Group Source Routing Protocol with Selective Forwarding for Mobile Ad Hoc Networks

Hoon Oh[1] and Do Minh Ngoc[2]

Abstract. In this paper, we have proposed a group dynamic source routing protocol, GDSR, for mobile ad hoc networks with high mobility. We focus on pursuing routing stability and making a fast recovery of link failure. Nodes in a network are divided into clusters, each being assigned a unique cluster label. A routing path is represented by a source route including a sequence of cluster labels, and the nodes having an identical cluster label are responsible for delivering packets cooperatively to the cluster whose label is the next one in the source route. We have also employed a distributed self-pruning algorithm DSP to prevent intermediate nodes from relaying RREQ unnecessarily, thus reducing considerable overhead. We compared our protocol with some existing ones and the result is proven to be highly dependable.

1 Introduction

In recent years, there has been an increasing interest in Mobile Ad Hoc Networks (MANETs). Many routing protocols have been proposed for MANETs. Basically, these protocols fall among three kinds based on their mode of operation: proactive, reactive and hybrid routing.

Proactive routing protocols such as DSDV [6] and OLSR [4] maintain fresh lists of destinations and their routes by distributing routing tables in the network periodically. Therefore, a source host can get routing path immediately if it needs one. The main disadvantages of such algorithms are bandwidth wastage in transmitting routing table and useless overhead in maintaining routes that are never going to be used in future.

Reactive routing protocols find routing path on demand by initiating a route discovery. That is, routing paths are searched only when needed. Reactive protocols have less control overhead and better scalability than proactive routing protocols,

[1] He is an assistant professor at the Department of Computer Engineering and Information Technology, University of Ulsan, Korea (hoonoh@ulsan.ac.kr).

[2] He is a graduate student at the same department (ngocdm@mail.ulsan.ac.kr).

but suffer from long delay due to route searching. DSR [2] and AODV [7] are examples of reactive routing protocols.

An approach providing a better trade-off between proactive routing and reactive routing is hybrid routing protocols that are designed to increase scalability by allowing nodes to work together as a zone or cluster. These protocols proactively maintain routes to nearby nodes while determining routes to distant nodes using on-demand route discovery strategy. Some protocols belonging to this kind of routing such as CBRP [3] and ZRP [1] have been proposed during the recently years.

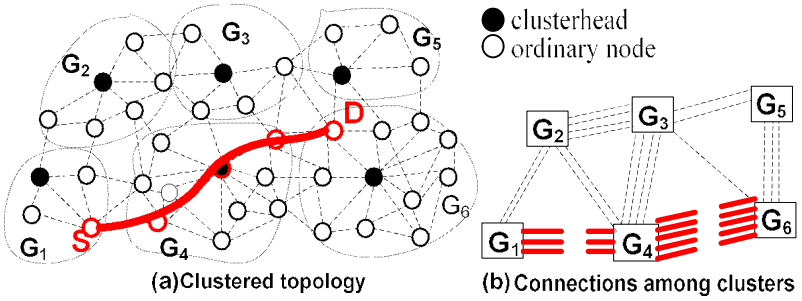

Fig. 1 GDSR's routing mechanism.

In this paper, we investigate a group dynamic source routing protocol (shortly GDSR) in which a group of nodes cooperates to deliver packets from one cluster to its neighbor cluster in the clustered network topology. In order to form and maintain the clustering of networked nodes, we employ the clustering algorithm proposed in our previous work [5]. Although DSR has low control overhead, its performance is not high in unstable networks because a source route including a sequences of node addresses represents only one path and becomes invalid when the path is broken. GDSR overcomes this shortcoming by dividing the network into clusters, each being assigned a unique cluster label. In GDSR, a source route representing routing path consists of a sequence of cluster labels instead of a list of node addresses as in DSR. Because of a high intra-cluster and inter-cluster connectivity in mesh networks, a source route may represent multiple paths from the source node to the destination and be still valid even when one path is broken, thus enabling a quick route recovery. Let us take a look at Fig. 1-(a) where the network is grouped into six clusters labeled with G_1, G_2, …, G_6. Suppose that *S* in G_1 wants to communicate with *D* in G_6. Then, the source route becomes (G_1, G_4, G_6) that represents 15 paths (3 x 5). The abstract representation of multiple paths in this network is given in Fig. 1-(b). The nodes in the G_1 cooperate to deliver packets to the G_2 that in turn does to G_6. Thus, note that if links between a pair of neighbor clusters are not completely broken, there always exist alternative paths. GDSR can therefore provide the excellent route stability and the high delivery ratio compared with DSR, especially in high mobility networks.

