Protocol

SLR uses DSR as the underlying protocol, which is an on-demand routing protocol that only establishes routes to destinations for active flows. When a node wants to find a route to a node, it broadcasts a route request. Each node that receives a route request that it has not seen before appends its own address to the source route and rebroadcasts the packet. When a node that knows a route to the destination or the destination itself receives the route request, a route reply is returned to the source.

In DSR, when a node detects a broken link to a neighbor, the corresponding route entries are deleted from the route cache and the source nodes that are actively using that link are informed of the link failure. Several optimizations for route maintenance have been proposed for DSR [18]. SLR utilizes optimizations such as increased spreading of route errors, automatic route shortening, snooping and route salvaging. With *increased spreading of route errors*, a source node receiving an error packet propagates this error with the next route request. In this way, stale information from route caches can be expunged. *Automatic route shortening* allows a node, which is not the intended next hop but exists later on the route to send the shorter route as a gratuitous route reply to the source. Nodes learn additional routes to each destination and check for route error messages by *snooping*. By *route salvaging*, a node detecting a link failure may send the packet by replacing the source route with an alternate route from its cache. While DSR's route salvaging is similar to route salvaging in bypass routing, DSR uses this optimization to provide the necessary time for a source to complete a new route discovery, while reducing the number of packets dropped due to link failures. Essentially, a node still sends an error message back to the source to indicate the broken link. Furthermore, a node salvages a packet by simply replacing the source route with an alternate route from its cache, and thereby loses the first (successful) part of the source route. The goal of bypass routing is to provide the source with a stable route as well as to inform the source of the route failure. Therefore, in SLR, a node searches its cache for complete loop-free routes to the destination to salvage a packet, preserving the first part of the route (i.e., from the source to the node detecting the route failure). Additionally, bypass recovery also guarantees loop-free routes by ensuring that any changes made to the route do not include nodes from the traversed route.

B. Route Caches

Route caching is an important part of any on-demand routing protocol. A cache hit with a valid route reduces overhead by eliminating the need for a route discovery and similarly reduces end-to-end delay since the packet can be sent immediately without waiting to discover a new route. On the other hand, route caching introduces the problem of effective strategies for managing the structure and the contents of the cache since node mobility can change network topology and possibly invalidate cached

routes. The use of an invalid cached route incurs additional bandwidth and delay. Therefore, careful design choices must be made about the cache *structure* and cache *timeout* mechanisms.

Two alternatives, *path cache* and *link cache*, have been proposed for implementing route caches in DSR [21]. Path caches store each complete per-destination route, while link caches create a unified graph from all links in all routes. Although path caches are simple to implement and easily guarantee loop-free routes, link caches provide nodes a more detailed view of the current network topology. However, generally, link caches require more complex route search algorithms (e.g., Dijktra's shortest path algorithm) compared to linear search algorithms.

Link caches with an adaptive timeout mechanism (i.e., a link's timeout is chosen according to its stability) have been shown to provide the most accurate information [22], [21]. However, the performance evaluations in [22], [21] do not use timeout mechanisms for path caches. Motivated by [23], we believe that path caches can also benefit from using a timeout so that stale routes are not used in response to route queries or to salvage broken routes. However, there might be cases when stale routes exist in route caches even if a timeout mechanism is employed. Specifically, there is no way to determine the freshness of routes in DSR when nodes promiscuously listen to all packets and cache routes [23]. Even if a route error message erases invalid routes, an *in-flight* data packet that carries the same stale route puts the route back in caches. *Epoch numbers* [24] are proposed to prevent the re-learning of a stale information. However, SLR does not employ such a mechanism, since one of the goals of bypass routing is to reduce the reliance on route caches. Bypass routing utilizes a local recovery strategy when caches contain obsolete information, which enables more effective route correction in caches as well as error recovery. For this reason, the implementation of SLR uses a simple path cache known as Path-Gen-34 [21] instead of more complex link caches.

Path-Gen-34 is composed of a *primary* and a *secondary* cache. Routes learned as a result of route discoveries are inserted into the primary cache. Routes learned by snooping are inserted into the secondary cache. The division of the cache into primary and secondary caches prevents snooped routes from competing for cache space with routes of known value to the node. The original Path-Gen-34 does not employ a timeout mechanism, so routes in the caches are not guaranteed to be fresh. Bypass routing uses MAC cache to determine the validity of routes. Basically, a node selects a route from its route cache if and only if the next hop exists in its MAC caches, effectively incorporating the timeliness of MAC caches into route caches. We will investigate the effect of link caches on local recovery in our future work.

