Improving Protocol Robustness in Ad Hoc Networks through Cooperative Packet Caching and Shortest Multipath Routing

Alvin Valera, Winston K.G. Seah, and S. V. Rao

Abstract

A mobile ad hoc network is an autonomous system of infrastructure-less, multihop, wireless mobile nodes. Reactive routing protocols perform well in this environment due to their ability to cope quickly against topological changes. This paper proposes a new routing protocol named CHAMP (**C**ac**H**ing **A**nd **M**ultiple **P**ath) routing protocol. CHAMP uses *cooperative packet caching* and *shortest multipath routing* to reduce packet loss due to frequent route failures. We show through extensive simulation results that these two techniques yield significant improvement in terms of packet delivery, end-to-end delay and routing overhead. We also show that existing protocol optimizations employed to reduce packet loss due to frequent route failures, namely local repair in AODV and packet salvaging in DSR, are not effective at high mobility rates and high network traffic.

Index Terms

Routing protocols; Algorithm/protocol design and analysis.

I. Introduction

Mobile ad hoc networking is rapidly gaining popularity due to the proliferation of miniature yet powerful mobile computing devices. Mobile ad hoc networks do not require any form of

A. Valera was with the Department of Computer Science, School of Computing, National University of Singapore, Singapore 117543.

W. Seah and S. V. Rao are with the Institute for Infocomm Research, 20 Science Park Rd, #02-34/37 Teletech Park, Singapore 117674.

An earlier version of this paper appeared in IEEE INFOCOM 2003.

fixed infrastructure for hosts to be able to communicate with one another. A source node that needs to communicate with a destination node uses either a direct link or a multihop route to reach the latter. This requires that all nodes must have some basic routing capability to ensure that packets are delivered to their respective destinations. Since nodes may move anytime, then the topology of the network may also change anytime. A major challenge in mobile ad hoc networking is how to maximize data packet delivery in the face of rapidly changing network topology without incurring a large routing overhead.

Over the last few years, many routing protocols for mobile ad hoc networks have been proposed [1], [2], [3], [4], [5], [6], [7], [8], [9]. A number of performance comparison studies [10], [11], [12] have revealed that on-demand routing protocols perform better in terms of packet delivery and routing overhead than proactive routing schemes especially in the presence of node mobility. Proactive and hybrid schemes do not perform well in dynamic topologies because of two major factors: slow detection of broken links and periodic exchange of route updates even when routes are not needed.

Slow detection of broken links causes data packets to be forwarded to stale or invalid paths thereby decreasing the packet delivery ratio. Proactive routing protocols rely on periodic updates to determine if a link to a neighbor is still up. The absence of several consecutive updates from a neighbor implies that the link to this neighbor is down. This causes a large delay before a broken link is declared as "down". Thus, during packet relay, packets are still forwarded to invalid routes. Of course, these packets never reach the destination. In a highly dynamic network topology where link changes are frequent, many such packets are dropped. One solution to make proactive routing protocols quickly detect broken links is to decrease the update interval but this would entail excessive routing overhead.

In on-demand routing, quick detection of broken links is facilitated by hop-by-hop acknowledgment of data packets or the use link layer feedback, if this is available. Aside from enabling rapid detection of broken links, this method also reduces routing overhead because there is no need to send periodic update messages to ascertain if a link is still "up". This approach may however require additional overhead because of data packet acknowledgment. But if the MAC protocol already provides this functionality such as the IEEE 802.11 [13], this is not a major issue.

Another downside of using data packet acknowledgment to determine link status is that link failure is only determined after failing to forward a packet. Hence, this packet and possibly more may become undeliverable. Under normal circumstances, if these undeliverable packets are not originating from the node that encounters the link failure, they are simply discarded. To avoid dropping these undeliverable packets, AODV incorporates an optimization known as "local route repair". DSR also provides a feature for the same purpose known "packet salvaging". However, as will be shown in this paper, these optimizations only worsen the performance of these protocols at high network load and high mobility rates because of their limited effectiveness and undesirable side effects.

We introduce *cooperative packet caching*, a technique that exploits temporal locality in dropped packets, aimed at reducing packet loss due to route breakage. Every node maintains a small buffer for caching data packets that pass through it. When a downstream node encounters a forwarding error, an upstream node with the same data in its buffer and alternative route can retransmit the data. For this strategy to be effective, nodes must store multiple routes to every active destination. Hence, we propose a simple route discovery mechanism that selects the *shortest multipath routes*.

The rest of the paper is organized as follows: In Section II, the property of temporal locality of dropped packets is presented. This property forms the basis of data caching. In Section III, a new routing protocol that implements cooperative packet caching and shortest multipath routing named Caching and Multiple Path (CHAMP) is presented. Performance evaluation through extensive simulations in *ns-2* are conducted in Section IV while Section V presents some related work. Section VI concludes the paper with a summary of the important findings and future work.

II. Motivation for Caching

Caching to improve performance was first introduced in 1965 by Wilkes [14] to bridge the speed gap between processor speed and main memory access time. A cache is a small but fast memory that stores data for use in the near future. It exploits the property of temporal and spatial locality in memory references in order to reduce latency and increase memory bandwidth [15]. Spatial locality is the property whereby access to a memory location indicates that a nearby location will very likely to be accessed in the near future. Temporal locality is the property whereby an access to a memory location indicates that the same location will very likely to be accessed again soon. Without these properties, (that is, if memory accesses are totally random and independent) caching will not improve performance since most cache accesses will lead to "misses".

Castaneda and Das [16] first investigated spatial locality in mobile ad hoc networks in the context of node mobility. They observed that when a mobile node moves, "it cannot move too far too soon". *Query localization techniques* exploit this property to lower routing overhead during route discovery and repair. Instead of network-wide flooding, route requests are flooded only to a limited region that is previously part of a valid route [16]. The reasoning is that the destination (or some nodes with valid routes) can never be too far too soon from these nodes, hence there is a big probability of finding a route.

We now consider this situation: Suppose that some node i is along some route from source h to destination j. Suppose further that whenever a node fails to forward a data packet to its next hop, it drops the packet and sends a route error message to the source using the reverse route. Observe that whenever i receives a route error from a downstream node k, the error indicates the dropping of a data packet that source h has *recently* sent and i has *recently* forwarded. This behavior mimics the property of temporal locality, that is, "a dropped packet is a recently sent packet". Therefore, if i has a buffer for caching forwarded data packets, even if the buffer is small, there is a big probability that the packet is still in that buffer. And if i has some other route to j, it can "salvage" the dropped packet and need not forward the route error upstream.

Although additional storage overhead is required for the data cache and multiple routes, this technique, which we call *cooperative packet caching*, can reduce packet loss due to route breakdowns. In existing reactive routing protocols, only the node encountering the error can salvage or retransmit a data packet. Cooperative packet caching enables more nodes to salvage a dropped packet, or in essence, *packet salvaging is distributed*.

III. CHAMP Routing Protocol

CHAMP operates in a reactive or event-driven fashion, executing only when a *packet is received* or when a *forwarding failure is detected*. Its execution can be logically subdivided into two phases, namely *route discovery* and *route maintenance*. A route discovery phase is initiated

Variable	Meaning
i	The current node executing the routing protocol
h	The source of a packet that is being processed at i
j	The destination of a packet that is being processed at i
$\mathbf{S}^i_j$	Set of next hop nodes at i to destination j
D^i_j	Shortest distance from i to destination j
ltu^i_{jk}	The time next hop node $k \in \mathbf{S}^i_j$ was last used for data relay for a packet destined to j
sn	A unique sequence number that identifies a routing message sent by some node h for j
$minfc^h_{ji}(sn)$	The smallest route request forward count received at i from source h searching for destination j
$\mathbf{P}^h_{ji}(sn)$	Set of nodes that sent route request with forward count equal to $minfc^h_{ji}(sn)$ received at i

TABLE I

NOTATIONS USED TO DESCRIBE THE PROTOCOL.

when a node searches for routes to a destination. A route maintenance phase is executed in reaction to network topological changes that affect active routes. We first describe the network model and state our assumptions. Table I lists all the variables that will be used in the following discussion. The Appendices contain a detailed algorithmic description of the protocol and the proof of correctness.

A. Network Model and Assumptions

A mobile ad hoc network is represented as a graph $\mathbf{G} = (\mathbf{N}, \mathbf{L})$, where $\mathbf{N}$ is the set of nodes and $\mathbf{L}$ is the set of edges or links. Every node $i \in \mathbf{N}$ has a unique identifier and can act both as a router or source. Let $\mathbf{N}^i$ be the set of neighbors of i defined as the set nodes where i has direct bi-directional connectivity. We can say that if $k \in \mathbf{N}^i$, then the link $(i, k) \in \mathbf{L}$.

The successor set at node i for each active destination j, denoted by $\mathbf{S}^i_j \subseteq \mathbf{N}^i$, is defined as the set of nodes that can be used by i as a next hop for packets to j. The length of any route from $k \in \mathbf{S}^i_j$ to j is D^k_j. A unique feature of CHAMP is that it only includes any node k in $\mathbf{S}^i_j$

if the length of the route from k to j is equal to the shortest path.

B. *Protocol Messages*

Nodes using CHAMP exchange three types of control messages, namely, route request (RREQ), route reply (RREP), and route error (RERR). The convention TYPE$(f_1, f_2, ..., f_n)$, where $f_1, f_2, ...$ are fields of the message, is used to represent these messages.

- RREQ(h, j, sn, k, fc, pr): A route request message from h searching for j. The field sn is the sequence number; k is the previous hop node; fc is the forward count; and pr is the propagation range in terms of hops. The tuple (h, sn) is used to distinguish between route requests coming from the same source h.
- RREP$(h, j, sn, k, \mathbf{P}, hc)$: A route reply message to the request RREQ from h searching for j. The field sn is the sequence number (taken from the RREQ); k is the previous hop node; $\mathbf{P}$ is the set nodes that can accept this message; and hc is the hop count from k to j plus one.
- RERR$(k, \mathbf{R})$: A route error message received from node k. The error resulted from the forwarding failure of some data packets whose header information (h, j, sn) are included in the set $\mathbf{R}$.

