Efficient On-Demand Routing Using Source-Tracing in Wireless Networks

Jyoti Raju
jyoti@cse.ucsc.edu
Computer Science Department
University of California
Santa Cruz, CA 95064
jyoti@cse.ucsc.edu

J.J. Garcia-Luna-Aceves
jj@cse.ucsc.edu
Computer Engineering Department
University of California
Santa Cruz, CA 95064
jj@cse.ucsc.edu

Abstract— **With on-demand routing, a router maintains routing information for only those destinations that need to be reached by the router. The approaches used to date to eliminate long-term or permanent loops in on-demand routing consist of obtaining complete routes to destinations dynamically, or obtaining only the next hops to destinations and validating the information using sequence numbers or internodal synchronization. We present a new approach to on-demand routing, which we call the DST (dynamic source tree) protocol. To eliminate looping, routers in DST communicate paths to destinations; however, only incremental updates to such paths are communicated by specifying the second-to-last hop and distance to each node in the subpath to the destination that must be updated. Simulations experiments are used to show that, in terms of control packet overhead, DST outperforms substantially the Dynamic Source Routing (DSR) protocol which is arguably one of the most efficient on-demand routing approaches to date, while achieving similar performance in terms of the average delay and throughput of data packets.**

I. INTRODUCTION

On-demand routing protocols have been designed to limit the amount of bandwidth consumed in maintaining up-to-date routes to all destinations in a network by maintaining routes to only those destinations to which the routers need to forward data traffic. The basic approach consists of allowing a router that does not know how to reach a destination to send a flood-search message to obtain the path information it needs. There are several recent examples of this approach (e.g., AODV [11], ABR [12], DSR [8], TORA [10], SSA [3], ZRP [7]) and the routing protocols differ on the specific mechanisms used to disseminate flood-search packets and their responses, cache the information heard from other nodes' searches, determine the cost of a link, and determine the existence of a neighbor. However, all the on-demand routing proposals use flood search messages that either: (a) give sources the entire paths to destinations, which are then used in source-routed data packets (e.g., DSR); or (b) provide only the distances and next hops to destinations, validating them with sequence numbers (e.g., AODV) or time stamps (e.g., TORA). One problem with source routing is that it results in long data-packet headers as the network size increases. On the other hand, protocols that use sequence numbers or timestamps incur additional overhead in resetting the sequence number and timestamps in the presence of partitions and node failures.

In this paper, we introduce and analyze the DST (dynamic source tree) protocol, which constitutes a new approach for on-demand distance vector routing in ad hoc networks. As in other on-demand routing algorithms, DST acquires routes to destinations only when traffic for those destinations exists and there is no correct route to the destination. The acquired route does not have to be the shortest path; it has to be valid and of finite metric value. DST does not use source-routed packets or time stamps to validate distance updates. DST uses a source-tracing algorithm similar to the one advocated in prior table-driven routing protocols in which routers maintain routing information for all network destinations [9]. Using information about the length and second-to-last hop ($predecessor$) of the shortest path to all known destinations, the source tracing algorithm reduces the number of loops and removes the counting to infinity problem of the distributed Bellman Ford algorithm.

This work was supported in part by the Defense Advanced Research Projects Agency (DARPA) under grant F30602-97-2-0338.

There are three key contributions of this paper: (i) introducing a new approach that uses a source-tracing algorithm for on-demand routing, (ii) presenting the design of a protocol that does not use sequence numbers or internodal synchronization to ensure correctness and (iii) showing through simulations that DST incurs less control overhead than DSR in most situations.

Section II gives a detailed description of DST and presents examples illustrating the working of the protocol. Section III uses simulations to compare DST's performance with the performance of DSR using the same simulation code and model used in [1] to compare DSR against other on-demand routing protocols.

