GREP: Protocol and Proof of Loop-Free Operation

Henri Dubois-Ferrière Matthias Grossglauser Martin Vetterli
School of Computer and Communication Sciences
EPFL
1015 Lausanne, Switzerland
{henri.dubois-ferriere,matthias.grossglauser,martin.vetterli}@epfl.ch

***Abstract*—GREP (Generalized Route Establishment Protocol) is a routing protocol for ad hoc networks that is particularly well suited to very mobile environments. In GREP, packets are routed through space and time, meaning that at each hop they advance either along their current route (thus advancing in space), or onto a fresher route (advancing in time) to their destination. We give a full definition of the protocol and prove that it is loop-free.**

I. Introduction

A mobile ad hoc network is a collection of mobile nodes which dynamically form a network without using any fixed or pre-established infrastructure. These nodes collaboratively relay other nodes' communications over multi-hop routes; establishing and maintaining these routes is the task of a routing protocol which must react to changes in topology that occur as a result of node mobility. Rapid rate of topology change is a first characteristic of these networks; a second is that nodes indifferently act as end-hosts or routers, unlike in traditional wired networks where the separation between infrastructure (routers) and end-points (hosts) is more clearly delineated.

A large number of routing protocols for mobile ad hoc networks have been proposed. One important characteristic of any ad hoc routing protocol is whether it operates in euclidean space or over a logical graph. A *geographic* routing protocol assumes that nodes know their physical location (using for example a GPS receiver); in this case a node can make forwarding decisions based on the geographical positions of its neighbors. To compute a route in a geography-enabled network requires first determining the location of the destination through a *location service*, then computing a route towards that location. A *blind* routing protocol on the other hand does not require knowledge of nodes' physical positions and operates only over the connectivity graph.

Topology change resulting from mobility is usually viewed as a challenge. In previous work we have shown that mobility can also be helpful. Specifically, in [1], we showed that while node mobility creates uncertainty about the network topology, this mobility can also be exploited to disseminate topological information. We call this implicit dissemination of topological information the *mobility diffusion effect*. One approach taking advantage of mobility diffusion is through *last encounter routing (LER)* in geographic routing protocols: every node maintains a table recording the time and location of the last encounter with every other node. We showed that the last encounter (LE) information alone can provide good routes in geography-enabled networks and for certain classes of mobility models.

In practice, it is often impossible to obtain coordinates for every node in the network, either (i) because of cost or power considerations for positioning systems such as GPS, (ii) because a beaconing system is simply not available (e.g., indoor operation), or (iii) more fundamentally, because the network topology cannot be well embedded in a euclidean space (e.g., because there are many obstacles). Therefore, in this paper we consider the case of *blind* routing protocols that do not make reference to geographic positions.

The last encounter routing algorithms in [1] make explicit reference to both the *time* and the *location* of encounters, and therefore cannot be used in a blind network. In [2], we introduced a route discovery algorithm called FRESH for blind nodes, which makes use only of last encounter *times*. Intuitively, FRESH exploits the noisy age gradient around the destination formed by mobility diffusion to discover the destination through a sequence of localized floods. Our simulations have shown that with standard mobility processes, this can reduce the flooding over-

The work presented in this paper was supported (in part) by the National Competence Center in Research on Mobile Information and Communication Systems (NCCR-MICS), a center supported by the Swiss National Science Foundation under grant number 5005-67322

EPFL Technical Report IC/2003/40

head by an order of magnitude compared to protocols like AODV or DSR, which make a direct search.

The main point in the above studies is to demonstrate the value of last encounter information both in geographic and in blind protocols. Therefore, a common point of EASE and FRESH is that they make *exclusive use* of last encounter information disseminated through mobility diffusion, and do not exploit other sources of information on the current network topology, such as information diffused implicitly through traffic. In other words, the only topological information used by FRESH to discover nodes and establish routes is *temporal*, i.e., the age of the last encounter with the destination at every node. Traditional routing protocols use *spatial* information, acquired either implicitly through traffic (a packet sent from a node X to a node Y establishes a reverse route in the network from Y to X) or explicitly through route request floods, etc.