C. *Data Structures*

Each node in the network maintains two data structures: a *route cache* to contain forwarding information; and a *route request cache* for storing recently received and processed RREQ. The route cache at node i is a list containing an entry for every active destination j. Each entry contains the following: destination identifier (j), distance to the destination (D^i_j), set of successor or next hop nodes to the destination ($\mathbf{S}^i_j$), the last time each successor node k was last used for forwarding ($ltu^i_{jk}, \forall k \in \mathbf{S}^i_j$), and number of times each successor node k is used ($use^i_{jk}, \forall k \in \mathbf{S}^i_j$). A route entry which has not been used for more that $RouteLifeTime$ seconds is deleted.

The route request cache at node i is a list containing an entry for every unique route request received and processed. Each entry contains the following: source identifier of the route request (h), identifier of the node being searched (j), sequence number of the request (sn), minimum forward count ($minfc^{h}_{ji}(sn)$), set of previous hop nodes or the nodes that forwarded the same request with $fc = minfc^{h}_{ji}(sn)$ ($\mathbf{P}^{h}_{ji}(sn)$), and status of the route request ($status^{h}_{ji}(sn)$) which can either be $Replied$ or $NotReplied$.

In addition to the two data structures, every node also maintains two first-in first-out buffers: a *send buffer* for storing packets waiting for routes; and a *data cache* for storing recently forwarded data packets.

D. Forwarding Data Packets

In CHAMP, data packets are identified by the source identifier and a source-affixed sequence number. Every data packet also contains the identifier of the previous hop in its header, serving as a “pointer” to the upstream node which cached the same packet.

When forwarding a data packet, a node i chooses the least used next hop neighbor $k \in \mathbf{S}^{i}_{j}$. This spreads the packets over all the routes in a round-robin fashion. Node i then saves a copy of the packet in its data cache, sets the previous hop field to its address, and then forwards the data packet to the chosen next hop. If i has no route to j and it is the source of the packet, it saves the packet in its send buffer and performs a route discovery. However, if i is not the source, it simply drops the packet and broadcasts a RERR containing the header information (source, destination, previous hop and sequence number) of the dropped packet.

E. Route Discovery

Route discovery is initiated by a source node h that has a data packet for j but has no available route to j. Node h sends a RREQ with sn set to the next available sequence number, $fc = 0$, and pr to the network diameter. It waits for replies in the form of RREP packets. If no reply is received after a waiting period, h sends a new RREQ. To avoid congesting the network, sending of the request is separated by an increasing interval using *binary exponential back off*. If no reply is received after a specified number of retries, the upper layer is informed that j is unreachable.

1) Route Request Propagation: The first time node i receives a RREQ from h to j with sequence number sn, it initializes $minfc^{h}_{ji}(sn)$ to fc and $\mathbf{P}^{h}_{ji}(sn)$ to the previous hop of the message. Node i then rebroadcasts the RREQ with fc increased by one. Every time i subsequently receives a RREQ, there are two possibilities:

- If the request traversed a path of the same length from h to this node ($fc = minfc^h_{ji}(sn)$), the previous hop of the message is included in $\mathbf{P}^h_{ji}(sn)$.
- If the request traversed a shorter path from h to this node ($fc < minfc^h_{ji}(sn)$), $minfc^h_{ji}(sn)$ is reset to fc, $\mathbf{P}^h_{ji}(sn)$ is reset to the previous hop of the message and the request is retransmitted.

When the destination node j hears of the RREQ, it immediately sends back a RREP if the request is new or $fc \leq minfc^h_{jj}(sn)$. When the destination node generates a RREP, it sets $\mathbf{P}$ to the previous hop node of the latest RREQ received and hc is set to 0.

2) Route Reply Propagation: A node i receiving a RREP processes the reply if its identifi er is included in $\mathbf{P}$, or if it is the requesting source h; otherwise, it ignores the reply.

Node i accepts the route in the RREP if the number of routes to j is less than $MaxRoutes$ and $hc \leq D^i_j$ or the last time a route to j was used is greater than $RouteFreshTime$. (If $hc < D^i_j$, S^i_j is reset to contain the previous hop of RREP.) Upon acceptance of the route, it sets its distance D^i_j to hc and the use count to zero or the minimum use count minus one, whichever is greater.

Regardless of whether i accepted the reply or not, the RREP is forwarded to its upstream nodes ($\mathbf{P}^h_{ji}(sn)$) with hc incremented by one if the corresponding RREQ has been not replied. Note that the resulting routes at i have equal lengths, in particular, the successor set is composed of $\mathbf{S}^i_j = \{k | D^k_j = D^i_j - 1, k \in N^i\}$.

F. Example

Figure 1 shows the propagation of a route request from a source S to a destination D. In (a), S broadcasts a RREQ with $fc = 0$. The number on top of each solid arrow is the value of fc. In (b), A, B, and C sets their $\mathbf{P}^S_{Di}$ to $\{S\}$ and $minfc^S_{Di}$ to 0 (for $i = A, B, C$). The dotted arrow represents $\mathbf{P}^S_{Di}$ while the number on top represents $minfc^S_{Di}$. A forwards the request, this time with $fc = 1$. In (c), F sets $\mathbf{P}^S_{DF}$ to $\{A\}$ and $minfc^S_{DF}$ to 1. Note that B also hears the request but does not include A in its $\mathbf{P}^S_{DB}$ since $minfc^S_{DB} = 0 < fc = 1$. B forwards the request it received from S, with $fc = 1$.

In (d), G sets its $\mathbf{P}^S_{DG}$ to $\{B\}$ while F includes B in its $\mathbf{P}^S_{DF}$ since $minfc^S_{DF} = fc$. A and C ignores the request. C forwards the request it received from S, with $fc = 1$. In (e), G includes

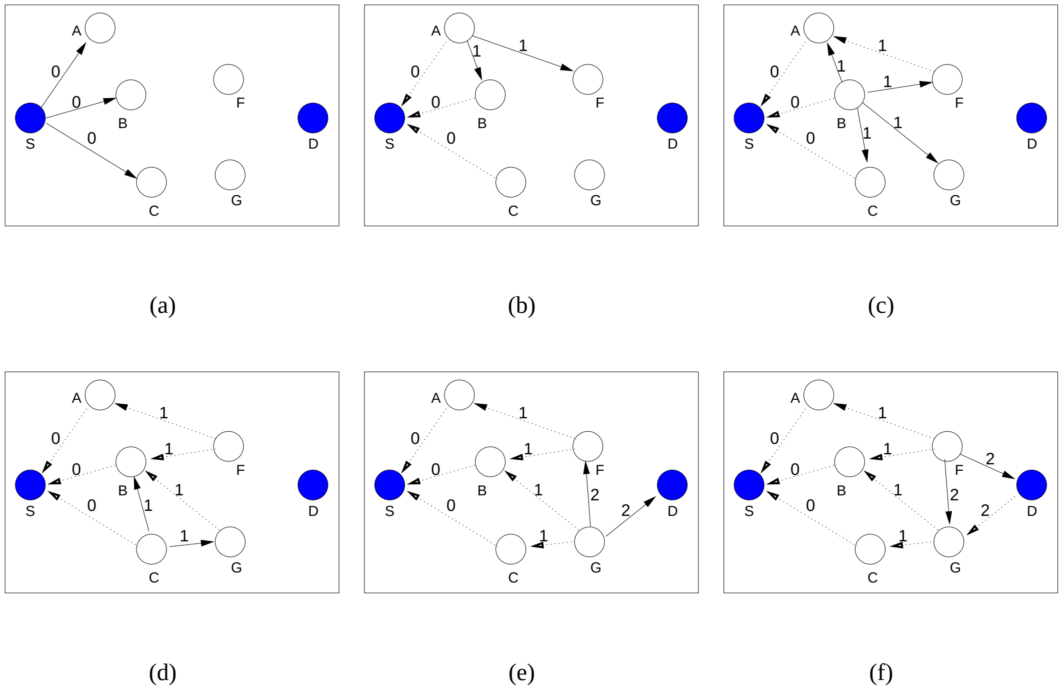


Fig. 1. Route request propagation from source S to destination D.

C in its $\mathbf{P}_{DG}^{S}$ since $minfc_{DG}^{S} = fc$. B ignores the request. G forwards the request it received from B, with $fc = 2$. Finally in (f), D hears the request from G, sets its $minfc_{DD}^{S}$ to 2, and sends back a RREP to G (not shown here). Meanwhile, F forwards the request it received from A, with $fc = 2$. D receives this again and will also reply to it since $minfc_{DD}^{S} = fc$.

Note that at the end of the RREQ propagation (Figure 1(f)), we see the path that the route replies will pass through, which, in effect constitute the routes from S to D. The actual propagation of route replies is shown in Figure 2.

Figure 2 shows the propagation of a route reply from a destination D to a source S. Note that Figure 2(a) is the end of the route request propagation discussed in the previous subsection. In (a), D sends a RREP with hop count $hc = 0$ for G. The number on top of the solid arrow represents hc. In (b), G accepts the reply and sets $D_{D}^{G} = 0$ and $\mathbf{S}_{D}^{G} = \{D\}$. D sends another RREP with hop count $hc = 0$ for F. G also forwards a RREP with $hc = 1$ to B and C. In (c), F accepts the reply from D and sets $D_{D}^{F} = 0$ and $\mathbf{S}_{D}^{F} = \{D\}$. B accepts the reply from G and sets $D_{D}^{B} = 1$ and $\mathbf{S}_{D}^{B} = \{G\}$. C also accepts the reply from G and sets $D_{D}^{C} = 1$ and $\mathbf{S}_{D}^{C} = \{G\}$. F forwards the RREP with $hc = 1$ to A and B. C forwards a RREP with $hc = 2$ to S.