II. THE DST PROTOCOL

A. Network Model

To describe DST, a network is modelled as an undirected graph with V nodes and E links. A router has a single node identifier, and a node has radio connectivity with multiple nodes through a single physical radio link. We map a physical broadcast link connecting a node and its multiple neighbors into point-to-point links between the node and its neighbors. Each link has a positive cost associated with it. If a link fails, its cost is set to infinity. A node failure is modelled as all links incident on the node getting set to infinity.

For the purpose of routing-table updating, a node A considers another node B as its neighbor if A receives an update from neighbor B. Node B is no longer node A's neighbor when the medium access protocol at node A sends a signal to DST indicating that data packets can no longer be sent successfully to node B.

Routing messages are broadcast unreliably and the protocol assumes that routing packets may be lost due to changes in link connectivity, fading or jamming. Since DST only requires a MAC indication that data packets can no longer be sent to a neighbor, the need for a link-layer protocol for monitoring link connectivity with neighbors or transmitting reliable updates is eliminated, thus reducing control overhead. If such a layer can be provided with no extra MAC overhead, then DST can be made more proactive by identifying lost neighbors before data for them arrives, resulting in faster convergence and decreased data packets losses.

B. Routing Information maintained in DST

A router in DST maintains a *routing table*, a *distance table*, a *data buffer* and a *query table*.

The routing table at router i contains entries for destinations needed by the router. Each entry consists of the destination identifier j, the successor to that destination s_j^i, the second-to-last-hop to the destination p_j^i, the distance to the destination D_j^i and a route tag tag_j^i. When the element tag_j^i is set to $correct$, it implies a loop-free finite value route. When it is set to $null$, it implies that the route still has to be checked and when it is set to $error$, an infinite metric route or a route with a loop is implied.

The distance table at router i is a matrix containing, for each $k \in N_i$ (where N_i is the list of known neighbors) and each destination j needed by such neighbor, the distance value of the route from i to j through k, D_{jk}^i and the second-to-last hop p_{jk}^i on that route. D_{jk}^i is always set

Report Documentation Page

Form Approved
OMB No. 0704-0188

Public reporting burden for the collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to a penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
2000		**00-00-2000 to 00-00-2000**

4. TITLE AND SUBTITLE	
Efficient On-Demand Routing Using Source Tracing in Wireless Networks	5a. CONTRACT NUMBER
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **University of California at Santa Cruz,Department of Computer Engineering,Santa Cruz,CA,95064**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**		**5**	

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std Z39-18

equal to $RD_j^k + l_k^i$, where RD_j^k is the distance reported by k to j in the last routing message and l_k^i is the link cost of link (i, k).

The data buffer is a queue that holds all the data packets waiting for routes to destinations. There are various schemes to do buffer management but we chose to use the scheme used by most existing on-demand routing protocols. Each data packet also has a time value, which is set to the time when the packet is put into the buffer. A packet that has been in the buffer for more than $data_pkt_timeout$ seconds is dropped. The data buffer is checked periodically for any packets that may be sent or dropped.

The query table is used to prevent queries from being forwarded indefinitely. As in DSR, there are two types of queries; queries with zero hop count which just get propagated to neighbors and queries with maximum hop count that are forwarded to a maximum distance of MAX_HOPS hops from the sender. For a destination j, the query table contains the last time a maximum hop query was sent qs_j^i, the last time a zero hop query was sent zqs_j^i, the hop count of the last query sent hqs_j^i, the last time a query was received qr_j^i. At the source of the flood search, two maximum hop count queries are always separated by $query_send_timeout$ seconds. A query is forwarded by a receiver only if the difference between the time it is received and qr_j^i is greater than $query_receive_timeout$, where $query_receive_timeout$ is slightly lesser than $query_send_timeout$.

C. Routing Information exchanged in DST

There are two types of control packets in DST - $queries$ and $updates$. All control packets headers have the source of the packet ($pkt.src$), the destination of the packet ($pkt.dst$), the number of hops ($pkt.hops$) and an identifier $pkt.type$ that can be set to $QUERY$ or $UPDATE$. Each packet has a list of routing entries, where each entry specifies a destination j, a distance to the destination RD_j^i and a predecessor to the destination rp_j^i.