Moreover, we employ and modify a distributed self-pruning algorithm, DSP, from our previous work [8] by which only selective intermediate nodes forward route request message. About 50% to 70% of nodes are blocked from forwarding the route request message, which reduces considerably overhead as well as collision.
The rest of this paper is organized as follows. Section 2 assesses some related works. Section 3 and Section 4 demonstrate the employed clustering algorithm with some improvement and our proposed routing protocol, respectively. Section 5 presents the DSP algorithm, and is followed by performance evaluation in Section 6. Finally, we summarize our contributions in Section 7.

2 Related Works

In this section, we briefly overview some previous protocols closely related to our approach for their key concepts and weak points.
In the DSR protocol [2], when a node wants to send a packet, it initiates a route discovery by flooding a route request (RREQ). Upon receiving a route request, a node checks if it knows a route to the destination or itself is the destination. In both cases, the complete route (represented as a sequence of nodes) from the source node to the destination is replied to the source node by using RREP. Otherwise, the node appends its address and rebroadcasts the request if it has not received the same request before. The source node attaches the found source route in the packet. All the intermediate nodes forward the packet to their next hop based on the source route. If a node in the route finds out that it can not forward the packets to the next hop, it immediately sends a route error to the source node. The source node therefore is able to quickly detect an invalid route and stop using it any longer. Since DSR does not use any periodic control message like HELLO, it has a low overhead. However, the DSR protocol is not dependable in high speed networks since a source route representing only one path becomes invalid when the path is broken. Furthermore, the protocol's performance largely depends on the cached routes that are determined to be fresh. However, in high speed networks, intermediate nodes might reply with a staled route frequently that was determined to be fresh.
AODV [7] shares the same on-demand characteristics as DSR but owns a different mechanism to maintain routing information. Each node maintains a routing table. Each entry in the table corresponding to a destination records next hop and the number of hops to reach to the destination. AODV discovers a route through broadcasting RREQ. When an intermediate node forwards RREQ, it records in its table the previous node from which the RREQ came to construct reverse path for RREP. A RREP sent by the destination contains the total hop count of the route. As the RREP travels back, each intermediate node sets up the forward link as a route entry. In AODV, HELLO messages can be optionally used to discover neighbors. An invalid link can be detected through MAC layer or by periodic HELLOs. Whenever a link in use is no longer valid, the upstream node of that link

immediately notifies active neighbors of the link which, in turn, notify their active neighbors for the route and so on until the source nodes using that link are reached. AODV works efficiently in the networks that requires low end to end delay but it does not has good performance as well as generate high routing overhead.

CBRP [3] is a routing protocol in clustered ad hoc networks. Every node maintains 2-hop topology information and a cluster adjacency table that stores the addresses of the neighboring cluster heads and the gateway through which the corresponding cluster head can be reached, by periodically broadcasting HELLO message. HELLO contains the sender's neighbors and cluster adjacency table. A node that needs a route to a destination broadcasts a RREQ to its cluster head. Subsequently, the request is flooded to the neighboring cluster heads through the gateway nodes, and so on until it reaches the cluster head of the destination which forwards the request to the destination. The RREQ only records the cluster heads it has passed. The actual route can be shortened during the returning of the route reply. A node can do local route repair based on 2-hop topology. However, CBRP suffers from three things. HELLO message size is relatively big. A cluster head broadcasts RREQ and for each neighbor cluster head, only one appointed gateway is allowed to forward it: If the gateway moves away, the RREQ delivery to the neighboring cluster head fails. Lastly, recovery for a broken link is made only when there does exist an alternative 2-hop path.

3 Backbone Constructions

In our previous work [5], nodes are grouped into clusters based on the lowest-ID, each being assigned a unique label, simply called *cluster label*. Cluster labels themselves create the network backbone. Fig. 2 shows an example topology including four clusters with labels 0.1, 1.1, 4.1 and 9.1.