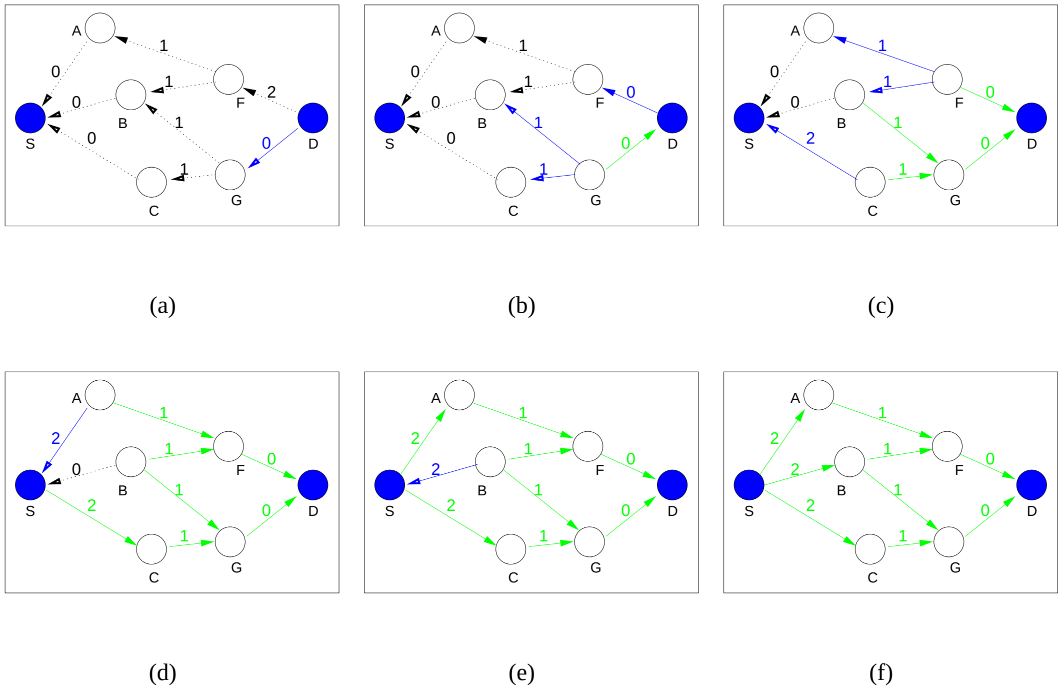


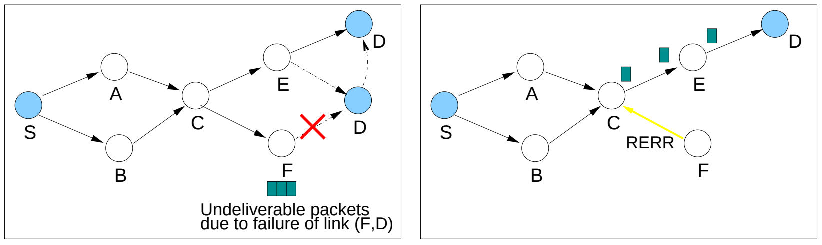
Fig. 2. Route reply propagation from destination D to source S

In (d), A accepts the reply from F and sets $D_D^A = 1$ and $\mathbf{S}_D^A = \{F\}$. B accepts the reply from F since $D_D^B = hc = 1$, $\mathbf{S}_D^B$ becomes $\{G, F\}$. Source S accepts the reply from C and sets $D_D^S = 2$ and $\mathbf{S}_D^S = \{C\}$. A forwards a RREP with $hc = 2$ to S. In (e), Source S accepts the reply from A since $D_D^S = hc = 2$, $\mathbf{S}_D^S$ becomes $\{A, C\}$.B forwards a RREP with $hc = 2$ to S. In (f), Source S accepts the reply from B since $D_D^S = hc = 2$, $\mathbf{S}_D^S$ becomes $\{A, B, C\}$.

At the end of the route reply propagation (Figure 2(f)), source S acquired three equal length routes to D, that is $\mathbf{S}_D^S = \{A, B, C\}$. If the allowed number of routes that can be stored per destination is just two, then S will only store the first two routes, namely the routes through A and C.

G. Route Maintenance

In CHAMP, nodes rely on data packet acknowledgment provided by the link layer to determine the state of a link. A link (i, k) is declared as "down" when node i does not receive any acknowledgment from the next hop node k after forwarding a data packet to k. When this occurs, k is deleted from $\mathbf{S}_j^i$. If i has another route to j, it forwards all affected packets through



(a) (b)

Fig. 3. Route maintenance and packet salvaging from data cache

this route. If there is no other route to j, node i broadcasts a RERR containing the header information (source, destination, sequence number) of all data packets (excluding packets that are originating from this node) that cannot be delivered as a result of the link failure. If there are undeliverable packets originating from this node, i performs a new route discovery for j.

H. Packet Salvaging from Data Cache

Recall that it is possible for the RERR to contain information about one or more data packets. Before parsing the RERR received from k, i creates a new RERR message that it will propagate upstream should it fail to salvage any packet. For every packet referred to in the message, node i performs the following:

Node i deletes k from $\mathbf{S}_j^i$, where j is the destination of the packet. If i originated the referred packet and it has no other route j, it initiates a new route discovery if there is no other route discovery for j is in progress. If i has other routes to j and it has a copy of the data packet in its data cache, it forwards the data packet according to the data forwarding rule (see Section III-D). If i has no other route to j and it has a copy of the data being referred to in its data cache, it removes the referred data packet from its data cache and adds the data packet header information in the RERR it created. If i does not have the packet in its data cache and it is the previous hop of the packet, i adds the data packet header information in the RERR it created. If after parsing the RERR node i fails to salvage one or more data packets, it broadcasts the RERR it created.

Figure 3 shows the route maintenance, including packet salvaging from data cache. In (a), D moves away from the range of F, breaking link (F, D). As a result, F has three undeliverable

Parameter	Broch, et al. [10]	Das, et al. [17]	This Paper
Simulator	ns-2	ns-2	ns-2
Simulation time	900 s	500 s	600 s
Network area	1500x300	2200x600	1500x600
Node count	50	100	100
Node speed range	0-20 m/s	0-20 m/s	10-30 m/s
CBR sources	10, 20, 30	10, 20, 40	10, 20, 30
Payload size	64 bytes	512 bytes	512 bytes
Sending rate	4 pkt/s	4 pkt/s, 3 pkt/s	4 pkt/s

TABLE II

SIMULATION PARAMETERS IN COMPARISON WITH TWO PREVIOUS STUDIES.

packets. In (b), F generates a RERR containing the header information (source, destination, sequence number and previous hop) of the three undeliverable packets to D and broadcasts it. C receives the RERR and salvages all packets that are still in its data cache. It sends the packets through E.

IV. PERFORMANCE EVALUATION

In this section, we present extensive performance evaluation through simulations for two important objectives: (i) to determine the behavior of CHAMP as a function of different protocol parameters and under different scenarios or conditions; and (ii) to determine how CHAMP performs compared with existing ad hoc routing protocols.

A. Simulation Parameters

All simulations are conducted using the simulator program *ns-2* [18] with wireless extensions from the Monarch Project [19]. This program was used in prior performance studies of ad hoc routing protocols [10], [12], [11], [17], [20], [21]. We divide the simulations into two parts: The first part evaluates the effect of the data cache size and number of stored routes to every destination on the performance of CHAMP. The second part is a performance comparison between CHAMP, AODV and DSR.

The network consists of 100 nodes in a 1500 m $\times$ 600 m area. Nodes move according to the "random way-point" model [10]. Six different pause times are used: 0, 30, 60, 120, 300, and 600 seconds. A zero pause time implies that the nodes are continuously moving while a 600-second pause time means that nodes are at rest for the entire simulation duration. Nodes move at randomly chosen speeds between 10 and 30 m/s.

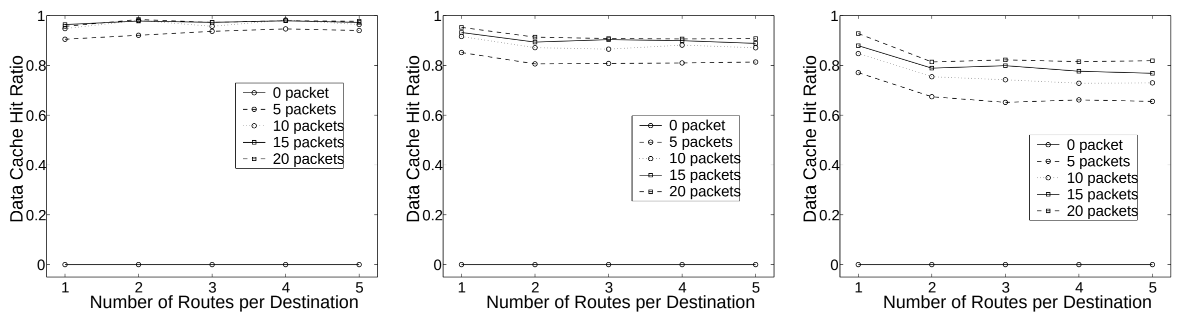
Source nodes are constant-bit-rate (CBR) sources sending four packets per second. The packet size is set to 512 bytes. A source can start sending packets at a randomly chosen time between 0 and 50 seconds. The two-ray ground reflection approximation is used to model radio propagation. The wireless interface device is modeled after a 2 Mb/s Lucent WaveLan card [22]. The IEEE 802.11 Distributed Coordination Function (DCF) [13] is used as the medium access protocol. The interface queue is a 50-packet drop-tail priority queue.