If the MAC layer allowed for transmission of reliable updates with no retransmission overhead, only incremental routing updates could be sent. In this paper, however, we assume a MAC protocol based on collision avoidance, in which sending larger control packets does not decrease throughput at the MAC layer, because the overhead for the MAC protocol to acquire the channel does not depend on packet size [5], [6]. Therefore, in the rest of the paper, we assume that routers transmit their entire routing tables when they send control messages. Control packet size may affect the delay experienced by packets in the MAC layer. However, as our simulations show, this does not affect data packet delays because the number of control packets we generate is substantially low.

Data packets in DST only need to have the source and destination in the header, rather than the source routes as in DSR.

D. Creating Routes

When a network is brought up, each node (i) adds a route to itself into its routing table with a distance metric (D_i^i) of zero, the successor set equal to itself (i) and the tag (tag_i^i) set to correct. To differentiate a route to itself from all other routes, a node sets the local host address (127.0.0.1) as the predecessor to itself.

When a data packet is sent by an upper layer to the forwarding layer, the forwarding layer checks to see if it has a correct path to the destination. If it does not, then the packet is queued in the buffer and the router starts a route discovery by broadcasting queries. Route discovery cycles are separated by *query_receive_timeout* seconds. One zero hop query and one maximum hop query are sent in every cycle. A zero hop query allows the sender to query neighboring routing tables with one broadcast. If the zero hop query times out ((present time - $zqs_j^i) > zero_qry_send_timeout$), then an unlimited hop query (with $pkt.hops$ set to MAX_HOPS) is sent out. Consider the six-node network in Fig. 1.a where all link costs are of unit value and where node d broadcasts a query for destination a , with the $pkt.src$ set to d, $pkt.dst$ set to a, and $pkt.hops$ set to MAX_HOPS. The parenthesis next to each node in the example depicts the routing table entry (distance, predecessor) for destination a. The symbol lh stands for local host address (127.0.01). The query packet contains a list of all the routing table entries of the sender d. The entries are shown within the square brackets, each entry in the (destination, distance, predecessor) form. The entries are in a increasing distance order, such that a node i receiving a query from an unknown neighbor k, adds the neighbor k to its distance tables on reading the first entry in the query and proceeds to consider all other entries as the distances reported by k.

Consider the node e, where a query is received. To process the query, each entry (j, RD_j^d, rp_j^d) is read. If the entry is for an unknown destination, then the destination is initialized ($D_j^i \rightarrow \infty$, $p_j^i, s_j^i \rightarrow NULL_ADDR$; $D_{jk}^i \rightarrow \infty$, $p_{jk}^i \rightarrow NULL_ADDR$ $\forall k \in N_i$. Then, the distance table entry for neighbor d is updated. Since the distance RD_d^d is equal to zero, d is marked as a neighbor. The value for j reported by other neighbors whose path contains d is also updated. This step helps prevent permanent loops by preemptively removing stale information.

Finally, routing table entries are updated to pick as successor a neighbor k to destination j if both the following conditions are true

1. k offers the shortest distance to all nodes in the path from j to i.
2. the path from j to k does not contain i and does not contain any repeated nodes.

If either of the two conditions are not satisfied, then tag_j^i is set to $error$. Else, it is set to $correct$ and neighbor k is designated the successor and the distance value to j is set to D_{jk}^i and the predecessor is set to p_{jk}^i.

After processing all entries and updating the routing table, the node e checks to see if it has a route to a. Since there is no route, a query packet is broadcasted with the same header fields as the processed query, besides $pkt.hops$ which is decremented by one if (a.) a node does not have a route to $pkt.dst$, (b.) $pkt.hops$ is greater than one, and (c.) if the time elapsed since the last query was received is greater than $query_receive_timeout$

The routing entries added to the forwarded query reflect the routing table entries of current node e. The packet is then broadcasted to the limited broadcast address. In Fig. 1.b, nodes e, f and c broadcast queries.