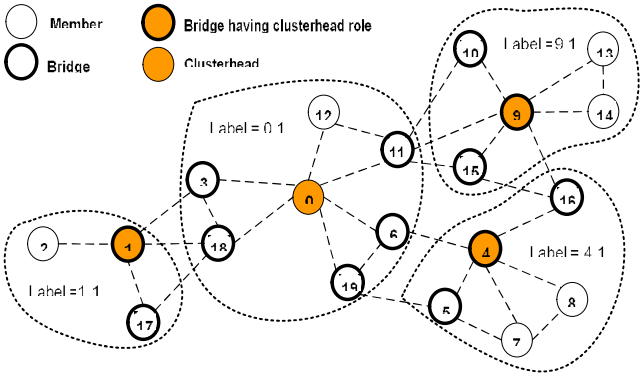

Fig. 2 The lowest-ID based clustering with labeling.

Every node periodically broadcasts HELLO message that contains the sender's address, state, cluster label and some information described later. It is assumed that if HELLO messages from a neighbor are missing for twice the hello-interval, the corresponding link is broken. A node is in either cluster head or member state if it joins in a cluster. Otherwise, it is in the orphan state. A node that has at least one neighbor belonging to another cluster is called bridge. A cluster label created by a cluster head includes the cluster head ID when the cluster is formed and the counter number that is initially set to one, two numbers being separated with a dot. If a cluster head disconnects from all the bridges that connect to one of its neighboring clusters, and finds a member in the same cluster to replace its role, it can form a new cluster as a cluster head. In this case, the counter number of the new label increases by one. More details about the clustering algorithm are presented in [5].

We modify the backbone maintenance mechanism in [5] to achieve the better cluster stability. A cluster head, when detecting a disconnection to its neighboring cluster, tries to find a member in the same cluster to replace its role based on criteria in a prioritized order as follows:

1. Select a node that provides as many connections to neighbor clusters as possible. Of course, the selected node should have more connections to neighboring clusters than the current cluster head.
2. Select a node that would have as many cluster members as possible if it were selected.

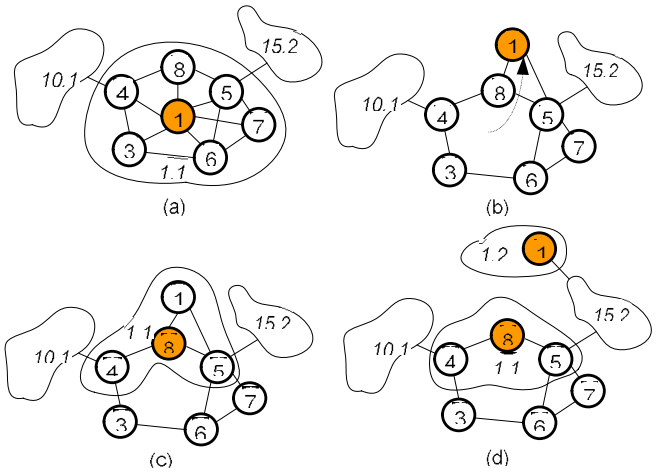

Fig. 3 An example of a new cluster head selection.

Consider an example in Fig. 3. Fig. 3(a) illustrates the network at the beginning. When node 1 moves away as shown in Fig. 3(b) and detects a complete disconnection from the only bridge 4 connecting to its neighboring cluster with label

10.1, it broadcasts an UpdateState message. Among the receiving nodes 5 and 8, node 8 is selected to replace the role of node 1 according to criterion 1 (such a choice enables routes from 10.1 to 15.2 and vice versus). Then, node 1 becomes a member immediately upon receiving UpdateState message from the new cluster head 8 as shown in Fig. 3(c). If node 1 continues to move away as shown in Fig. 3(d), it does not belong to cluster 1.1 and forms a new cluster, and its cluster label becomes 1.2, but not 1.1.