A summary of some of the simulation parameters, in comparison with two previous studies are shown in Table II. Note the difference in the range of node speed. To gain insight as to how this parameter affects the network topology, we calculated the number of link changes every second. The study by Broch, et al. [10] reported an average of 11835 link changes in 900 seconds. This is equivalent to 13 link changes/second. In this study, the scenario created 75240 link changes for a duration of 600 seconds. This is equivalent to 125 link changes/second, or around 10 times more compared to the scenarios used in [10].

B. Performance Metrics

In CHAMP, every node maintains a first-in first-out data cache for storing forwarded packets for future use. To determine the effectiveness of this cache, we measure the *data cache hit ratio*, or the total number of successful cache "reads" divided by the total number of cache reads.

In the performance comparison, the protocols are evaluated using the following criteria: (i) *packet delivery ratio* - the total number of packets delivered divided by the total number of packets sent; (ii) *end-to-end delay* - the delay for every packet delivered; and (iii) *routing overhead* - the total number of routing messages sent and forwarded every second.



(a) 10 sources (b) 20 sources (c) 30 sources

Fig. 4. Data cache hit ratio.

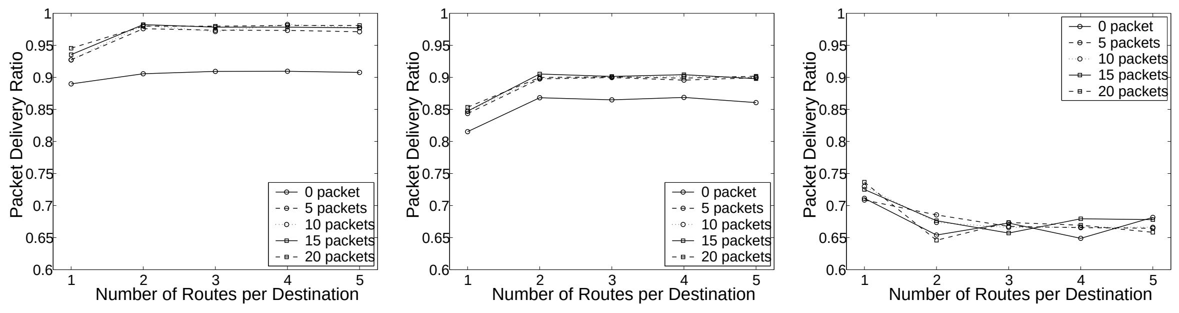
C. CHAMP Performance

The performance of CHAMP is influenced by two parameters, namely, *data cache size* and *number of stored routes per destination*. We vary these parameters and determine their effects on the packet delivery ratio, end-to-end delay, routing overhead, and data cache hit ratio.

1) Impact of Data Cache Size: In CHAMP, each node maintains exactly one data cache regardless of the number of connections or destinations that node serves. A larger data cache means that more data packets can be stored at any given time. Hence, larger data cache implies cache accesses are more probably successful. This hypothesis is highly supported by the results.

Figure 4 shows the data cache hit ratio. In general, the data cache hit ratio is high, greater than 70% and shows small incremental increase as the data cache size is incrementally increased by five. At ten sources, regardless of the data cache size, more than 90% of the accesses are successful. The hit ratio peaks at 97% when the data cache size is five or more. At 20 sources, the plots become more spread. With a data cache of five packets, the hit ratio is 80% which is the lowest. With a data cache of 20 packets, the hit ratio is at 95% which is the highest. At 30 sources, the hit ratio ranges from 70% when the data cache size is five to 90% when the data cache is 20.

These results suggest the existence of temporal locality in the dropped packets. Recall that in CHAMP, whenever a node fails to forward a data packet to its next hop, it drops the packet and sends a route error message containing the header information (source, destination, and sequence



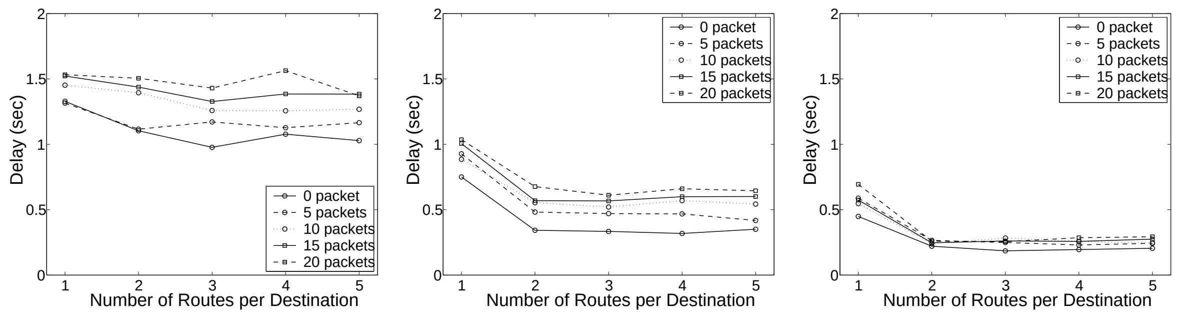
(a) 10 sources (b) 20 sources (c) 30 sources

Fig. 5. Packet delivery ratio.

number) of the dropped packets. Every node receiving this message then accesses its data cache to see if it can salvage any of the packets. The high data cache hit ratio suggests that oftentimes, a recently dropped packet downstream is still in the data cache of the nodes upstream. Hence, a node does not need a large data cache since with temporal locality, only the most recently cached packets are accessed and used for packet salvaging. Furthermore, this implies that no matter how large the data cache is, the relatively old packets are never utilized.

Figure 5 shows the packet delivery ratio with varying number of sources. At 10 and 20 sources, the packet delivery ratio without data caching reaches a peak of 90% and 85%, respectively. With data caching (various data cache sizes of 5, 10, 15 and 20), the packet delivery ratio reaches a maximum of 99% and 90%, respectively. The use of data cache significantly increases the packet delivery ratio, 7-8% at 10 sources and 5-7% at 20 sources. At 30 sources, the use of data cache contributes only around 2-3% to the packet delivery ratio. A noticeable feature of the plot is that multipath routing seems to adversely affect the protocol's performance. We discuss this result later.

These results show that caching of data packets in combination with at least two routes, can indeed improve packet delivery. However, contrary to our hypothesis, there is no evidence suggesting that incrementally increasing the data cache size results in incremental increase of delivered packets. Looking specifically at the 20-source case, the packet delivery ratio for data cache sizes of 5, 10, 15 and 20 are almost the same at 90% (the maximum variation is around 2%). From the data cache size of five packets, there is no observable increase in packet delivery



(a) 10 sources (b) 20 sources (c) 30 sources

Fig. 6. End-to-end delay.

as the data cache size is further increased. The same is observed at the 30-source case (Figure 5(c)). The packet delivery stays at around 66% from the data cache size of five packets up to 20 packets.

Figure 6 shows the end-to-end delay. At 10 sources, the increase in the delay is not significant. At 20 and 30 sources, the delay shows incremental increase as the data cache is incrementally increased. The delay increases by as much as half a second from the case where there is no data cache to the case where the data cache has 20 packets. This increase is not at all unexpected. Recall that in CHAMP, when a node fails to salvage a data packet from its data cache, it propagates the route error message upstream. These upstream nodes are farther from the link failure and destination. If the data cache is small, then only nodes close to the route failure may have the packet in their respective caches. On the other hand, if the data cache is large, many nodes, including those that are far from the route failure (and therefore farther from the destination) may still have the packet in their caches. Consequently, if these far nodes can salvage packets, delays should increase.

2) Impact of Using Multiple Routes: The success of data caching depends on the availability of alternative routes to the destination of the packet. Having a pertinent packet in the data cache does not guarantee that a salvaged packet will reach the destination. Logically, having more routes should result in increased packet delivery ratio since there will be a greater chance for salvaged packets to get delivered to their respective destinations.

We first consider how the number of stored routes affect the packet delivery ratio (see Figure

5). At 10 and 20 sources, when there is only one route, the packet delivery ratio reaches a peak of 95% and 84%, respectively. When there are multiple routes (2, 3, 4 and 5), the packet delivery ratio increases by 3-5% for both 10 and 20 sources. Note that the increase due to multiple path routing is smaller in comparison with the contribution of data caching. However, the effectiveness of data caching is greater when there are multiple routes.

Specifically looking at the 10-source scenario, data caching trebled its contribution from 2-3% when there is only one route to 7-8% when there are multiple routes. We expected that when nodes maintain only one route to every destination, data caching should not cause any improvement in the packet delivery ratio. However, recall that when a *data source* receives a route error message and it has no other route, it moves the data packet being referred by the route error from its data cache (if it is still there) to its send buffer and performs a new route discovery. If the route discovery is successful, then the salvaged packet gets retransmitted. Hence, the 2-3% increase in packet delivery can be attributed to source retransmissions.

Earlier, we hypothesized that having more routes should result in some increase packet delivery ratio. However, the results do not support this prediction. Observe that at 20 sources, as the number of stored routes is increased from 2, 3, 4, then 5, there is a no observable increase in the packet delivery ratio. (We did not consider the 10-source scenario since the packet delivery ratio already reaches its peak even when there are two routes.) We found two reasons for this behavior. First, note that the parameter specifies the maximum number of routes, and most nodes do not actually obtain this much number of routes especially when it is high. The number of stored routes is actually less than this, depending on the topology. Second, for a node to successfully retransmit a salvaged packet, it requires only one alternative route.

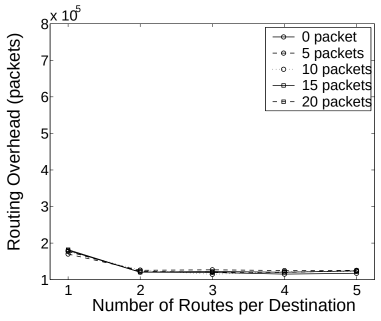
While the presence of multiple path partly increases the packet delivery at 10 and 20 sources, at 30 sources (see Figure 5(c)), it does the opposite. In this scenario, the use multiple paths cause a 5-6% drop in the packet delivery ratio. This result is surprising since we expected (and observed in the 10- and 20-source scenarios) that multiple path routing increases the packet delivery ratio. Two reasons can be attributed for this unexpected behavior. Recall that CHAMP's route discovery allows non-disjoint routes to be discovered and used. In the presence of congestion, which is the second reason, the inefficiency of non-disjoint routes becomes apparent.