In this paper, our goal was to design a complete and practical routing protocol that exploits both temporal and spatial information, i.e., combining the advantages of last encounter (temporal) routing with a distance-vector (spatial) routing. The result is a routing algorithm that is particularly well suited to very dynamic environments with rapid topological change due to node mobility. In such an environment, spatial algorithms alone incur significant overhead, because routes fall apart very quickly, forcing frequent, costly rediscoveries. GREP, on the other hand, by combining temporal and spatial metrics, does much better. This is because routing state associated with a route that has been invalidated is still useful, and the age of the state is a measure of how useful. GREP is therefore able to establish a new route for a packet by combining "pieces" of outdated routes to the same destination, possibly repairing gaps through local route requests.

GREP does not make explicit reference to absolute time as a measure of age of a routing table entry, contrary to EASE and FRESH. All GREP really needs is to be able to compare the ages of entries for the same destination. This can be achieved by associating a sequence number with every routing table entry, which we can view as a virtual clock associated with the corresponding destination. Each node maintains its own clock and increments this clock with every interaction with the rest of the network (e.g., sending out a packet)[1].

We summarize the key advantages of GREP:

- GREP improves routing performance over existing ad hoc routing protocols and is particulary well suited for highly mobile scenarios.
- GREP integrates last encounter routing (LER) with traditional on demand distance vector routing.
- At each hop GREP advances along either the spatial or the temporal axis, meaning that it advances along its current route (moving in space) or onto a route which was more recently established (moving in time).
- Route repair and route establishment are carried out using the same procedure; in fact there is no distinction between the two.
- Routing happens on a hop-by-hop basis, meaning that all interactions (packet forwarding, route repair/establishment) are done locally, without involving the end-point nodes of the route. In fact, there is no notion of end-to-end route which must be established and converged before nodes can send packets.
- GREP allows a flexible and dynamic tradeoff between pure on-demand and pure proactive operation.

II. GREP: Generalized Route Establishment Protocol

In the previous section we have informally described the protocol. We now give a detailed and unambiguous formulation, which will be necessary for the proof of loop-freedom in Section III.

We describe GREP as a set of *routing table rules*, which collectively describe all interactions by which routing state is created or modified, and a set of *packet processing rules*, which collectively describe all packet control and forwarding actions. We should emphasize that these rules comprehensively specify *all packet processing and routing actions*, showing that local processing and data structures are extremely simple, in particular when considering the flexibility and generality offered by the protocol.

A. Assumptions

Our protocol specification given below makes some assumptions on the properties of the link layer employed. We outline these assumptions below, noting that they are not absolute requirements, but allow us to give a concise definition of GREP (for example avoiding the minutiae of network-layer acknowledgements).

We assume a link layer which supports both broadcasting or unicasting of packets. A broadcast packet is received

[1] Note that AODV also uses a per-node sequence number; however, it is used there only to ensure loop freedom.

and processed by all nodes within range. A unicast packet carries the address of the next hop it is intended for. In either case collisions or interference may prevent reception of a packet.

This presentation of GREP also assumes that link-layer acknowledgements are provided, meaning that the link layer informs the network layer when a unicast packet transmission has failed. As other protocols, GREP can use network-layer acknowledgements if link-layer acknowledgements are not available. The use of network-layer acknowledgements is not described here; it does not change the basic operation of GREP.

B. Routing Table Entries

Each node maintains a distance-vector routing table which contains one entry for each destination node. Each entry is a tuple of three fields: next hop, hopcount, and destination sequence number.

TABLE I
ROUTING TABLE ENTRIES

nh_D^N	Next hop along the path to node D as known at node N.
hc_D^N	Hopcount to node D as known at node N.
sn_D^N	Sequence number of node D as known at node N.

Table I summarizes the notation used to describe routing state at each node. We use the convention that if node N has no entry for D, $nh_D^N = null$, $hc_D^N = \infty$, and $sn_D^N = 0$.