In Fig. 1.c, we see that nodes e, f, a do not send any more queries because the time elapsed since the last query sent is lesser than $query_receive_timeout$. On the other hand, at nodes a and b, a finite and valid route to a is found and a reply update is sent. A reply update sent by a node i has a different structure than a regular update, which has $pkt.dst$ set to the limited broadcast address and $pkt.src$ set to i. The reply update sent by b has field $pkt.dst$ set to the $pkt.src = d$ of the query and the field $pkt.src$ set to the $pkt.dst = a$ of the query. All updates are broadcast to the limited broadcast address.

When node i receives an update, it checks the value of $pkt.dst$. If it is set to a value other than the limited broadcast address, then the update being sent is a reply update, else it is a regular update. The entries are processed in a manner similar to the entries of the query. A regular update is broadcast in response to a regular update, with $pkt.dst$ set to the limited broadcast address and $pkt.src$ set to i if (a.) the distance to a known destination increases, or (b.) if a node loses the last finite route to a destination. The reply update has different rules for propagation. In Fig 1.d , a reply update is rebroadcasted by e with the original $pkt.dst$ and $pkt.src$, because (a.) a finite path to $pkt.dst = d$ exists, and (b.) the distance to $pkt.src = a$ changes from infinite to finite after processing the reply update. Nodes a and b do not rebroadcast reply updates because the second condition is not satisfied. Node d gets a reply update from node e and will set its successor to node e after processing the entries in the query. Node d does not send any more reply updates. However, a regular update will be sent if any of the two conditions for

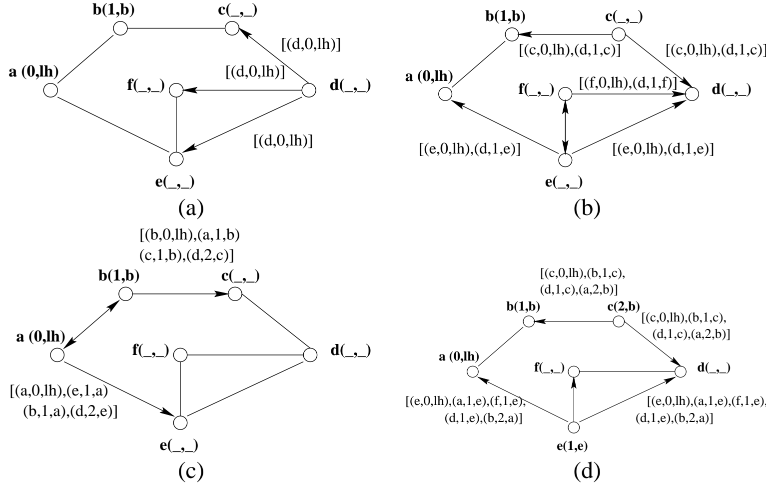

Fig. 1. Example of the Query-Reply process in DST. Node d is searching for destination a. The parenthesis contains the distance and predecessor values for a.

regular updates are satisfied.

Using the above procedure, DST allows a source to get multiple paths to a required destination. By forwarding a reply update only when the route to the required destination changes from infinite to finite, the number of updates is reduced at the expense of non-optimal routes. The same reasoning motivates not sending regular updates when a new destination is found or when a distance to a destination reduces. However, distance increases prompt updates because a loop can occur *only* when a node picks as successor a neighbor that has a distance greater than itself.

E. Maintaining Routes

DST does not poll neighbors constantly to figure out link connectivity changes, which avoids control overhead due to periodic update messages, but may result in sub-optimal routes and longer convergence time. A link to a neighbor is discovered only when an update or a query is received from that neighbor. On finding a new neighbor k the neighbor is added to the distance table. An infinite distance to all destinations through k is assumed, with the exception of node k itself and any destinations reported in the received routing message.