V. Performance Evaluation

The effectiveness of local recovery via bypass routing can be evaluated by its impact on the following properties: routing overhead, number of route requests, packet delivery ratio, goodput, average hop count and average end-to-end delay. The main goal of bypass routing is to localize reaction to route failures and reduce the routing overhead, which is the ratio of the bypass routing and DSR overhead to the amount of data received at the destinations. Additionally, the number of route requests forwarded at each hop shows how frequently route discovery is triggered and the scale of its effect on the whole network. Communication quality in the presence of route failures is represented in terms of packet delivery ratio and average end-to-end delay. While packet delivery ratio quantifies the packet loss rate and is calculated as the ratio of data packets delivered to destinations, average end-to-end delay is the difference between the time a packet was sent by the sender and the time it was received at the destination, including delays due to local recovery. To understand the effects of both routing overhead and of route length, goodput measures the ratio of the number of data packets received to the overhead of bypass routing, DSR and of forwarding data along each hop. Finally, average hop count is the average number of hops a packet travels to reach its destination.

The goal of our simulation study is to measure the success in meeting the design goals of bypass routing. Since SLR is a source routing protocol using bypass recovery, it is natural to compare it to DSR to highlight the effectiveness of bypass recovery. To see the impact of caching on both protocols, we compare the following schemes: *DSR (cache)*, *SLR (cache)*, *DSR (nocache)* and *SLR (nocache)*. DSR (cache) and SLR (cache) fully utilize route caches via propagation of route replies from caches and salvaging, while these options are disabled in DSR (nocache) and SLR (nocache).

We implemented SLR in the ns-2 network simulator using the CMU wireless extension [25]. Our simulation results represent an average of five runs with identical traffic models, but different randomly generated network topologies. We use the *random waypoint* [3] mobility model with a speed uniformly distributed between 0-20 m/sec. We will study the performance of SLR with different mobility models as future work, since the random waypoint model has been shown to have limitations [26]. Table I lists the SLR parameters.

TABLE I

PARAMETERS USED IN SLR SIMULATION

Fail-record: Table size (number of entries)	34
Fail-record: Timeout (s)	1.0
Fail-buffer: Packet timeout (s)	0.02
MAC Cache: Refresh interval (s)	0.05
MAC Cache: Delete interval (s)	3.0

TABLE II

AVERAGE HOP COUNT WITH DIFFERENT TRAFFIC LOAD

Network	SLR (cache)	DSR (cache)	SLR (nocache)	DSR (nocache)
1500x500	3.427	3.144	3.324	3.160
2000x1500	5.484	4.861	5.590	5.383

A. *Impact of traffic load*

To evaluate the impact of traffic load, simulations were run with varying transmission rates. The network simulated is a 1500m $\times$ 500m network with 60 nodes and 20 long-lived CBR connections that start randomly between 20s and 25s. Transmission rates vary between 0.2Kb/s-2.2Kb/s. All data packets are 128 bytes. Each simulation runs for 600s. A pause time of 60s was used to achieve moderate mobility.

As the network load varies, SLR is successful in localizing route recovery overhead, while DSR imposes overhead on the whole network due to flooding of route requests (see Figure 4). Essentially, SLR (cache) incurs 3-5 times less routing overhead compared to DSR (nocache) and 2-4 times less routing overhead compared to DSR (cache). With SLR (nocache), there is an increase in routing overhead when the network traffic load is low. As network load increases, local recovery becomes possible with more frequently updated MAC caches and the overhead of SLR (nocache) reduces to DSR (cache) overhead. Additionally, while the number of route requests ranges between 10000-20000 packets for DSR(nocache), this is reduced to 2000-6000 in SLR (cache) (see Figure 5). As network load increases SLR (nocache) performs comparable to DSR (cache), since DSR (cache) reduces the number of route requests during route discovery by allowing relay nodes to reply to route requests. However, DSR (cache) has no effect on the number of route requests generated due to route recovery. In contrast, SLR (nocache) reduces the number of route requests from recovery via bypass routing but does not affect the number of route requests generated during discovery.