In a congested network, the medium access control protocol encounters many collisions when sending packets. After so many attempts, the MAC protocol gives up and informs the routing protocol that the packet cannot be forwarded to the next hop. Since the routing protocol is not aware of congestion, it always assumes that a forwarding failure is due to link failure even when this is not the case. Consequently, it deletes the routes passing through the failed link and generates a route error message and broadcasts it. The problem is that when routes are non-disjoint, a single route error can cause the deletion of many other routes. In the 10- and 20-source scenarios, most of the reported errors are correct, that is, the route is indeed broken. At 30 sources, a significant number of reported errors, more than 20%, are incorrect. With non-disjoint routes, these faulty errors are further multiplied. Consequently, the multiplied number of route error messages adds to the already congested network, possibly causing more faulty errors to be generated.

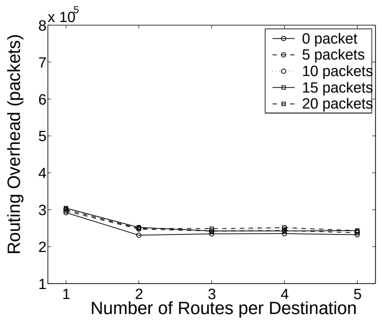
The presence of at least two routes significantly reduces the end-to-end delay (see Figure 6. At 10 and 20 sources, the end-to-end delay drops to almost half the delay when the number of routes is increased to two or more. This benefit can be attributed to the per packet, round-robin load distribution used by CHAMP. However at 30 sources, no gain is achieved due to excessive network congestion and *route coupling*.

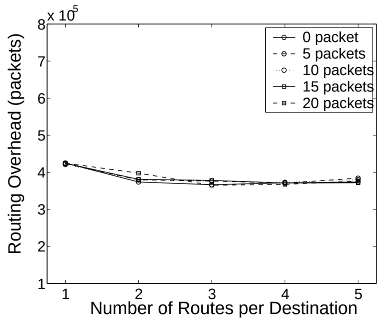
Two routes are said to be coupled if they have common nodes or links [23]. In wired networks, route coupling can occur only to non-disjoint routes whereas in wireless networks, it can occur even to node-disjoint routes due to the broadcast nature of transmissions. The adverse effect of route coupling is to decrease the efficiency of multiple routes since transmissions along a route affects the transmissions at the other route, just like in a single-path route. In a congested environment, multiple path routing of CHAMP does not ease congestion due to coupling. We believe that even if CHAMP is modified to only discover node-disjoint routes, this problem will not be completely solved.

The routing overhead of CHAMP (see Figure 7) is only affected by the number of routes stored. At 10 and 20 sources, it decreases significantly when the number of routes is increased from one to two and stays from hereon. This result is natural since the presence of more than one route avoids route discoveries. Recall that CHAMP performs a new route discovery only



(a) 10 sources





(b) 20 sources (c) 30 sources

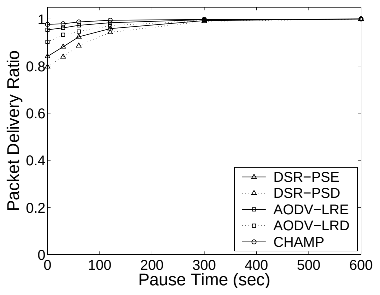
Fig. 7. Routing overhead.

when all routes to an active destination are lost. At 30 sources, there is no observable decrease in the routing overhead. Again, this can be attributed to congestion.

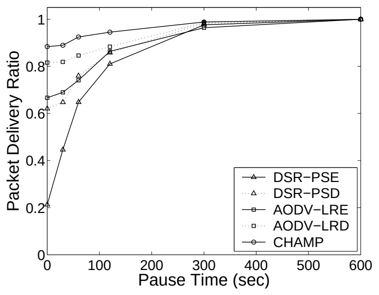
D. Performance Comparison

We compare the performance of CHAMP (five-packet data cache, two routes per destination configuration) with DSR and AODV. Many studies show that AODV and DSR perform better compared to proactive and multipath routing protocols in terms of packet delivery and routing overhead especially in the presence of mobility [10], [11], [24]. The reader is referred to Johnson and Maltz [5] for a detailed description of DSR and to Perkins and Royer [7] for a complete discussion of AODV.

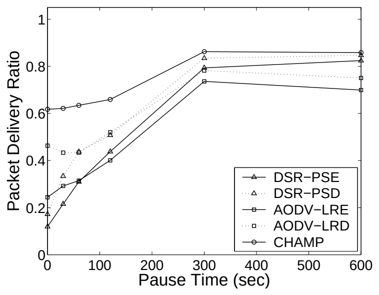
The default protocol parameter values for AODV and DSR reported in prior performance evaluation studies [10], [17] are used. For AODV, we simulate two cases, with local repair (i)



(a) 10 sources

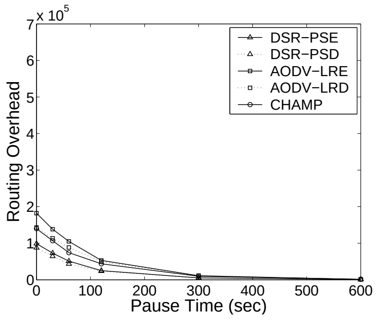


(b) 20 sources

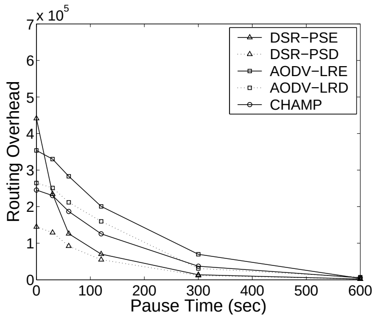


(c) 30 sources

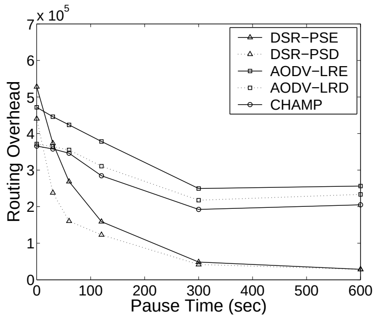
Fig. 8. Fraction of successfully delivered data packets as a function of mobility.



(a) 10 sources



(b) 20 sources



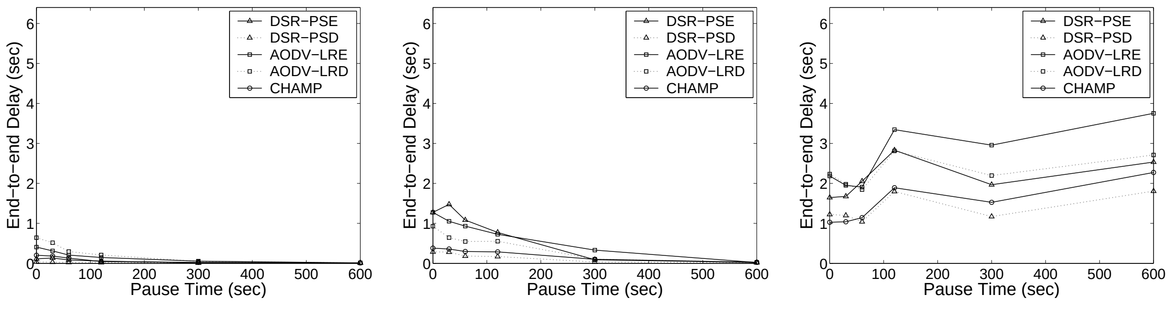
(c) 30 sources

Fig. 9. Routing overhead in terms of packets as a function of mobility.

enabled and (ii) disabled. We made modifications on the ns-2 implementation of AODV local repair to make it compliant to AODV draft specifications [25]. For DSR, we also simulate two cases, with packet salvaging (i) enabled and (ii) disabled. The protocols are simulated over ten randomly generated network topologies. For fairness, the same scenario and traffic patterns are used across protocols.

1) Discussion: Figures 8, 10 and 9 show the packet delivery ratio, end-to-end delay and routing overhead, respectively, for the simulations with varying number of CBR sources as a function of pause time. AODV-LRE denotes AODV with local route repair enabled while AODV-LRD refers to AODV with local route repair disabled. DSR-PSE and DSR-PSD refers to DSR with packet salvaging enabled and disabled, respectively.

All protocols show almost 100% packet delivery when nodes are stationary except when there



(a) 10 sources (b) 20 sources (c) 30 sources

Fig. 10. End-to-end delay as a function of mobility.

are 30 sources. Expectedly, packet delivery decreases as mobility increases. Likewise, it also decreases as the number of sources increases.

Noticeably, CHAMP shows the highest packet delivery ratio for all the number of sources and mobility rates. Its advantage is highest in higher node mobility and source count scenarios. At 10 sources, CHAMP delivers more than 98% regardless of mobility. AODV-LRE and AODV-LRD delivers 95% and 90%, respectively at 0 pause time. DSR-PSE and DSR-PSD show the least impressive performance, delivering 84% and 80%, respectively at 0 pause time.

At 20 sources, CHAMP edges AODV-LRD and AODV-LRE by at most 8% and 20%, respectively. CHAMP outperforms DSR-PSE and DSR-PSD by more than 65% and 25%, respectively. Once again, at 30 sources, CHAMP edges AODV-LRD and AODV-LRE by at most 20% and 45%, respectively. Finally, CHAMP outperforms DSR-PSE and DSR-PSD by at most 50%.