A failure of a link is detected when a lower-level protocol sends an indication that a data packet can no longer be sent to a neighbor. The neighbor is removed from the distance table and the routing table is updated.

DST provides two conditions to prevent data packets from looping. A data packet is dropped and a regular update is sent if (a.) the data packet is sent by a neighbor that is in the path from the present node to the destination of the data packet, or (b.) the path implied by the neighbor's distance table entry is different from the path implied in the routing table.

F. Packet Forwarding

The data packet header contains only the source and the destination of the data packet. When a data packet originated at a node arrives at its forwarding layer, the packet is buffered if there is no finite route to the destination. The node then starts the route discovery process. If a finite and correct route is found, then the packet is forwarded to the successor as specified by the routing table.

If a data packet is not originated at a node, then the data packet is only buffered if there is no entry in the routing table for $pkt.dst$. In this case, route discovery is started by the intermediate node. If there is a correct and finite route then the data packet is first checked for conditions a and b described in section II-E. If the two conditions are satisfied, the data packet is forwarded to the successor $s^i_{pkt.dst}$. If there is route with infinite distance, then the packet is dropped and a regular update is broadcast to all neighbors. Eventually, the source of the data will learn of the loss of routes and it will restart the route discovery process.

III. Performance Evaluation

We ran a number of simulation experiments to compare DST's average performance against the performance of DSR, which has been proved very efficient in earlier studies [1]. Both protocols are implemented in CPT, which is a C++ based toolkit that provides a wireless protocol stack and extensive features for accurately simulating the physical aspects of a wireless multi-hop network. The stack uses IP as the network protocol. The routing protocols uses UDP to transfer update packets. The MAC layer implements IEEE 802.11 standard based collision avoidance and the physical layer is based on a direct sequence spread spectrum radio with a link bandwidth of 1 Mbit/sec.

To run DSR in CPT, we ported the DSR code available in the $ns2$ [4] wireless release. There are two differences in our DSR implementation as compared to the implementation used in [1]. Firstly, we do not use the $promiscuous$ listening mode in DSR. We, however, implement the promiscuous learning of source routes from data packets. This follows the specification given in the Internet Draft of DSR. Our reason for not allowing promiscuous listening is that, besides introducing security problems, it cannot be supported in any IP stack where the routing protocol is in the application layer and the MAC protocol uses multiple channels to transmit data. Secondly, we do not reschedule packets that have already been scheduled over a link (for both protocols) since the routing protocol in our stack does not have access to the MAC and link queues. Tables I and II show the constants used in the implementation of DSR and DST, respectively.

TABLE I
Constants used in DSR simulation

Parameter	Value
Time between ROUTE REQUESTS (exponentially backed off)	500 msec
Size of source route header carrying carrying n addresses	$4n$+4(bytes)
Timeout for Ring 0 search	30 msec
Time to hold packets awaiting routes	30 sec
Max number of pending packets	50

TABLE II

CONSTANTS USED IN DST SIMULATION

Query send timeout	5 sec
Zero query send timeout	30 msec
Data packet timeout	30 sec
Max number of pending packets	50
Query receive timeout	4.5 sec
MAX_HOPS	17

A. Scenarios used in comparison

We compared DSR and DST using two types of scenarios. In both scenarios, we used the "random waypoint" model described in [1]. In this model, each node begins the simulation by remaining stationary for *pause time* seconds and then selects a random destination and moves to that destination at a speed of 20 m/s. Upon reaching the destination, the node pauses again for *pause time* seconds, selects another destination, and proceeds there as previously described, repeating this behavior for the duration of the simulation. We used the speed of 20m/s, which is the speed of a vehicle, because it has been used in simulations in earlier papers [1], [2] and thus provides a basis for comparison with other protocols. In both scenarios, we used a 50 node ad-hoc network, moving over a flat space of dimensions 7 X 6 miles (11.2 X 9.7 km) and initially randomly distributed with a density of approximately one node per square mile.