In addition to reducing the overhead, SLR achieves high packet delivery ratio compared to DSR (see Figure 6). SLR (cache) shows the best performance by increasing the delivery ratio between 3-19%. With SLR (nocache), the increase in delivery ratio is lower in low traffic loads. As the network load, and so the amount of communication, increases, the nodes provide more robust recovery with more frequently updated MAC caches. As shown in Figure 6, the packet delivery ratio of DSR (cache) is lower than DSR (nocache) and SLR. This indicates that SLR uses route caches more effectively than DSR by tying route validity to up-to-date neighborhood information.

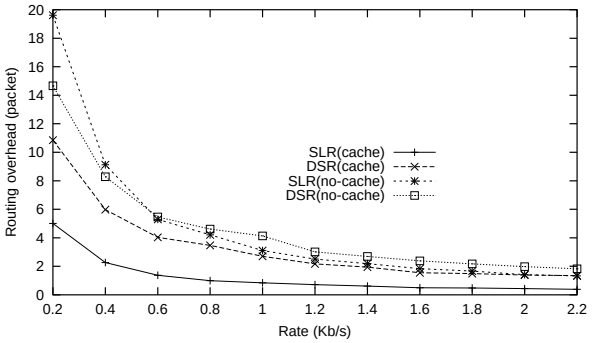

Fig. 4. Routing overhead vs. traffic load in terms of packets, 20 CBR connections, 60 nodes, 1500mx500m region, speed 0-20m/s

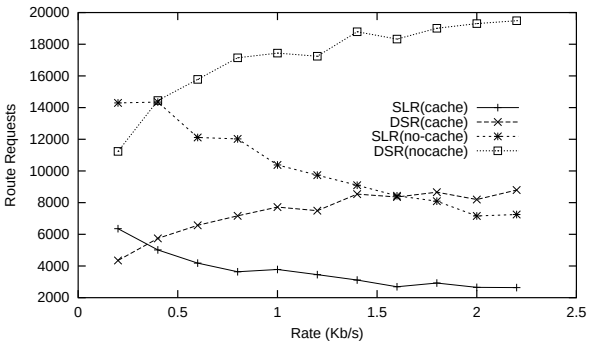

Fig. 5. Route Requests vs. traffic load, 20 CBR connections, 60 nodes, 1500mx500m region, speed 0-20m/s

However, in terms of goodput, SLR performs comparable to DSR (see Figure 7). The main reason for comparable goodput is that SLR forwards more data over longer routes (see Table II). However, although SLR uses longer routes, it does not incur higher delays and delivers most of the packets in less than 0.01 seconds. Essentially, SLR reduces the delays due to route discovery while incurring little increased delay from the optimistic route recovery (see Figure 8).

These simulation results show that SLR performs significantly better than DSR in terms of throughput, balancing local recovery overhead with efficient communication. It is interesting to note that to achieve better delivery ratios with DSR, DSR (nocache) should be used. While DSR (cache) is successful in reducing the routing overhead, it

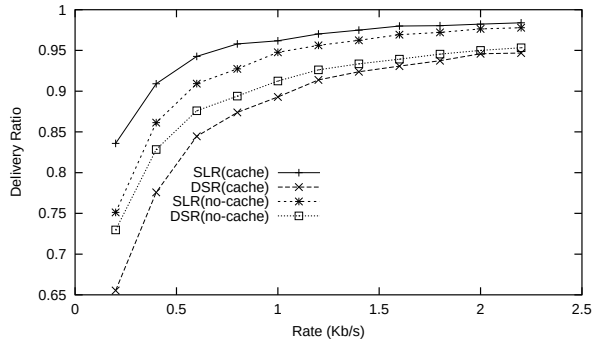

Fig. 6. Delivery Ratio vs. traffi c load, 20 CBR connections, 60 nodes, 1500mx500m region, speed 0-20m/s