4 GDSR Algorithm

To route packets, every node maintains its *Connecting Cluster List* (CCL) that lists the cluster labels of the clusters it bridges, and *Bridge List* (BL) that consists of all neighboring labels and for each neighboring label, a set of bridges that directly connect to the cluster having the label. HELLO messages need to contain the sender's CCL. Neighbors of a bridge receive HELLO messages from the bridge and update their BL using CCL in the received HELLO message. For example, in Fig. 2, there are bridges with CCL as follows: $CCL_0 = \{\}$, $CCL_1 = \{0.1\}$, $CCL_3 = \{1.1\}$, $CCL_4 = \{0.1\}$, $CLL_5 = \{0.1\}$, $CCL_6 = \{4.1\}$, $CCL_9 = \{0.1, 4.1\}$ and so on. Every node has its BL as follows: $BL_0 = \{(1.1, \{3, 18\}), (9.1, \{11\}), (4.1, \{6.1, 19\})\}$, $BL_1 = \{(0.1, \{3, 18, 17\})\}$, $BL_2 = \{(0.1, \{1\})\}$ and so on.

A. Route Discovery

A node that has data to send initiates a route discovery by broadcasting a RREQ = (*destAddr, srcAddr, seqNum, ll*) where the *destAddr* is the address of the destination, the *srcAddr* is the address of the source node, the *seqNum* is incremented with every new RREQ the node initiates, and the *ll* is a label list to which a node receiving the RREQ appends its cluster label, only if its label has not been appended previously. Every node detects duplicate reception of the identical RREQ based on the seqNum and srcAddr fields. When a node receives a RREQ, it processes the request according to the following rules:

- *Discard rule*: If either the same RREQ was received previously, or the label of this node appears before at least one other label in ll, or this node is not a cluster head or a bridge, the RREQ is discarded.
- *Reply rule*: If this node is the destination, it creates a reply, RREP including a copy of ll and sends it toward the source node along the reverse path in ll. The way the RREP reaches the source node is similar to the one a packet data does (*Section B*).
- *Relay rule*: If this node does not satisfy both the discard and the reply rule, it appends its label to the ll of the RREQ, and then broadcasts RREQ.

B. Data Transmission

Every packet carries a source route with it. Since the source route tells only the sequence of groups identified by their respective labels, every node has to find a particular node (*bridge*) to forward. A node makes a routing decision by looking up both its BL and the source route in the packet. A node having a packet first takes the next label that comes right after its own cluster label in the source route. If this node is a cluster head, it finds a neighboring bridge that connects to the cluster with the next label of the source route by looking up its BL. If there does exist one, it forwards the packet to the selected bridge. Otherwise, the cluster head sends an error message, RERR to the source node. In case of a member having a packet, it finds a neighboring bridge that connects to the cluster with the next label of the label route by looking up its BL. If there does exist such a bridge, the node forwards a packet to the bridge. Otherwise, it forwards the packet to its cluster head. The same process continues until the packet reaches the destination.

C. Route Maintenance

In our implementation, each node that transmits a packet is confirmed for a successful data delivery over the corresponding link by exploiting the IEEE 802.11 ACK mechanism. A node that has sent a packet to the next hop without receiving ACK concludes that the link is broken. The node then selects one more link, if any, by looking up the BL. If there is no alternative link, the node initiates a route error RERR toward the source node. The source node then should explore another route to the destination.

5 Distributed Self-Pruning Algorithm

Recall that a RREQ is initiated by a source node to explore a route to destination. Following the rules in Section 4 may cause high overhead. In Fig. 4, all nodes except node 6 participate in relaying RREQ if the rules are applied. We therefore modify a distributed self-pruning algorithm named DSP from our previous work [8] to prevent intermediate nodes from relaying RREQ unnecessarily.

In DSP, every node maintains a data structure called *coverableSet* that stores labels of clusters to which the node connects and belongs, and distance in number of hops to the corresponding cluster heads. Note that the distance of a cluster head to itself is given 0 to make sure that a cluster head always has to broadcast. A RREQ carries the *coverableSet* of the sender that becomes *coveredSet* in view of the receiving nodes. Fig. 4 depicts in detail all of the nodes' *coverableSets*.

Let us explain how the DSP algorithm works. In Fig. 5, a node that receives a RREQ puts it into a queue with a calculated timeout according to Eq. (3). When the timer expires, the node decides to relay RREQ if one of the following conditions is satisfied:

1. There is at least one cluster that is covered by this node but not covered by any neighboring node from which the RREQ was received.
2. There is at least one element in coverableSet such that its distance is smaller than that of any element having the same label in the union of coveredSets of the RREQs received along different routes.