Two nodes can hear each other if the attenuation value of the link between them is such that packets can be exchanged with a probability p, where $p > 0$. Attenuation values are recalculated every time a node moves. Using our attenuation calculations, radios have a range of approximately 4 miles (135 db).

We have random data flows, where each flow is a peer-to-peer constant bit rate (CBR) flow and the data packet size is kept constant at 64 bytes. Data flows were started at times uniformly distributed between 20 and 120 seconds and they go on till the end of the simulation. The total load on the network is kept constant at 80 data packets per second (40.96 kbps) to reduce congestion. Our rationale for doing this is that increasing the packet rate of each data flow does not test the routing protocol. On the other hand, having flows with varying destinations does so. We also vary the pause times: 0, 30, 60, 120, 300, 600 and 900 seconds as done in [1].

In the first scenario, there are 20 CBR sources, each of which establishes a connection with a randomly picked destination.

In the second scenario, the number of sources is fixed at 10 sources with 60 flows and each source has peer-to-peer connections with 6 destinations. This scenario helps us evaluate the scalability of the approaches with respect to the number of outward flows each source has.

B. Metrics used

In comparing the two protocols, we use the following metrics:

- *Packet delivery ratio*: The ratio between the number of packets received by an application and the number of packets sent out by the corresponding peer application at the sender.
- *Control Packet Overhead*: The total number of routing packets sent out during the simulation. Each broadcast packet is counted as a single packet.
- *End to End Delay*: The delay a packet suffers from leaving the sender application to arriving at the receiver application. Since dropped packets are not considered, this metric should be taken in context with the metric of packet delivery ratio.

Packet delivery ratio gives us an idea about the effect of routing policy on the throughput that a network can support. It also is a reflection of the correctness of a protocol.

Control packet overhead has an effect on the congestion seen in the network and also helps evaluate the efficiency of a protocol. Low control packet overhead is desirable in low-bandwidth environments and environments where battery power is an issue.

Average end-to-end delay is not an adequate reflection of the delays suffered by data packets. A few data packets with high delays may skew results. Therefore, we plot the cumulative distribution function of the delays. This plot gives us a clear understanding of the delays suffered by the bulk of the data packets. Delay also has an effect on the throughput seen by reliable transport protocols like TCP.

C. Simulation results

C.1 Scenario 1

Scenario 1 is identical to the one presented in [1]. There are 20 CBR sources each of which picks a random destination to send traffic to.

Fig. 2 shows the control packet overhead for varying pause times. An obvious result is that the control packet overhead for both the protocols reduces as the pause time increases. DST is about 34 % better than DSR at pause time zero. At low rates of movement, DST is a clear winner with one tenth the control packet overhead of DSR. Clearly, the fact that the updates in DST contain the entire routing table, means that nodes running DST have a higher chance of knowing paths to destinations for whom no route discovery has been performed in the past. We are able to mimic the behavior of table-driven routing protocols in low topology change scenarios, in that we almost have information about the entire topology with very few flood searches.

As shown in Fig. 3, at lower pause times, DSR has the same packet delivery ratio as DST. However, as the pause time decreases, DSR suffers due to data packets getting dropped at the link layer, indicating that the routes provided in the source routes are not correct any more. At lower pause times, links get broken faster. Even though this results in higher control overhead, the routes obtained are relatively new. As mentioned earlier, we keep the load on the network constant. Since this load is divided among a large number of flows, we see very little congestion and therefore most packets get through at higher pause times during which the topology is close to static.

Fig. 4 shows the cumulative delay of the protocols. The graphs shown are logarithmic in time to accommodate the wide variation. Almost all packets are sent within 8 seconds in DST while some packets in DSR take almost 30 seconds. This is because a packet is allowed to stay in a buffer for a maximum of 30 seconds before it is dropped. These are packets that found the path just in time.