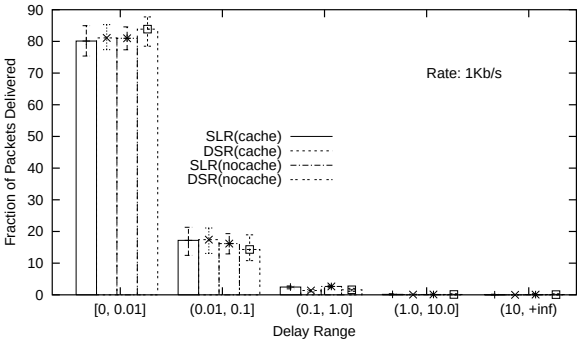

Fig. 8. Delay vs. traffi c load, 20 CBR connections, 60 nodes, 1500mx500m region, speed 0-20m/s

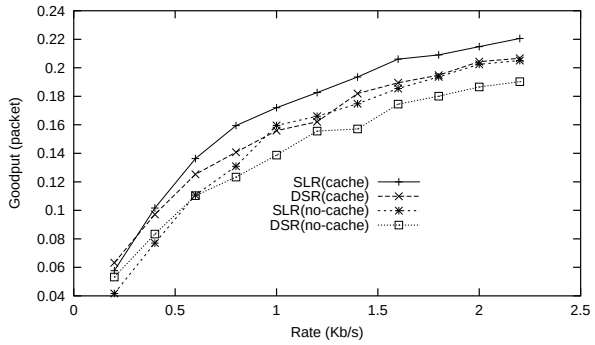

Fig. 7. Goodput vs. traffi c load in terms of packets, 20 CBR connections, 60 nodes, 1500mx500m region, speed 0-20m/s

only provides modest delivery ratios. In comparison, SLR is able to achieve both high delivery ratios and low routing overhead without being forced to choose between caching and no caching.

B. Impact of network size

To determine the scalability of SLR as the network diameter increases, SLR was evaluated in a larger network with 150 nodes. Nodes move in a 2000m $\times$ 1500m region. A total of 40 long-lived connections start randomly between 20s and 25s. Traffic sources are CBR with transmission rates between 0.4Kb/s-4.4Kb/s. All data packets are 256 bytes. Each simulation runs for 900s. A pause time of 60s is used to achieve moderate mobility.

In larger networks, DSR (cache) routing overhead increases considerably due to the fact that DSR (cache) relies on route caches to discover and repair routes. However, in a large network with long routes, the probability that a route breaks increases, increasing the probability that a cached route is stale. DSR (cache) performance degrades since it replies to route requests with stale data. This is further amplified by the fact that other nodes overhear and cache stale routes from route replies. This kind of behavior has also been reported in [27]. Simulation results show that DSR (cache) creates 5 times more overhead on average compared to DSR (nocache) and SLR (see Figure 9). Essentially, more route requests are generated as routes provided by relay nodes fail. Figure 10 confirms our analysis that DSR (cache) incurs the highest route request overhead, which is 427,000 pakets on average. This is significantly larger compared to the average of 199,000 packets in SLR (cache).

Unfavorable effects of caching on DSR are also observed in terms of delivery ratio (see Figure 11). However, in comparison to DSR (cache), the delivery ratio of SLR (cache) remains high until the per node traffic load increases to more than 2.0Kb/s. For higher traffic loads, SLR (cache) delivers 2-13% percent less data compared to DSR and SLR without caching. However, SLR (cache) performs with 22-35% higher delivery ratio compared to DSR (cache) at all traffic rates. Furthermore, SLR (nocache) provides the best delivery ratio.

SLR is able to scale to larger networks via bypass recovery. In particular, SLR (nocache) shows the best performance by achieving high delivery ratio with the least amount of overhead compared to DSR and SLR (cache). Although SLR (cache) is able to reduce the unfavorable effects of caching for low traffic loads, as the traffic load increases, there is a degradation in delivery ratio performance. The decrease in delivery ratio comes from aggressive use of the underlying DSR caching mechanism during route discovery.

C. Impact of Mobility

The final set of simulations evaluate the impact of mobility on SLR. Simulations were run in a 1500m $\times$ 500m network with 60 nodes and 20 CBR connections with transmission rates of 4Kb/s. The data packet size is 256 bytes. The mobility rate is changed by setting pause times to 0, 30, 60, 120, 300 and 600 simulation seconds.