Note that local repair in AODV contributes at most 5% improvement in packet delivery when there are 10 sources. However, at 20 and 30 sources, the use of local repair significantly reduces the packet delivery by 15% and 30%, respectively. Packet salvaging in DSR also increases the packet delivery by at most 5% when there are 10 sources. But at 20 and 30 sources, it degrades the performance by 40% and 10%, respectively, at 0 pause time. The results for the 20- and 30-source scenarios are indeed surprising as these two optimizations are employed to improve the performance of the two protocols.

We first discuss the unexpected performance degradation caused by local repair in AODV. Recall that a node invokes local repair when it fails to forward a data packet to its next hop and the node is nearer to the destination than to the source of the packet. Without local repair, this undeliverable packet is simply discarded. With local repair, this packet (and other undeliverable packets) is stored in the forwarder's send buffer. If the repair is successful, then this packet (and other undeliverable packets) can be successfully forwarded.

The lower packet delivery ratio at 20 and 30 sources when local repair is used can be attributed to several factors. A major factor is the increase of dropped packets caused by local repair failure. Note that when some node i performs local repair for destination j, an upstream node k may continue forwarding packets to i destined to j. This causes packets destined to j to be "stranded" at node i. When the route repair fails, these stranded packets at i are simply dropped.

Another major factor is increased congestion caused by local repair overhead. Note that at 20 and 30 sources, the routing overhead (see Figures 9(b) and 9(c)) increases by 30% when local repair is enabled. This additional overhead causes signifi cant network congestion, which in turn causes a signifi cant number of packets to be dropped and end-to-end delay (see Figures 10(b) and 10(c)) to shoot up. Note that CHAMP's distributed packet salvaging scheme does not suffer from this problem.

Similar to local repair, packet salvaging in DSR is also meant to decrease packet loss caused by frequent route failures. Packet salvaging is performed by some node i that is upstream of a link failure. Without packet salvaging, node i drops all undeliverable packets that are intended to pass through the failed link. With packet salvaging, node i forwards these undeliverable packets through some alternative path. This alternative path may be a route that node i overheard. Note that unlike CHAMP, this alternative path is not refreshed periodically.

The success of packet salvaging is dependent on the availability of a *valid* alternative path at the node encountering the forwarding failure. The policy of no route expiration used by DSR causes an alternative path to be already invalid. This is most likely the case in situations where node mobility is higher since most routes are short-lived. As a result, salvaged packets are actually forwarded to incorrect paths. This wastes bandwidth and increases network congestion.

Another factor that caused the poor performance of packet salvaging is that similar to local repair, it can only be performed by a node that encounters a forwarding failure. This limits the effectiveness of these optimizations since when they fail, the node that invoked them has no other choice but to drop all undeliverable packets. If the route from a source to a destination involves many hops, dropping packets along a route is expensive in terms of bandwidth and energy. Indeed, when we performed simulations in narrow areas as that in [10], [17] to force long routes, CHAMP's performance edge over AODV and DSR in terms of packet delivery increased by more than 10%. Note that we selected the area of 1500 m x 600 m to provide fair comparison. Choosing a narrow area would lead CHAMP to have an unfair advantage.

In terms of end-to-end delay, all protocols show signifi cant increase as the number of sources increases. Likewise, the delays also rise as mobility rate increases (except at 30 sources.) In most scenarios, CHAMP exhibits the most consistent and relatively lower end-to-end delay.

The higher delay when local repair in AODV is used stems from the fact that packets stay for some time at a node performing a local repair. Note the difference: without local repair, a packet may only be delayed at the source (because of route discovery); with local repair, a packet may also be delayed at intermediate nodes (because of local repair). Another reason is that as mentioned in the preceding discussion, the network becomes congested at 20 and 30 sources because of the overhead generated by local repair itself.

For DSR, the delay is higher when packet salvaging is used. This is due to congestion and repetitive salvaging of certain packets. Note that in the *ns* implementation of DSR, there is no limit on the number of times a packet can be salvaged. Naturally, packets that have been salvaged many times experience higher delays.

With respect to routing overhead, at all sources and mobility rates, AODV-LRD and CHAMP generate almost the same amount of routing overhead. DSR-PSE shows the worst performance, more than 50% that of CHAMP. However, at low mobility rates (greater than 120 seconds of pause time), both DSR-PSD and DSR-PSE outperform CHAMP, AODV-LRE and AODV-LRD. This is the advantage of caching of overheard routes employed by DSR. At low mobility rates, since routes tend to be valid for longer periods, caching of overheard routes reduces the need for route discoveries.

These results confi rm that the routing overhead of CHAMP is comparable if not better than that of AODV and DSR. At zero pause times, CHAMP's routing overhead is less than 2/3 the overhead of DSR-PSE, DSR-PSD, and AODV-LRE. These results indicate an excellent achievement for

CHAMP since it is a multiple path routing protocol. Logically, CHAMP is expected to generate a greater number routing overhead compared to AODV or DSR since it discovers multiple routes.

The increase in overhead when local repair is used in AODV at 20 and 30 sources can be attributed to the excessive routing overhead caused by more frequent local repairs. Note that without local repair, only the sources can propagate route request packets. With local repair, any node can propagate route request packets. Coupled with frequent link failures, the overhead increases by at least 30% with local repair. This is the major disadvantage of using local repair when route changes are frequent.

The significantly higher routing overhead generated when packet salvaging is enabled in DSR at zero pause times can be traced to route error message overhead and no-route-expiration policy. In situations where nodes more mobile, routes become invalid quickly. Hence, salvaged packets are often relayed to erroneous routes. Recall that in packet salvaging in DSR, the salvaging node also sends a route error message back to the source. This source then performs a new route discovery that generates routing overhead. When a salvaged packet reaches a non-existent link, a route error message is once again propagated to the source. Once again, the source node may perform a new route discovery. Note that this mechanism of packet salvaging causes a source node to perform unnecessary route discoveries.

Before ending the discussion, we note that the load settings of 10, 20, and 30 sources do not fully utilize the 2 Mb/s channel capacity. Indeed, at 30 sources, the traffic generated is only 24% of the channel capacity. But even at this seemingly low normalized load, the network is already highly saturated. Das and others [17] found that using 802.11 MAC, the delivered throughput to the application is at most 2-5%. The three important reasons for the low channel utilization in ad hoc networks are: (i) The bandwidth consumed by successfully delivered data packets is proportional to the number of hops between the source and destination. The multihop nature of ad hoc connections therefore implies higher bandwidth consumption; (ii) Dropped packets also consume bandwidth, and that their consumed bandwidth is also greater when they are dropped further away from their source; and (iii) Routing protocols generate control messages which also consume significant bandwidth.

V. Related Work

Since the introduction by Wilkes [14] in 1965, caching is now used in many distributed systems, such as distributed file systems and the World Wide Web, and in various system layers [26]. In a previous study [27], we investigated the performance of cooperative packet caching and shortest multipath routing in ad hoc networks with moderate mobility. CHAMP already showed significant performance improvements over AODV and DSR both in CBR and TCP simulations. This paper further demonstrates the robustness of CHAMP over a wider range of node mobility.

One of the earliest works on adaptive multipath routing, proposed by Gafni and Bertsekas [28], uses a series of "link-reversals" to form a directed acyclic graph (DAG) rooted at the destination. One problem is that this protocol, which came to be known as GB, exhibits instability when the network is partitioned. Corson and Ephremides [29] developed a new algorithm based on this concept, termed Lightweight Mobile Routing (LMR) algorithm. Subsequently, the same authors introduced TORA [2]. LMR and TORA share many similarities including the route construction and route maintenance phases. TORA has an additional phase known as route deletion. One major drawback of TORA is the maintenance overhead. CHAMP eliminates such overhead by spreading packets over all the routes in a round-robin fashion.

Vutukury and Garcia-Luna-Aceves [30] used shortest multipath routes in MDVA, a proactive multipath distance-vector routing protocol for a network with changing topology. The emphasis of their work is to ensure loop-freedom and correctness at every instant by using loop-free invariant conditions. MDVA operates proactively and may not be suitable for highly mobile ad hoc networks. In this study, we developed a fast and simple way of discovering shortest multipath routes suitable for on-demand routing and does not generate large overheads.

Several studies have made multipath extensions to single-path routing protocols. Lee and Gerla [31] proposed an extension to AODV, called AODV-BR (AODV with back-up routes). AODV-BR utilizes multiple routes organized in a "fish-bone" structure and has been shown to improve protocol performance and robustness against mobility in light load conditions. However, unlike CHAMP, it does not perform well in highly loaded scenarios.

Marina and Das [32] proposed another multipath extension of AODV called AOMDV. It uses the notion of an "advertised hop count" to maintain multiple loop-free paths. This approach also lends some similarities to CHAMP's route discovery procedure. Simulation results show that AOMDV outperforms AODV by as much as 5% in terms of packet delivery, half the delay and 20% less routing overhead. Clearly, CHAMP shows significantly better performance than AOMDV. While CHAMP discovers non-disjoint multipath routes, AOMDV ensures the discovery of link-disjoint routes. A recent study have shown that non-disjoint multipath routes are more resilient and energy-efficient than disjoint routes [33].

Nasipuri and Das [34] proposed a multipath extension to DSR. As noted in many similar studies [17], [35], DSR's aggressive caching and no route expiration policy adversely affects both CBR and TCP connections in mobile scenarios. Hence, extending DSR to support multiple routes without addressing the caching problem may also aggravate this problem. Furthermore, there is a high probability that the alternative route is always stale, as it is never used until the primary route fails. CHAMP cleverly avoids this problem by using all the available routes in a round-robin fashion.

Gallager [36] introduced an algorithm for distributing load over multiple paths that leads to minimum delays. As the algorithm converges slowly, it is unsuitable or networks where load is highly dynamic. Vutukury and Garcia-Luna-Aceves [37] discussed an approximate solution to the problem. Krishnan and Silvester [38] found that a per packet load distribution policy provides the best results. However, this comes at the expense of out-of-order packet delivery, which can incur additional packet re-sequencing delays [39] and may adversely affect TCP connections [40]. Several work, namely that of Blanton and Allman [41], and Bohacek and others [42] tackled the issue of resiliency of TCP against packet reordering.