C.2 Scenario 2

Fig. 5 show the amount of control packet overhead each protocol incurs for varying pause times. In both protocols control packet overhead is a function of the workload and the changes in link connectivity. The control overhead of DSR is substantially higher than DST, almost 580% higher. Due to the nature of on-demand routing protocols, both protocols show higher overhead when there are flows to more destinations.

Fig. 6 shows the percentage of data packets received. This metric shows very similar behavior for both DSR and DST, rising rapidly after pause time 15. Since there is very little congestion due to control packets at the higher pause time, most of the data packets get through.

Fig. 7, show the delay behavior of the data packets and DSR performs marginally better.

IV. CONCLUSIONS

We presented DST, which is an on-demand routing algorithm based on a source tracing algorithm. DST does not use sequence numbers and therefore is not prone to inefficiencies in the presence of node failures; this leads us to suggest that our scheme of using only local timestamps to prevent indefinite route queries can be adopted by other one-demand

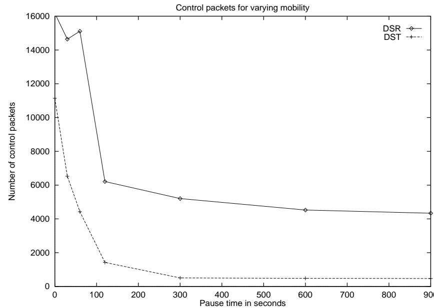

Fig. 2. Number of control packets sent for 20 sources picking random destinations for peer-to-peer flow

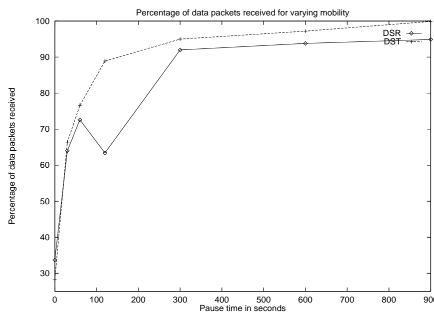

Fig. 3. Percentage of data packets received for 20 sources picking random destinations for peer-to-peer flow

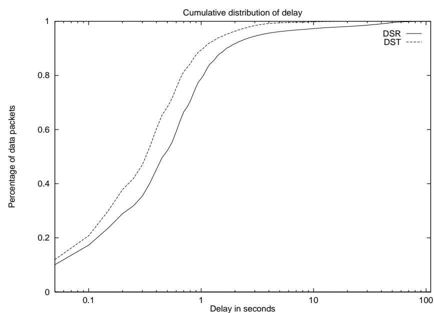

Fig. 4. Cumulative delay for pause time 0, 20 sources picking random destinations for peer-to-peer flow

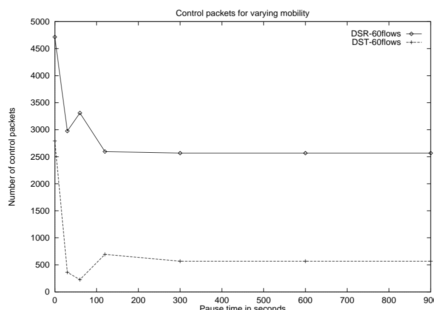

Fig. 5. Number of control packets sent for 60 flows - 10 sources, 6 destinations

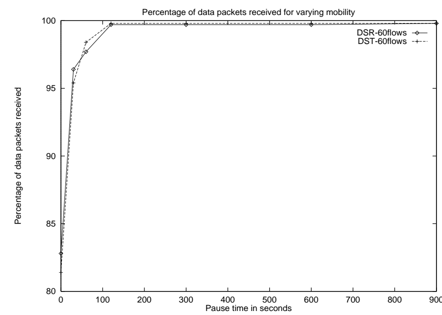

Fig. 6. Percentage of data packets received for 60 flows - 10 sources, 6 destinations