As the mobility rate changes, SLR performs similar to first set of experiments with different traffic loads, which

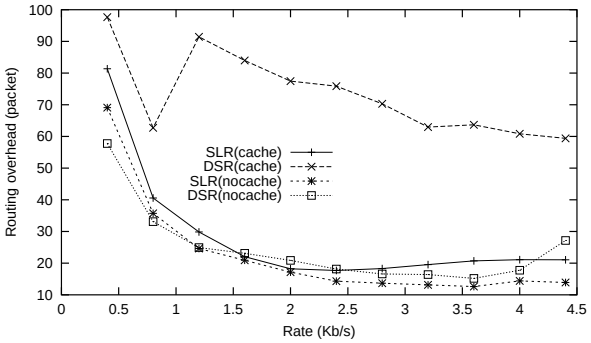

Fig. 9. Routing Overhead in terms of packets vs. traffi c load, 40 CBR connections, 150 nodes, 2000mx1500m region, speed 0-20m/s

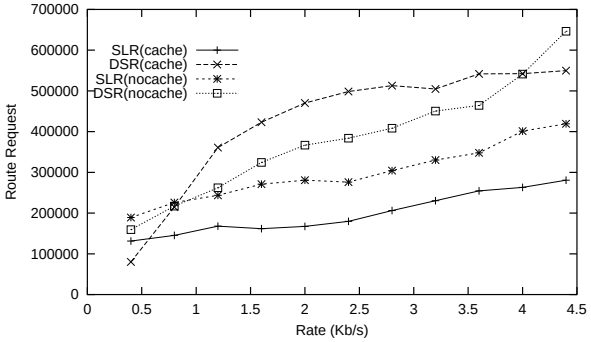

Fig. 10. Route Requests vs. traffi c load, 40 CBR connections, 150 nodes, 2000mx1500m region, speed 0-20m/s

indicates that SLR is successful in localizing the reaction to topological changes. Specifically, SLR outperforms DSR in terms of routing overhead (see Figure 12) . While SLR (cache) incurs half as much overhead compared to DSR (nocache), SLR (nocache) overhead performance is comparable to DSR (cache). This can be clearly seen in Figure 13, where DSR (cache) and SLR (nocache) create a similar amount of route requests. On the other hand, the route request overhead of SLR (cache) remains flat at 3000 packets. In contrast, the number of route requests in DSR (nocache) is significantly larger.

In the presence of mobility, the packet delivery ratio of SLR is higher than DSR (see Figure 14). While the performance improvement is higher under high mobility, DSR catches up with SLR when mobility is low and the number of broken routes decreases. However, SLR's improved throughput is achieved with only comparable goodput due to use of longer hop routes (see Figures 15 and 16).

The evaluation of delivery ratio, routing overhead and hop length confirm our expectations of SLR's performance in the presence of mobility. Essentially, these simulation results are similar to the results presented in V-A, showing the same performance trends.

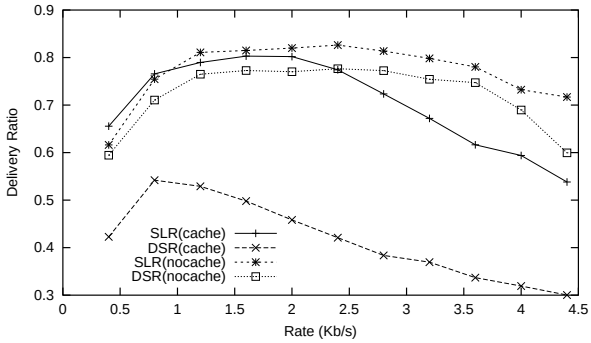

Fig. 11. Delivery Ratio vs. traffi c load, 40 CBR connections, 150 nodes, 2000mx1500m region, speed 0-20m/s

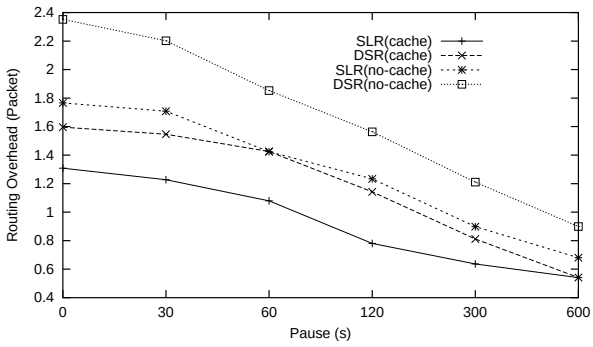

Fig. 12. Routing overhead (packet) vs. mobility, 20 CBR connections, 60 nodes, 1500mx500m region, speed 0-20m/s