Sequence numbers are 32 bits long. The sequence number space is circular: the next sequence number after 2^{32} is 0. Sequence number comparisons are also made in the circular space: given two sequence numbers a and b, we say that $a > b$ if $a \neq b$ and $|a - b| \mod 2^{31} = 0$.

C. Packet types

GREP employs four logical packet types: Data (DATA), Route request (RREQ), route reply (RREP), and route advertisement (RADV) packets. In all cases the GREP-specific information is carried in a thin header (8-12 bytes) and so all packet types (not only DATA packets) can be piggybacked with application data.

A node sends a *route request* packet when it does not have a route to the destination, or if the next hop along the route is broken. It sends a *route reply* packet in reply to a route request when it has a route which satisfies that request. Note that often the node answering the route request is an intermediate node which is nearer than the destination; in this case we say that this node answered *on behalf of* the destination. A *route advertisement* packet is sent in order to advertise reachability and to update remote nodes' routing tables with an up-to-date entry to the advertising node. Route advertisements are optional; a node can change frequency and scope of its route advertisements can change at any time. Finally a *data* packet is a regular packet carrying application data.

Data and route reply packets are unicast. Route request and route advertisement packets are flooded (using MAC-layer broadcasts); the scope of flooded packets is constrained by a time-to-live (TTL) hopcount. Route requests proceed as an expanding ring search: when a request times out with no answer, the source increases the TTL (until a maximum value is attained) and initiates a new request. If the maximum TTL is attained, subsequent requests use a binary exponential backoff since the destination might be unreachable.

Note that GREP does not employ any route error packets: since link breaks are always repaired locally, there is no need to inform the source and upstream nodes when this occurs.

Table II lists the different packet header fields as well as the notation employed to refer to them; for each field it specifies for which of the four packet types these fields are valid.

xxx/does the following paragraph help at all or should i rm it... what do you think??
We say that a packet p *advertises* node A when reception of this packet at a node I may cause I to update its routing entry for A. Under this notation, a packet p always advertises its source ($p.src$). Note that RADV packets are therefore not the only ones to advertise their source; however their *only purpose* is to advertise reachability. Also, every packet p advertises the node $p.l2src$ which has just transmitted it. Since $p.l2src$ changes at each hop, this advertising has only a one-hop scope. xxx/phrase better A route request packet advertises the node $p.dst$ which it is looking for. Finally a RREP packet advertises two nodes, the source of the packet and the on-behalf-of ($p.osrc$) source of the packet.

TABLE II

PACKET FIELDS

Label	Description	Packet Types
p.src	*Source node*: The node which originated this packet.	All
p.dst	*Destination node*: The intended last-hop recipient for a DATA or RREP packet. The node to which a route is requested for a RREQ packet.	DATA, RREP, RREQ
p.l2src	*Relay node*: The node which last transmitted this packet. Therefore relay node and source node are equal only at the first hop of a packet.	All
p.ssn	*Source sequence number*: The sequence number of the source when it originated the packet.	All
p.shc	*Source hopcount*: The number of hops this packet has traversed since leaving the source.	All
p.ttl	*Time-to-live*: The number of hops this packet can traverse before being dropped.	RADV, RREQ
p.dsn	*Destination sequence number*: Last known sequence number for the destination to which a route is requested.	RREQ
p.dhc	*Destination hopcount*: The number of hops for requested destination as known to the source when it originated the route request.	RREQ
p.osrc	*On-behalf-of source*: The node on behalf of which a route reply is sent.	RREP
p.ohc	*On-behalf-of hopcount*: The hopcount in the answering node's table for the on-behalf-of source	RREP
p.osn	*On-behalf-of sequence number*: The sequence number in the answering node's table for the on-behalf-of source	RREP

D. Routing Table Rules

We now define the rules for updating routing tables. In GREP, routing state can only be modified as a consequence of sending or receiving a packet. Rule 1 (RTR1) is applied every time a node originates or forwards a packet, and simply increments that node's sequence number. Rules 2, 3, and 4 are applied when a node receives a packet and update the receiving node's routing table, if it is possible to do so without violating protocol invariants.