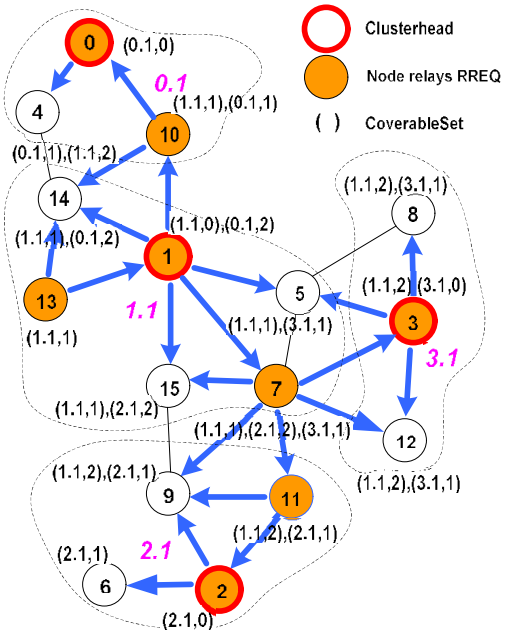

Fig. 5 The DSP algorithm (node 13 initiates a RREQ to find a route to node 6).

In DSP, if multiple nodes receive the same RREQ, which node relays the received RREQ first is extremely important. The equation that determines node's relay delay time is derived based on the following rules:

1. Cluster head always has to relay RREQ and should be earlier than the others because it may have some members that can not be reached by the RREQs from any other nodes.
2. The more clusters a node covers, the sooner the node relays.
3. Collision because of multiple initiations of RREQ should be prevented if possible.

From rules (1) and (2), Eq. (1) and Eq. (2) in Fig. 5 are derived, respectively. From (3), we include the term random(t) in Eq. (3), where t is given $\alpha \times$fixedTime, $0<\alpha\leq 1$. Basically, α has to be chosen such that Ratio×fixedTime is dominant over random(t).

```
// RREQ = (destAddr, srcAddr, seqNum, ll, coveredSet)
o Node k that receives RREQ:
  if node k is ORPHAN then
    Free RREQ;
  else
    if node k is DESTINATION then
      Send RREP;
    else            // I decide to relay or drop
      if <srcAddr, seqNum> has been seen already then
        Free RREQ;
      else
        if node k is CLUSTER HEAD then
          Ratio = 0;                                          (1)
        else
          Ratio = | RREQ.coveredSet | / | coverableSet_k |;   (2)
        endif;
        // random(t) gives a random value from 0 to t
        t = α×fixedTime;          // 0<α≤1
        Delay = random(t) + Ratio×fixedTime;                  (3)
        Put RREQ into the queue with timeout = Delay;
    endif;
  endif;
o Node k whose timer expries:
  if <srcAddr, seqNum> does not exist in seenList then
    S = coverableSet_k;
    for each RREQ the queue do
      Get element (l,d) from S;      //l: label, d: distance
      if ( ∃(l,e) ∈ RREQ.coveredSet) such that (x ≤ d)
        S = S – {(l,d)};
      endif;
    endfor;
    if (S != ∅) then
      Relay RREQ;
    endif;
    Add <srcAddr, seqNum> to seenList;
    Remove all RREQ from the queue;
  endif;
```

Fig. 6 Steps in GDSR using DSP.

For an example, let α and fixedTime be 1/6 and 30 ms, respectively. In Fig. 4, node 1 broadcasts the RREQ initiated by node 13 since it is cluster head. Nodes 5, 7 and 15 receive the RREQ from node 1, and then put it into their queues with timeout calculated by equations in Fig. 3. Delays of nodes 5, 7 and 15 are given 1

+ (2/2)×30 = 31, 5 + (2/3)×30 = 25, and 0 + (2/2)×30 = 30, respectively, where values 1, 5 and 0 all come from random(t). Node 7, although it has random(t) higher than any of the other nodes, has the smallest delay. So, it relays the RREQ before the other ones. Nodes 5 and 15 decide not to relay the RREQ after receiving the RREQ from node 7 since clusters to which they connect are covered by node 7. In case of nodes 9 and 11 that receive the RREQ from node 7, their delay times solely depends on random(t) because they have the same coverableSet size. For nodes 2 and 9, cluster head 2 always relays RREQ before node 9 although node 9 covers more clusters. Finally, in Fig. 4 only 8 nodes participate in relaying RREQ when DSP is applied to GDSR.