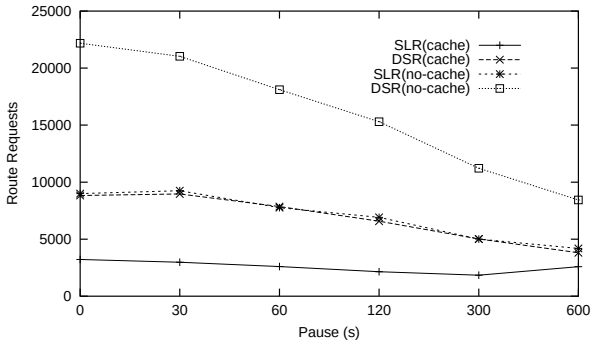

Fig. 13. Route requests vs. mobility, 20 CBR connections, 60 nodes, 1500mx500m region, speed 0-20m/s

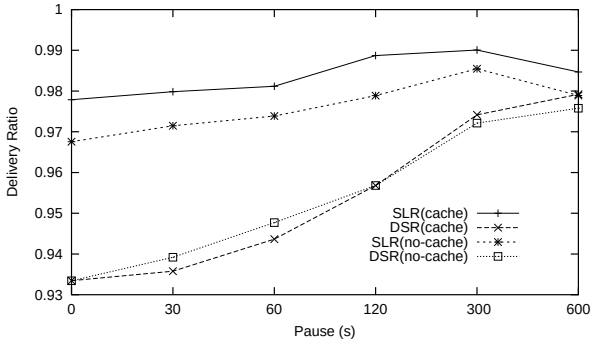

Fig. 14. Delivery Ratio vs. mobility, 20 CBR connections, 60 nodes, 1500mx500m region, speed 0-20m/s

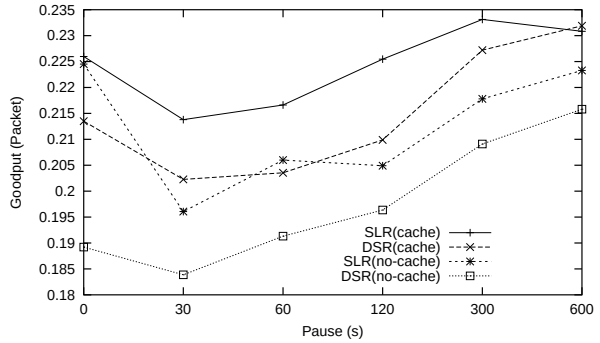

Fig. 15. Goodput (packet) vs. mobility, 20 CBR connections, 60 nodes, 1500mx500m region, speed 0-20m/s

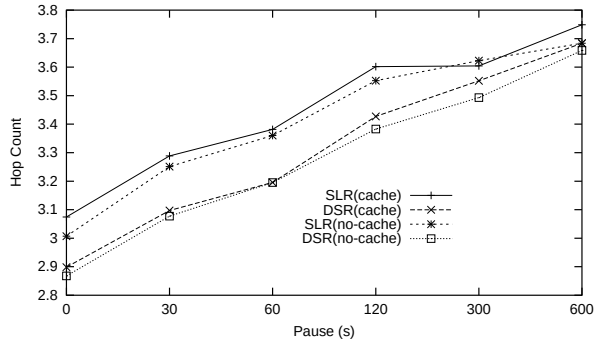

Fig. 16. Average number of hops vs. mobility, 20 CBR connections, 60 nodes, 1500mx500m region, speed 0-20m/s

VI. Discussion

It was not possible to evaluate the performance of SLR in comparison with other local recovery techniques discussed in Section II-C, since they are not publicly available. However, we discuss the performance of SLR compared to AODV-BR [13] and NSR [8] based on the observed behavior as published.