RTR 1: **Incrementing own sequence number.**
A node N increments its sequence number before sending any packet (whether originated or forwarded by N):

$$sn_N^N \leftarrow sn_N^N + 1$$

RTR 2: **Updating an entry to the source node.**
A node N receiving any packet p from relay node T verifies if either of the following conditions holds:

$$p.ssn > sn_{p.src}^N$$
$$(p.ssn = sn_{p.src}^N \wedge p.shc < hc_{p.src}^N)$$

If either of the two conditions holds, N updates (or creates) its routing entry for the source of the packet:

$$sn_{p.src}^N \leftarrow p.ssn$$
$$hc_{p.src}^N \leftarrow p.shc \qquad nh_{p.src}^N \leftarrow T.$$

RTR 3: **Updating an entry to the on-behalf-of node.**
A node N receiving a RREP packet p from relay node T verifies if either of the following conditions holds:

$$p.osn > sn_{p.osrc}^N$$
$$(p.osn = sn_{p.osrc}^N \wedge p.ohc \leq hc_{p.osrc}^N)$$

If either of the two conditions holds, N updates (or creates) an entry for the on-behalf-of source of the packet:

$$sn_{p.osrc}^N \leftarrow p.osn$$
$$hc_{p.osrc}^N \leftarrow p.ohc + p.shc \qquad nh_{p.osrc}^N \leftarrow T.$$

RTR 4: **Updating an entry to the destination node.**
A node N receiving a RREQ packet p from relay node T verifies if either of the following conditions holds:

$$p.dsn > sn_{p.dst}^N$$
$$(p.dsn = sn_{p.dst}^N \wedge p.dhc + p.shc < hc_{p.dst}^N)$$

If either of the two conditions holds, N updates (or creates) an entry for the destination of the route request packet:

$$sn_{p.dst}^N \leftarrow p.dsn$$
$$hc_{p.dst}^N \leftarrow p.shc + p.dhc \qquad nh_{p.dst}^N \leftarrow T.$$

RTR 5: **Updating an entry to the relay node.**
A node N receiving a packet p updates (or creates) a one-hop entry to the relay T node of the packet, and increments its sequence number for that node:

$$sn_T^N \leftarrow sn_T^N + 1$$
$$hc_T^N \leftarrow 1 \qquad nh_T^N \leftarrow T.$$

E. Packet Processing Rules

We now describe the rules for creating and forwarding packets.

Rule 1 (PPR 1) ensures that duplicate and out-of-order flood packets (RADV, RREQ) are discarded. Rule 2 concerns unicast forwarding of DATA packets. All unicast forwarding happens identically whether the packet is generated locally or has arrived from a neighbor node. Rules 3 and 4 concerns originating and forwarding of RREQ packets, as well as originating of RREP packets in answer to an incoming RREQ. RREP forwarding, defined in Rule 5, is identical to DATA forwarding except that the packet is discarded if forwarding fails. Finally Rule 6 concerns the forwarding of DATA packets which were buffered pending a route request.

PPR 1: **Discarding duplicate and out-of-order flood packets.**
A node N receiving a RADV or RREQ packet p verifies if either of the following conditions holds:

$$p.ssn < sn^N_{p.src}$$
$$(p.ssn = sn^N_{p.src} \wedge p.shc > hc^N_{p.src})$$

If either condition holds, N discards the packet without any further processing actions.

PPR 2: **Originating DATA packets.**
A node N originating a DATA packet p for destination D initializes header fields as:

$$p.src \leftarrow N \qquad p.dst \leftarrow D$$
$$p.shc \leftarrow 1 \qquad p.ssn \leftarrow sn^N_N$$

and then forwards the packet according to PPR 5.

PPR 3: **Originating RREQ packets.**
A node N originating a route request to node D initializes the RREQ packet q as follows:

$$q.src \leftarrow N \qquad q.dst \leftarrow D$$
$$p.ssn \leftarrow sn^N_N \qquad q.shc \leftarrow 0$$
$$q.dsn \leftarrow sn^N_D \qquad q.dhc \leftarrow hc^N_D$$

If this is the initial route request (meaning that it was triggered by a failure in forwarding a DATA packet to node D) the time-to-live is set to TTL_START. If this is a subsequent route request (meaning that it was triggered by a timer expiry for a previous route request to D) then the time-to-live is set to xxx (see if expo or additive like in AODV).