6 Performance Evaluation and Discussion

Our approach, GDSR, based on both cluster label and DSR was implemented on the ns2 simulator to compare with its ancestor - DSR, the cluster-based routing protocol - CBRP, and AODV. Furthermore, DSP's efficiency is also examined by simulation. Note that in figures, the notation GDSR_NO_DSP refers to the GDSR protocol without using the DSP algorithm while GDSR corresponds to GDSR using DSP.

In simulations, the IEEE 802.11standard was used as the MAC layer to avoid collision. Mobile nodes used a shared media radio with normal bit rate of 2Mb/sec. Traffic sources were Continuous Bit Rate, CBR. Pairs of source and destination were randomly chosen over the network. We used Random Waypoint Model for node mobility. Each node started moving from a random location with a randomly chosen speed. Each experiment ran ten times with the same traffic but different randomly generated mobility scenarios, and we present the average result for each metric.

Four important metrics evaluated in our experiments are:

- *Delivery ratio*: the ratio of data packets delivered to destinations to those generated by CBR sources
- *Normalized routing overhead*: the number of routing packets transmitted per data packets delivered at the destination.
- *End-to-end delay*: the sum of all possible delays caused by buffering during route discovery, propagation…
- *Average jitter*: measured as the average variance of the inter-arrival times of packets at destinations.

The first simulation with parameters listed in Table. 1 was performed with the variation of mobility speed as 0, 5, 10, 15, 20 and 25 m/s.

Fig. 6 shows that GDSR significantly outperforms the other protocols in terms of delivery ratio, especially in high mobility because of the existence of multi-paths between pairs of source and destination in mesh networks. When nodes move fast,

the gap of the ratio between GDSR and its ancestor, DSR, is about 20% while in most cases, AODV has difficulty with the ratio less than 50%.

Simulation time	500 secs
Dimension area	1500x1500 m^2
Transmission range	250 m
CBR sessions	50 sessions with 2 pkts/sec
Packet size	400 bytes
Number of nodes	100
Pause time	30 secs

Table 1 Parameters of simulation 1.

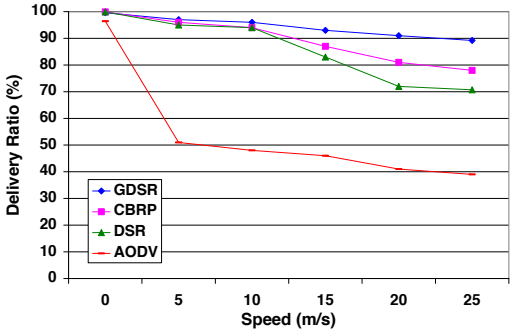

Fig. 6 Delivery ratio.

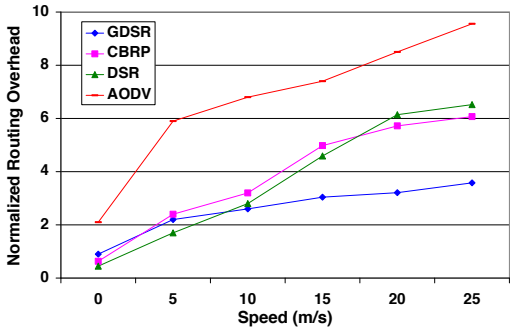

Fig. 7 Normalized routing overhead.

Referring to Fig. 7, when nodes move relatively slow, DSR that does not use any periodic Hello message generates the least overhead of them. AODV shows the highest overhead because it executes frequent route discoveries, each of which in-

volves network-wide broadcasting. As node speed increases, overhead of CBRP and GDSR become less than that of DSR because they do not flood RREQ. Especially, GDSR shows the lowest overhead overall because it not only does less route discovery by maintaining multiple paths but also reduces the number of relayed RREQs greatly by employing the DSP algorithm (refer to Fig. 10 and Fig. 11). Consequently, we can tell that GDSR is very stable with variation of node speed compared to the other protocols.