In AODV-BR, a node that is not on the direct path snoops data packets to maintain alternate routes without using additional control messages. Therefore, alternate routes are created only when building the actual routes. On the other hand, SLR only uses overheard packets to determine the current neighborhood of a node, and therefore incurs less processing overhead. The main difference between AODV-BR and SLR is that when a route breaks in AODV-BR, the node detecting the failure broadcasts the packet to its neighbors. Since any node that receives the broadcast tries to forward it to the destination, the packet can potentially be delivered to its destination through multiple alternate paths. AODV-BR incurs more overhead as the number of alternate paths increases and delivers more copies to the destination. On the other hand, SLR uses a one hop broadcast to query its neighbors for connectivity information and sends the packet through the best alternate path. If the packet reaches its destination using the alternate path, the destination warns the source of the route break and informs it about the new path. This is in contrast to AODV-BR, which still sends a route error back to the source and leaves the responsibility of reconstructing a new route to the source. Simulation results show that as the load in the network increases, AODV-BR cannot provide any improvements over AODV due to increased contention and collisions [13]. In certain cases, AODV-BR performs worse than AODV, since data packets traveling through alternate paths collide with packets using primary paths. We have not seen such a trend in the performance of SLR, which provides a high route delivery ratio even when the network load increases, since only one copy of the packet is ever in-flight at a time. Essentially, SLR captures the current state of the network better through MAC caches and adapts route caches accordingly.

In NSR, a node maintains a partial topology of the network consisting of the links to its 1-hop and 2-hop neighbors and the links on the known paths to destinations. Link-state information is maintained proactively via periodically broadcast HELLO messages (or piggybacked on route requests and replies), and used to repair routes when link failures occur. Compared to NSR, SLR does not incur any overhead to maintain 1-hop neighborhood information and depends only on overhearing packets. To acquire 2-hop neighborhood information, a query message is sent on-demand in the presence of a link failure. In contrast, the overhead from HELLO messages in NSR exists even when the network is stationary and no routes fail. To reduce the control overhead from HELLO messages, NSR sets the time interval between two HELLO messages to 59s [8]. Reducing this time interval guarantees more recent information but increases overhead. SLR maintains neighborhood information passively using more fine-grained timers (e.g., 0.05s) and therefore, bypasses broken links using more recent connectivity information. Furthermore, NSR reports new routes to the source only in certain cases and does not validate the success of the alternate route. SLR reports all successfully repaired routes. Simulation results in [8] show that NSR achieves only comparable delivery ratio to DSR when there is n sources and n destinations in the network. There is a 40%-50% increase in delivery ratio when there are n sources and 1 destination due to NSR's success in distributing the load better than DSR. SLR provides better delivery ratio even when the communication patterns include n sources and n destinations. Essentially, SLR achieves the same goals as NSR but with a novel on-demand protocol, which provides a complete solution with salvaging, bypass recovery and error reporting.

VII. Conclusions

On-demand routing protocols are attractive for ad hoc networks. However, their effectiveness is limited by the use of flooding to discover new routes when the current route breaks. In this paper, we propose bypass routing, which reduces the need to perform route discovery for broken routes via bypass recovery and a novel cache invalidation mechanism. Our primary concern is to provide robustness to route failures and maintain high delivery ratio and low overhead. We implemented a prototype of bypass routing based on source routing. Simulations show that SLR (Source Routing with Local Recovery) achieves significantly higher throughput while maintaining acceptable overhead. The results verify that local recovery is the right approach for route recovery in ad hoc networks.

Our future plan is to investigate the benefits of bypass routing with other on-demand protocols. We also plan to evaluate the effectiveness our approach by comparing its performance with hybrid and local recovery protocols. Additionally, we will look into other issues related to local recovery such as utilizing link caches in SLR's implementation and further evaluate our scheme with more realistic mobility models. Our future research goal is to study bypass routing in power-saving ad hoc networks. We believe the advantages gained through bypass routing due to its localized behavior will enable better power management.

References

[1] C. Perkins and P. Bhagvat, "Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers," in *ACM SIGCOMM*.

[2] E. Royer and C.-K. Toh, "A review of current routing protocols for ad hoc mobile wireless networks," *IEEE Personal Communications*, vol. 6, no. 2, pp. 46–55, Apr. 1999.

[3] 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 *4th Annual International Conference on Mobile Computing and Networking (MobiCom)*, 1998.

[4] Z. J. Haas and M. R. Pearlman, "Determining the optimal path configuration for the zone routing protocol," *Journal of Selected Areas in Communication*, vol. 17, no. 8, pp. 1395–1414, 1999.