VI. Conclusions and Future Research

Reactive routing protocols perform well in mobile ad hoc networks due to their ability to react quickly to topological changes. Using local repair and packet salvaging improves the performance of the protocols but only at lower number of sources and lower mobility rates. At higher number of sources and higher node mobility, these optimizations exacerbate the performance of routing protocols by increasing network congestion due to additional routing overhead which consequently leads to reduced packet delivery. Their effectiveness is also limited because only the node that experiences the link failure can perform them.

We introduced cooperative packet caching, a strategy similar to cooperative caching aimed at reducing packet loss due to route breakdowns. We have shown that because of the property of temporal locality in dropped packets, a small data cache is sufficient to reduce the number of dropped packets thereby improving packet delivery. Although caching is now employed in many distributed systems, this is the first application in the network layer, particularly in mobile ad hoc networks.

A multiple path route discovery mechanism suitable for mobile ad hoc networks is also developed. The discovery mechanism selects non-disjoint shortest multipath routes of equal lengths. We have shown that having at most two routes for every destination is already the optimum. Unlike local route repair and packet salvaging, cooperative packet caching and shortest multipath routing do not entail undesirable side effects that may worsen routing performance.

We have shown that by using the above-mentioned values for the data cache size and number of stored routes per destination, significant improvement can be obtained over AODV-LRE, AODV-LRD, DSR-PSE and DSR-PSD especially in more dynamic and higher load scenarios. In terms of packet delivery, CHAMP outperforms AODV-LRE and AODV-LRD at most 15%. CHAMP edges DSR-PSE and DSR-PSD more than 50%. At high mobility rates, CHAMP generates routing overhead comparable to that of AODV-LRD and less than 2/3 that of AODV-LRE and DSR-PSE. In most scenarios, CHAMP exhibits a consistent and relatively lower end-to-end delay.

Caching of data packets at the network layer is a new technique for improving robustness and it needs further investigation. One possible direction is the use of other cache placement and replacement policies. While the simulations performed in this paper were quite extensive, further research is required to determine the performance of CHAMP. As CHAMP is designed to solve real-world ad hoc routing problems, it is important to test the behavior of CHAMP in real mobile ad hoc networks.

APPENDIX

FORMAL PROTOCOL DESCRIPTION

In the following algorithmic description, the send buffer at node i is denoted as $SendBuffer_i$, the data cache at node i is denoted as $DataCache_i$, and the route request cache at node i is denoted as $RqCache_i$. The parameter $CurrentTime$ specifi es the current time at that node, which may be unequal with the current time at other nodes. This implies that the protocol does not need time synchronization for proper operation.

The symbol "$\leftarrow$" denotes an assignment operation while the symbol "$\Leftarrow$" denotes either an insertion or deque operation on a queue or data structure. The command **broadcast** means that a sent packet may be processed by all the neighbors of the sending node. The command **unicast** means that a sent packet is processed by a specifi ed node.

A. Data Forwarding

```
procedure process_data(DATA$(h, j, sn)$)
00 begin
01     if $(j = i)$ then
02         send DATA$(h, j, sn)$ to upper layer;
03         exit;
04     endif
05     if $(\mathbf{S}_j^i = \phi)$ then
06         if $(i = h)$ then
07             $SendBuffer_i \Leftarrow$ DATA$(h, j, sn)$;
08             call route_discovery$(j)$;
09         else
10             broadcast RERR$(i, \{(h, j, sn)\})$;
11         endif
12     else
13         $DataCache_i \Leftarrow$ DATA$(h, j, sn)$;
14         unicast DATA$(h, j, sn)$ to $k$,
14             where $use_{jk}^i = min\{use_{jl}^i\}, \forall l \in \mathbf{S}_j^i$;
15         $use_{jk}^i \leftarrow use_{jk}^i + 1$;
16         $ltu_{jk}^i \leftarrow CurrentTime$;
17     endif
```

```
18 end
```

B. Reaction to Data Forwarding Failure

```
procedure forwarding_failure( )
00 begin
01     $\mathbf{R}_i \leftarrow \phi$;
02     for every undeliverable DATA$(h, j, sn)$
03         if $(\mathbf{S}^i_j \neq \phi \vee h = i)$ then
04             call process_data(DATA$(h, j, sn)$);
05         else
06             $\mathbf{R}_i \leftarrow \{\mathbf{R}_i, (h, j, sn)\}$;
07         endif
08     endfor
09     if $(\mathbf{R}_i \neq \phi)$ then
10         broadcast RERR$(i, \mathbf{R}_i)$;
11     endif
12 end
```

C. Handling Route Request

```
procedure process_rreq(RREQ$(h, j, sn, k, fc, pr)$)
00 begin
01     if $(h = i)$ then exit;
02     else if $(j = i)$
03         if ( $(h, j, sn) \notin RqCache_i \vee fc \leq minfc^h_{ji}(sn)$)
03         then
04             $minfc^h_{ji}(sn) \leftarrow fc$;
05             broadcast RREP$(h, j, sn, i, \{k\}, 0)$;
06         endif
07     else if ( $(h, j, sn) \notin RqCache_i$ /*RREQ  is new*/ )
```

```
08          $minfc^h_{ji}(sn) \leftarrow fc$; $\mathbf{P}^h_{ji}(sn) \leftarrow \{k\}$;
09          $status^h_{ji}(sn) \leftarrow NotReplied$;
10          **if** $(\mathbf{S}^i_j \neq \phi \wedge k = h \wedge k \notin \mathbf{S}^i_j \wedge pr = 0)$ **then**
11              **broadcast** RREP$(h, j, sn, i, \mathbf{P}^h_{ji}(sn), D^i_j + 1)$;
12              $status^h_{ji}(sn) \leftarrow Replied$;
13          **else if** $(pr > 0)$
14              **broadcast** RREQ$(h, j, sn, i, fc + 1, pr - 1)$;
15          **endif**
16      **else if** ( $(h, j, sn) \in RqCache_i$ /*RREQ  is not new*/ )
17          **if** $(fc < minfc^h_{ji}(sn))$ **then**
18              $minfc^h_{ji}(sn) \leftarrow fc$; $\mathbf{P}^h_{ji}(sn) \leftarrow \{k\}$;
19              **broadcast** RREQ$(h, j, sn, i, fc + 1, pr - 1)$;
20          **else if** $(fc = minfc^h_{ji}(sn) \wedge$
20          $status^h_{ji}(sn) = NotReplied)$
21              $\mathbf{P}^h_{ji}(sn) \leftarrow \{\mathbf{P}^h_{ji}(sn), k\}$;
22          **endif**
23      **endif**
24  **end**
```

D. Handling Route Reply

```
**procedure** *process_rrep*(RREP$(h, j, sn, k, \mathbf{P}, hc)$)
00  **begin**
01      **if** $(j = i)$ **then exit**;
02      **else if** $(h = i \vee i \in \mathbf{P})$
03          **if** $(hc < D^i_j \vee$
03          $CurrentTime - \max\{ltu^i_{jm} | \forall m \in \mathbf{S}^i_j\} >$
03          $RouteFreshTime)$ **then**
04              $D^i_j \leftarrow hc$; $S^i_j \leftarrow \phi$;
05          **else if** $(hc > D^i_j \vee$
05          $(hc = D^i_j \wedge |\mathbf{S}^i_j| \geq MaxRoutes))$
06              goto line 10;
07          endif
08          $ltu^i_{jk} \leftarrow CurrentTime$;
09          $use^i_{jk} \leftarrow \min\{use^i_{jm} | \forall m \in \mathbf{S}^i_j\} - 1$; $\mathbf{S}^i_j \leftarrow \{\mathbf{S}^i_j, k\}$;
10          if $(h \neq i \wedge status^h_{ji}(sn) = NotReplied)$ then
11              broadcast RREP$(h, j, sn, i, \mathbf{P}^h_{ji}(sn), D^i_j + 1)$;
12              $status^h_{ji}(sn) \leftarrow Replied$;
13          endif
14      endif
15  end
```

E. Handling Route Error

```
procedure process_rerr(RERR$(k, \mathbf{R})$)
00  begin
01      $\mathbf{R}_i \leftarrow \phi$;
02      for every $(h, j, sn) \in \mathbf{R}$
03          $\mathbf{S}^i_j \leftarrow \mathbf{S}^i_j - \{k\}$;
04          if $((h, j, sn) \in DataCache_i)$ then
05              DATA$(h, j, sn) \Leftarrow DataCache_i$;
06              if $(\mathbf{S}^i_j \neq \phi \vee h = i)$ then
07                  call process_data(DATA$(h, j, sn)$);
08              else
09                  $\mathbf{R}_i \leftarrow \{\mathbf{R}_i, (h, j, sn)\}$;
10              endif
11          endif
12      endfor
13      if $(\mathbf{R}_i = \phi)$ then
14          broadcast RERR$(i, \mathbf{R}_i)$;
```

```
15      endif
16 end
```

F. *Proof of Correctness*

To prove the correctness of CHAMP, the route discovery mechanism must be shown to result in the discovery of loop-free routes. The proof is subdivided into *properties* and *theorems*.

Property 1: The number of route request transmissions is finite and is bounded by the number of nodes in the network.

Proof: Recall that route requests are uniquely identified by the source-destination-sequence number triple. Route requests are only retransmitted at lines 14 and 19 of *process_rreq*. Line 14 is executed when a route request is new while line 19 is carried out when node i receives the same route request but with shorter distance to the source. In the worst-case scenario, line 19 is executed D_i^h times, where D_i^h is the initial distance from source h to the current node i. This situation occurs when node h moves *closer* towards node i (or vice versa) faster than the route reply back-propagation. The number of times a route request is transmitted is therefore bounded by the summation of $D_i^h \forall i \in \mathbf{N}, i \neq j$. Since D_i^h is always less than $|\mathbf{N}|$ anywhere in the network, then the number of times a route request is transmitted is less than $|\mathbf{N}|^2$ times. ■

Property 2: The graph formed by the propagation of route request from any source is acyclic.