PPR 4: **Forwarding RREQ packets and Originating RREP packets.**

A Node N receiving a RREQ packet p verifies if either of the following two conditions holds:.

$$sn^N_{p.dst} > p.dsn$$
$$(sn^N_{p.dst} = p.dsn \wedge hc^N_{p.dst} + p.shc \leq p.dhc)$$

If either condition holds, N initiates a route reply by unicasting a RREP packet q which is initialized as:

$$q.src \leftarrow N \qquad q.dst \leftarrow p.src$$
$$q.ssn \leftarrow sn^N_N \qquad q.shc \leftarrow 0$$
$$q.osrc \leftarrow p.dst$$
$$q.osn \leftarrow sn^N_{p.dst} \qquad q.ohc \leftarrow hc^N_{p.dst}$$

Otherwise, N decrements $p.ttl$, increments $p.shc$, and rebroadcasts the RREQ packet if the TTL is still positive.

PPR 5: **Forwarding DATA and RREP packets.**
A node N receiving a unicast (DATA, RREP) packet p not destined for itself increments the source hopcount ($p.shc \leftarrow p.shc + 1$) and forwards the packet to $nh^N_{p.dst}$.

If forwarding fails (or if $nh^N_{p.dst} = null$) and p is a RREP packet then N discards the packet with no further actions. If forwarding fails (or if $nh^N_{p.dst} = null$) and p is a DATA packet then N buffers packet p for later retransmission and initiates a route request procedure for $p.dst$ according to PPR 3. If N's packet buffer was full, N drops the packet without initiating a route request.

PPR 6: **Forwarding buffered DATA packets.**
Each time a node N updates an routing table entry to a node D, N checks to see if it has any buffered packets for D. If so, N cancels any timer for pending route requests for D, and forwards the buffered packets according to PPR 5.

III. Analysis

In this section we prove that GREP is free of routing loops.

We distinguish between *packet loops* and *route loops*. A packet loop happens when a unicast packet traverses the same node twice. A route loop happens when a unicast packet traverses the same node twice, and the routing state pertaining to the packet's destination at that node does not change between both traversals. A route loop is potentially infinite (unless some mechanism is used to kill packets which have traversed more than some number of hops). In other words a packet gets "stuck" in a route loop but not in a packet loop; since the routing state changes when it traverses the same node for the second time.

We say that a protocol is *route loop-free* when it guarantees that route loops cannot happen. A stronger condition is for a protocol to be *packet loop-free*, meaning that packet loops cannot happen.

We first state some trivial invariants on routing tables maintained under GREP.

Invariant 1: No routing entry for a node can have sequence number greater than the actual sequence number of that node:

$$\forall j \forall k, sn_k^j \leq sn_k^k$$

Proof. The result is trivial when $j = k$. The key observation is that the sequence number for a node k can only be initialized by other nodes (directly in RTR 2, indirectly in RTR 3, 4) as a result of k setting the $p.ssn$ field of a packet to its own sequence number. Once a node j has a sequence number for k, this sequence number may be modified again by (RTR 2, 3, 4), which again comes directly or indirectly from k. Or it may be modified by RTR 5. Since RTR 5 is only applied after overhearing a packet transmitted by k, and since node k has incremented its own sequence number (RTR 1), we will continue to have that $sn_k^j \leq sn_k^k$.

Invariant 2: Routing table entries have strictly positive hopcounts.

$$\forall j \forall k, \neq j, hc_j^k > 0$$

Proof. This follows from the observation that hopcount fields in packets can only be incremented by nodes forwarding these packets, and that the source hopcount is always set to 1 by the source node of a packet.

We can now state the invariant property underlying GREP's loop-free operation; informally that at each hop, a packet moves closer to its destination either in the sequence number space or in the hopcount space.