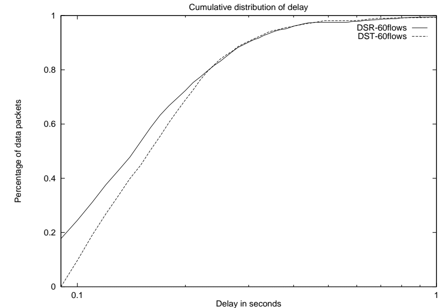

Fig. 7. Cumulative delay sent for 60 flows - 10 sources, 6 destinations (*pause time 0)*

routing protocols to prevent routing inefficiencies. DST also does not use reliable updates or polling of neighbors. This implies that DST creates substantially less overhead than protocols that use the above features. We introduce conditions that reduce control packet overhead at the expense of non-optimal routes, all the while preventing permanent looping of data packets.

Simulations were used to compare our protocol against DSR, which is arguably one of the most efficient on-demand routing protocols reported in literature. For all scenarios, DST performs consistently better than DST with respect to control overhead. The results for delay, hop count and percentage of data packets received are mixed, which leads us to believe that both protocols are at par when performance in these metrics is taken into consideration. Our simulations results show that DST is very suitable for ad-hoc networks and incurs limited control overhead, even in cases of high mobility.

REFERENCES

[1] J. Broch et. al. A Performance Comparison of Multi-Hop Wireless Ad Hoc Network Routing Protocols. In *Proc. ACM MOBICOM 98*, Dallas, TX, October 1998.

[2] Per Johansson et. al. Scenario Based Performance Analysis of Routing Protocols for Mobile Ad-Hoc Networks. In *Proc. ACM Mobicom'99*, Seattle, Washington, August 1999.

[3] R. Dube et. al. Signal Stability-Based Adaptive Routing (SSA) for Ad-Hoc Mobile Networks. *IEEE Pers. Commun.*, February 1997.

[4] Kevin Fall and Kannan Varadhan. *ns notes and documentation*. The VINT Project, UC Berkeley, LBL, USC/ISI and Xerox PARC, 1999. Available from http://www-mash.cs.berkeley.edu.

[5] C.L. Fullmer and J.J. Garcia-Luna-Aceves. Solutions to Hidden Terminal Problems in Wireless Networks. In *Proc. ACM SIGCOMM'97*, Cannes, France, September 1997.

[6] J.J. Garcia-Luna-Aceves and A. Tzamaloukas. Reversing The Collision-Avoidance Handshake in Wireless Networks. In *Proc. ACM/IEEE Mobicom'99*, Seattle, Washington, August 1999.

[7] Z. Haas and M. Pearlman. The Performance of Query Control Schemes for the Zone Routing Protocol. In *Proc. ACM SIGCOMM '98*, Vancouver, British Columbia, August 1998.

[8] D. B. Johnson and D. A. Maltz. Dynamic Source Routing in Ad-Hoc Wireless Networks. *Mobile Computing*, 1994.

[9] S. Murthy and J.J Garcia-Luna-Aceves. An Efficient Routing Protocol for Wireless Networks. *ACM Mobile Networks and Applications Journal*, 1996.

[10] V. D. Park and M. S. Corson. A Highly Adaptive Distributed Routing Algorithm for Mobile Wireless Networks. In *Proc. IEEE INFOCOM'97*, Kobe, Japan, April 1997.

[11] C. E. Perkins. Ad Hoc On-Demand Distance Vector (AODV) Routing. Internet Draft–Mobile Ad hoc NETworking (MANET) Working Group of the Internet Engineering Task Force (IETF). To be considered Work in Progress., November 1997.

[12] C.K. Toh. Associativity-Based Routing for Ad-Hoc Mobile Networks. *Wireless Personal Communications Journal, Special Issue on Mobile Networking and Computing Systems, Kluwer Academic Publishers*, 4(2):103–109, Mar. 1997.