Proof: The proof is by contradiction. Suppose that there is a cycle that includes nodes i and k as a result of route request propagation from a source h. A loop or cycle is formed when node k is both an "upstream node" and a "downstream node" of node i relative to the source h. Recall that every node stores the minimum forward count ($minfc_{ji}^h$) for every route request it receives, which is the smallest forward count of the route requests received. Now, if k is an upstream node of i, then it is nearer to h than i, hence $minfc_{jk}^h < minfc_{ji}^h$. If k is a downstream node of i, then $minfc_{jk}^h > minfc_{ji}^h$. From lines 16-21 of *process_rreq*, $minfc_{jk}^h$ can only have one value, that is the smallest of all forward counts. Suppose that k is originally located upstream of i and it moved fast so as to become a downstream of i. If this is the case, then $minfc_{jk}^h$ will not change because downstream, k will hear route requests with larger forward counts than $minfc_{jk}^h$. If the reverse happens, $minfc_{jk}^h$ might change and become smaller than $minfc_{ji}^h$, therefore still satisfying the conditions. Hence, there is no way that some node k can be both an upstream and downstream of some other node i to form loops. ■

Theorem 1: The routes formed by a route reply from a destination to a source are loop-free.

Proof: The proof is simple and follows consequently from Property 2. Since the route reply traverses the reverse path from the destination h to the source j, from Property 2, the routes formed are free from loops. ■

Theorem 2: In a connected network, any node that sent a route request will receive route replies in finite time.

Proof: Since the transmission of route requests is bounded (Property 1) and the paths traversed by route replies are loop-free (Theorem 1), then consequently, the source node will receive route replies after some finite time. ■

REFERENCES

[1] C. Perkins and P. Bhagwat, "Highly dynamic destination-sequenced distance-vector routing (dsdv) for mobile computers," in *Proc. ACM SIGCOMM '94*, Aug. 1994, pp. 234–244.

[2] M.S. Corson and A. Ephremides, "A distributed routing algorithm for mobile wireless networks," *ACM/Baltzer Wireless Networks J.*, vol. 1, no. 1, pp. 61–82, Feb. 1995.

[3] S. Murthy and J.J. Garcia-Luna-Aceves, "An efficient routing protocol for wireless networks," *Mobile Networks and Applications*, vol. 1, no. 2, pp. 183–197, 1996.

[4] R. Dube, C. Rais, K.Y. Wang, and S.K. Tripathi, "Signal stability-based adaptive routing (ssa) for ad hoc mobile networks," *IEEE Personal Commun.*, vol. 4, no. 1, pp. 36–45, Feb. 1997.

[5] D. Johnson and D. Maltz, "Dynamic source routing in ad hoc wireless networks," in *Mobile Computing*, T. Imielinski and H. Korth, Eds., pp. 153–181. Kluwer Academic Publishers, Norwell, Mass., 1996.

[6] Z. Haas, "A new routing protocol for the recongurable wireless networks," in *Proc. IEEE ICUPC '97*, Oct. 1997.

[7] C. Perkins and E. Royer, "Ad hoc on-demand distance vector routing," in *Proc. IEEE WMCSA '99*, Feb. 1999, pp. 90–100.

[8] G. Pei, M. Gerla, and T. Chen, "Fisheye state routing: A routing scheme for ad hoc wireless networks," in *Proc. IEEE ICC '00*, June 2000, pp. 70–74.

[9] P. Jacquet, P. Muhlethaler, A. Qayyum, A. Laouiti, L. Viennot, and T. Clausen, "Optimized link state routing protocol. draft-ietf-manet-olsr-04.txt," Work in Progress.

[10] J. Broch, D. Maltz, D. Johnson, Y.C. Hu, and J. Jetcheva, "A performance comparison of multi-hop wireless ad hoc network routing protocols," in *Proc. ACM/IEEE MOBICOM '98*, Oct. 1998, pp. 85–97.

[11] S.J. Lee, C.K. Toh, and M. Gerla, "Performance evaluation of table-driven and on-demand ad hoc routing protocols," in *Proc. IEEE PIMRC '99*, Sept. 1999, pp. 297–301.

[12] P. Johansson, T. Larsson, N. Hedman, B. Mielczarek, and M. Degermark, "Scenario-based performance analysis of routing protocols for mobile ad-hoc networks," in *Proc. ACM/IEEE MOBICOM '99*, 1999, p. 195206.

[13] IEEE Computer Society LAN MAN Standards Committee, "Wireless lan medium access protocol (mac) and physical layer (phy) specification, ieee std 802.11-1997," The Institute of Electrical and Electronics Engineers, New York, NY, 1997.

[14] M.V. Wilkes, "Slave memories and dynamic storage allocation," in *Trans. IEEE*, Apr. 1965, vol. EC-14, pp. 270–271.

[15] D. Patterson and J. Hennessy, *Computer Architecture, A Quantitative Approach*, Morgan Kaufmann, 1996.

[16] R. Castaneda and S. Das, "Query localization techniques for on-demand routing protocols in ad hoc networks," in *Proc. ACM/IEEE MOBICOM '99*, 1999, pp. 186–194.

[17] S. Das, C. Perkins, and E. Royer, "Performance comparison of two on-demand routing protocols for ad hoc networks," in *Proc. IEEE INFOCOM '00*, 2000, pp. 3–12.

[18] The VINT Project, "The network simulator - ns-2," Available at http://www.isi.edu/nsnam/ns/.

[19] The Monarch Project, "Rice monarch project: Mobile networking architectures," Available at http://www.monarch.cs.rice.edu/.

[20] A. Boukerche, "Performance comparison and analysis of ad hoc routing algorithms," in *IEEE ICPCC 2001*, 2001, pp. 171–178.

[21] R. Boppana and S. Konduru, "An adaptive distance vector routing algorithm for mobile, ad hoc networks," in *Proc. IEEE INFOCOM '01*, Mar. 2001.

[22] B. Tuch, "Development of wavelan, an ism band wireless lan," *AT&T Technical Journal*, vol. 72, no. 4, pp. 27–33, 1993.

[23] M. Pearlman, Z. Haas, P. Sholander, and S. Tabrizi, "On the impact of alternate path routing for load balancing in mobile ad hoc networks," in *Proc. ACM MOBIHOC '00*, 2000, pp. 3–10.

[24] S. Das, R. Castaneda, J.T. Yan, and R. Sengupta, "Comparative performance evaluation of routing protocols for mobile, ad hoc networks," in *Proc. IEEE ICCCN '98*, Oct. 1998.

[25] C. Perkins, E. Belding-Royer, and S. Das, "Ad hoc on-demand distance vector (aodv) routing. draft-ietf-manet-aodv-12.txt," Work in Progress.

[26] V. Milutinovic, "Caching in distributed systems," *IEEE Concurrency*, pp. 2–3, July-September 2000.

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

[28] E. Gafni and D. Bertsekas, "Distributed algorithms for generating loop-free routes in networks with frequently changing topology," *IEEE Trans. Commun.*, vol. 29, no. 1, pp. 11–15, January 1981.

[29] M.S. Corson and A. Ephremides, "A distributed routing algorithm for mobile radio networks," in *Proc. IEEE MILCOM '89*, Oct. 1989.

[30] S. Vutukury and J.J. Garcia-Luna-Aceves, "Mdva: a distance-vector multipath routing protocol," in *Proc. IEEE INFOCOM '01*, 2001, vol. 1, pp. 557–564.

[31] S. Lee and M. Gerla, "Aodv-br: Backup routing in ad hoc networks," in *Proc. IEEE WCNC '00*, Sept. 2000.

[32] M. Marina and S. Das, "On-demand multipath distance vector routing in ad hoc networks," in *Proc. IEEE ICNP '01*, 2001, pp. 14–23.

[33] D. Ganesan, R. Govindan, S. Shenker, and D. Estrin, "Highly-resilient, energy-efficient multipath routing in wireless sensor networks," in *Proc. ACM MOBIHOC '01*, Oct. 2001, pp. 251–253.

[34] A. Nasipuri and S. Das, "On-demand multipath routing for mobile ad hoc networks," in *Proc. IEEE ICCCN '99*, Oct. 1999, pp. 64–70.

[35] G. Holland and N. Vaidya, “Analysis of tcp performance over mobile ad hoc networks,” in *Proc. ACM/IEEE MOBICOM ’99*, Aug. 1999, pp. 219–230.

[36] R. Gallager, “A minimum delay routing algorithm using distributed computation,” *IEEE Trans. Commun.*, vol. 25, pp. 73–84, Jan. 1977.

[37] S. Vutukury and J.J. Garcia-Luna-Aceves, “A simple approximation to minimum-delay routing,” in *Proc. ACM SIGCOMM ’99*, 1999, pp. 227–238.

[38] R. Krishnan and J. Silvester, “Choice of allocation granularity in multipath source routing schemes,” in *Proc. IEEE INFOCOM ’93*, Mar. 1993, pp. 322–329.

[39] N. Gogate and S. Panwar, “Assigning customers to two parallel servers with resequencing,” *IEEE Trans. Commun. Lett.*, vol. 3, no. 4, pp. 119, Apr. 1999.

[40] N. Gogate and S. Panwar, “Supporting applications in a mobile multihop radio environment using route diversity,” in *Proc. IEEE ICC ’98*, June 1998.

[41] E. Blanton and M. Allman, “On making tcp more robust to packet reordering,” in *ACM SIGCOMM CCR*, Jan. 2002, vol. 32, pp. 20–30.

[42] S. Bohacek, J. Hespanha, J. Lee, C. Lim, and K. Obraczka, “Tcp-pr: Tcp for persistent packet reordering,” in *Proc. IEEE ICDCS 2003*, May 2003, pp. 222–.