Invariant 3: Unicast (DATA, RREP) packets are forwarded at each hop to a node with a higher sequence number for the destination or with the same sequence number and a lower hopcount.

Proof. Consider a packet p with destination D which node i forwards to node j at time t.

Under this notation we must show that either of the following two conditions is satisfied:

$$sn_D^j > sn_D^i \quad (1)$$

$$sn_D^j = sn_D^i \wedge hc_D^j < hc_D^i \quad (2)$$

If $j = D$ the result is trivial from invariants 1 and 2 which state that $sn_D^D \geq sn_D^i$ and $hc_D^i = 1 > 0 = hc_D^D$.

Now consider the case $j \neq$D. Since i forwarded packet p to j, we have that $nh_D^i = j$, which can only have happened as a result of i having previously received from j a packet q such that either $q.src = D$ (RTR 2), $q.osrc = D$ (RTR 3), or $q.dst = D$ (when q is a RREQ packet, RTR 2).

In either case, the key observation is that, since this packet was forwarded to i by j, j has 'seen' the routing information pertaining to destination D before i has seen this information.

If the packet q was a flood (RREQ or RADV) packet, then the conditions in PPR 1 did not hold at j, since j did not discard the packet. If the conditions in PPR 1 did not hold, then the conditions in RTR 2 did hold at j, meaning that j has applied RTR 2 and has an entry for D. Let us now assume that $sn_j^D < sn_i^D$. This is immediately contradicted by the fact that j has forwarded the packet q; therefore, j has updated its sequence number for D from the same packet q as i has. therefore we have that $sn_j^D \geq sn_i^D$.

We now have two cases to consider. If $sn_D^i = sn_D^j$, then $hc_D^i = hc_D^j + 1 < hc_D^j$ since j incremented $q.shc$ before sending the packet, in which case condition (2) is satisfied. The other case is when $sn_D^j > sn_D^i$, which satisifies condition (1), and may happen if j received, after packet q, some other packet with a higher sequence number for D which it did not forward to i (for example if this was a DATA packet for j, or if it was a RADV packet which j did not forward because it had a TTL of 1).

We can now show that GREP is route loop-free:

Property 1: A packet routed by GREP can never enter a route loop.

Proof. Let us assume that a packet p is in a route loop, that is that it traverses node N twice with $sn_{p.dst}^N$, $nh_{p.dst}^N$, and $hc_{p.dst}^N$ having identical values at both times. This is immediately contradicted by invariant 3 since at each hop the packet advances monotonically in the sequence number and hopcount space.

IV. Discussion

We have shown above that GREP routes are loop-free. Our analysis has made the distinction between *packet loops* and *route loops*, the latter being what are commonly referred to as "routing loops", the absence of which must

be guaranteed for a protocol to be viable in practice. Indeed, routing loops can potentially be infinite and may cause packets to remain in the network for an indefinite amount of time (until, for example a TTL-based mechanism indicates that the loopy packet should be dropped).

We have shown in Section III that GREP is free of route loops. However, GREP does not guarantee the absence of packet loops. Therefore GREP offers a relaxed guarantee compared to those protocols which establish routes on an end-to-end basis and require a route to be converged before sending packets. This relaxed guarantee on worst-case behavior of the protocol can be seen as a consequence of GREP's hop-by-hop approach which is allows GREP in the average case to outperform end-to-end routing protocols. As reported in [3], despite the possibility for occasional packet loops, GREP outperforms an end-to-end routing approach, and therefore relaxing protocol guarantees to allow packet loops allows an increase in efficiency which make this worthwhile.

We discuss a small example of a packet loop showing that even when a packet loop does occur, subsequent packets will shortcut the loop and therefore packet loops cannot happen on back-to-back packets. In Fig. 1, there is a route from S to D, which might have been established by a packet sent earlier from D to S with sequence number 1. D has since moved and therefore the last hop of this route is broken. E also currently has a one-hop route to D, which might have been established by a packet transmitted earlier by D to E with sequence number 2.