[5] V. Ramasubramanian, Z. J. Haas, and E. G. Sire, "SHARP: A hybrid adaptive routing protocol for mobile ad hoc networks," in *4th ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, 2003.

[6] C. E .Perkins, E. M. Royer, and S. R. Das, "Ad hoc on-demand distance vector (AODV) routing," draft-ietf-manet-aodv-11.txt, June 2002.

[7] C.-K. Toh, "Associativity-based routing for ad hoc mobile networks," *Wireless Personal Communications Journal, Special Issue on Mobile Networking & Computing Systems*, vol. 4, no. 2, pp. 103–109, 1997.

[8] M. Spohn and J. J. Garcia-Luna-Aceves, "Neighborhood aware source routing," in *2nd ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, 2001.

[9] I. D. Aron and S. K. S. Gupta, "Analytical comparison of local and end-to-end recovery for reactive protocols for mobile ad hoc networks," in *3rd ACM International Workshop on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, 2000.

[10] M. K. Marina and S. R. Das, "On demand multipath distance vector routing in ad hoc networks," in *IEEE International Conference of Network Protocols (ICNP)*, 2001.

[11] S.-J. Lee and M. Gerla, "Split multipath routing with maximally disjoint paths in ad hoc networks," in *IEEE International Conference on Communications (ICC)*, 2001.

[12] J. Raju and J. J. Garcia-Luna-Aceves, "A new approach to on-demand loop-free multipath routing," in *IEEE International Conference on Computer Communications and Networks (ICCCN)*, 1999.

[13] S.-J. Lee and M. Gerla, "Backup routing in ad hoc networks," in *IEEE Wireless Communications of Networking Conference (WCNC)*, 2000.

[14] Y.-B. Ko and N. H. Vaidya, "Location-aided routing (LAR) in mobile ad hoc networks," in *4th Annual International Conference on Mobile Computing and Networking (MobiCom)*, 1998.

[15] R. Castaneda and S. R. Das, "Query localization techniques for on-demand routing protocols in ad hoc networks," in *5th Annual International Conference on Mobile Computing and Networking (MobiCom)*, 1999.

[16] A. Tsirigos and Z. J. Haas, "Multipath routing in the presence of frequent topological changes," *IEEE Communications Magazine*, 2001.

[17] P. P. Pham and S. Perreau, "Performance analysis of reactive shortest path and multi-path routing mechanism with load balance," in *IEEE INFOCOM*, 2003.

[18] D. B. Johnson, D. A. Maltz, Y.-C. Hu, and J. G. Jetcheva, "The dynamic source routing protocol for mobile ad hoc networks (DSR)," draft-ietf-manet-dsr-07.txt, Aug. 2002.

[19] A. Valera, W. K. G. Seah, and S. V. Rao, "Cooperative caching and shortest multipath routing in mobile ad hoc networks," in *IEEE INFOCOM*, 2003.

[20] IEEE 802 LAN/MAN Standards Committee, "Wireless LAN medium access control (MAC) and physical layer (PHY) specifications," IEEE Standard 802.11, 1999.

[21] Y.-C. Hu and D. Johnson, "Caching strategies in on-demand routing protocols for wireless ad hoc networks," in *6th Annual International Conference on Mobile Computing and Networking (MobiCom)*, 2000.

[22] W. Lou and Y. Fan, "Predictive caching starategy for on-demand routing protocols in wireless ad hoc network," in *2nd ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, 2001.

[23] M. K. Marina and S. R. Das, "Performance of route caching strategies in dynamic source routing," in *In Proceedings of the 2nd Wireless Networking and Mobile Computing (WNMC)*, 2001.

[24] Y.-C. Hu and D. B. Johnson, "Ensuring cache freshness in on-demand ad hoc network routing protocol," in *Principles of Mobile Computing (POMC)*, 2002.

[25] UCB/LBNL/VNT, "Network simulator-ns2 and cmu monarch extension," http://www.mash.cs.berkeley.edu/ns, http://www.monarch.cs.cmu.edu.

[26] J. Yoon, M. Liu, and B. Noble, "Random waypoint considered harmful," in *IEEE INFOCOM*, 2003.

[27] G. Holland and N. Vaidya, "Analysis of tcp performance over mobile ad hoc networks," in *5th Annual International Conference on Mobile Computing and Networking (MobiCom)*, 1999.