We now consider a packet originated at S for destination D. This packet will arrive at node C, where forwarding to D will fail.

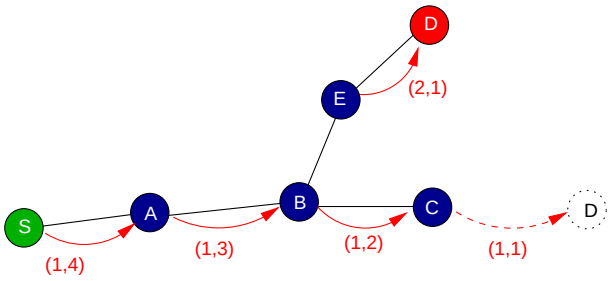

Fig. 1. A network with a route from S to D. D has moved, breaking the last hop. Routing table entries are shown for destination D as (sequence number, hopcount) associated with the corresponding link.

Since forwarding the DATA packet to D has failed, node C buffers this packet and initiates a route request for D. The route request is answered by E, since it has a route to D with a higher sequence number than 1. When C receives the route reply from E, it updates its routing entry sends the buffered packet.

This is an instance of a packet loop since the packet has traversed node B twice. Note that this is not a *route loop* since B's routing entry for destination D has changed between the first and second traversals, and therefore the packet does not get "stuck" in a loop between B and C. Note also that subsequent packets for D will now be forwarded by B to E; *each instance of a packet loop can only occur once*.

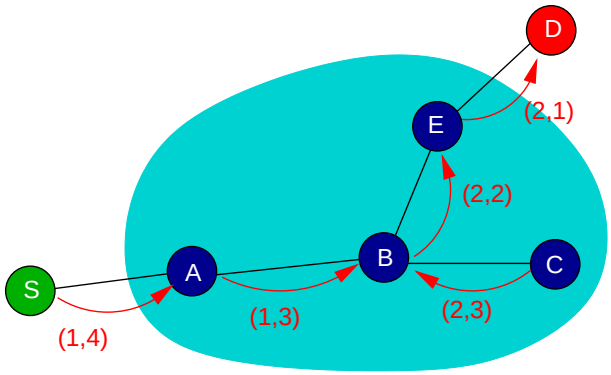

Fig. 2. C has made a route request for D, which was answered by E (the set of nodes reached by the route request with TTL=2 is shown in cyan). The packet for D buffered at C can now be transmitted to B, resulting in a packet loop. This packet loop will only occur for one packet; subsequent packets will be routed by B directly to E.

V. Conclusions

In this paper we have introduced GREP, a protocol that routes packets hop-by-hop in a distance-vector framework using a spatio-temporal distance metric. Packets are routed through space and time, meaning that at each hop they advance either along their current route (thus advancing in space), or onto a fresher route (advancing in time).

We have given a precise and unambiguous definition of the protocol which specifies all packet processing and routing actions. Using this defintion, we have shown that routing loops cannot occur under GREP. Our analysis has distinguished between route loops, where a packet gets "stuck" in a circular route, and the milder notion of packet loops, where a packet traverses a same node twice. Unlike many end-to-end routing protocols which guarantee that both route and packet loops are impossible, GREP has relaxed this constraint and does not impose packet loop-freedom, allowing for a more efficient hop-by-hop routing approach.

REFERENCES

[1] Matthias Grossglauser and Martin Vetterli, "Locating nodes with ease: Last encounter routing for ad hoc networks through mobility diffusion," in *Proceedings of the IEEE Conference on Computer Communications (INFOCOM)*, San Francisco, CA, April 2003.

[2] Henri Dubois-Ferriere, Matthias Grossglauser, and Martin Vetterli, "Age matters: Efficient route discovery in mobile ad hoc networks using last encounter ages," in *Proceedings of The Fourth ACM International Symposium on Mobile Ad Hoc Networking and Computing*, Annapolis, MD, June 2003.

[3] Henri Dubois-Ferriere, Matthias Grossglauser, and Martin Vetterli, "Grep: Space-time routing for highly mobile ad hoc networks," in *Submitted for Publication*, June 2003.