According to Fig. 8, it is shown that CBRP has the highest end-to-end delay. In case of CBRP, a node that fails to send packet to next hop saves the packet and tries to fix the route locally by finding another next hop based on 2-hop topology information. Furthermore, error is reported to source node, and a new route discovery is performed to maintain the shortest path, even though the route is fixed locally. Because of these behaviors, CBRP suffers from the high end-to-end delay. DSR shows slightly lower end-to-end delay than GDSR since DSR always tries to establish the shortest path and does not salvage a packet frequently. On the other hand, AODV has the shortest end-to-end delay, but fails to deliver a lot of packets to destinations. However, the lost packets may increase end-to-to-end delay if they are salvaged and delivered successfully along a repaired path.

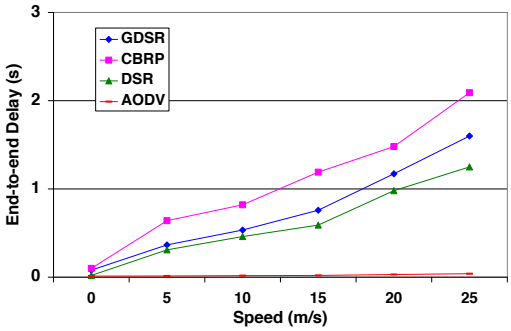

Fig. 8 End-to-end delay.

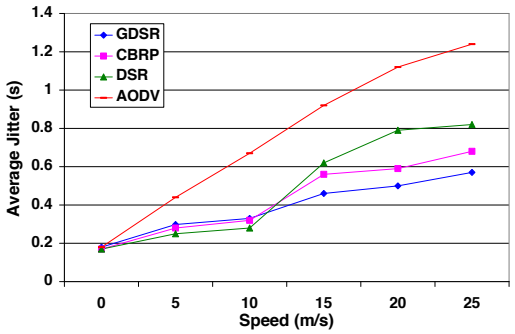

Fig. 9 Average jitter.

Fig. 9 compares the average jitter. GDSR shows slightly higher jitter than both CBRP and DSR. However, as node speed increases, it shows a competitive value in the jitter even though it maintains the high delivery ratio. AODV has the highest jitter overall since it loses lot packets.

Simulation time	500 secs
Dimension area	1000x1000 m^2
Transmission range	250 m
CBR sessions	30 sessions with 4 pkts/sec
Packet size	400 bytes
Speed	10 m/s
Pause time	30 secs

Table 2 Parameters for simulation 2.

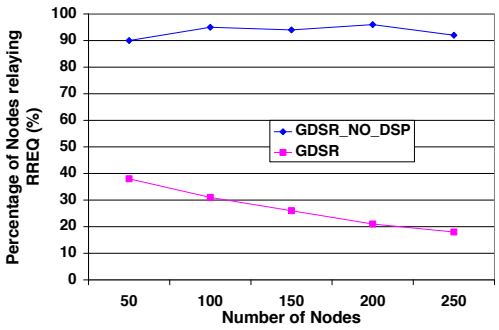

Fig. 10 Percentage of nodes that relay RREQ.

We examined the effect of the DSP algorithm applied to GDSR by comparing the percentage of nodes that relay RREQ and generated routing overhead through the second simulation with parameters listed below.

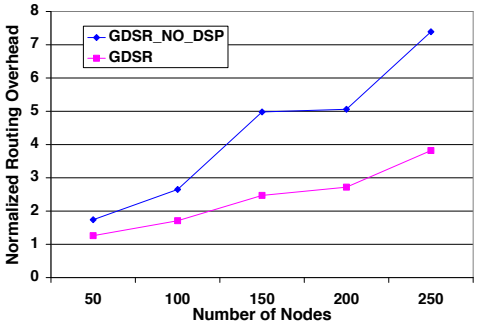

Fig. 11 Normalized control overhead.

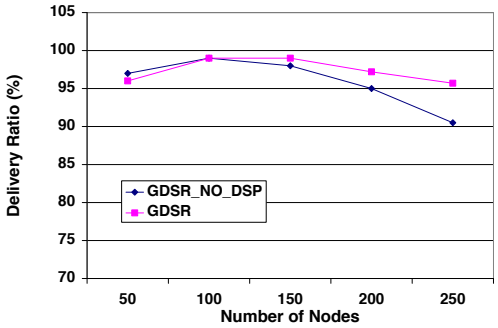

Fig. 12 Delivery ratio.

Take a look at Fig 10. In the terrain of 1000x1000 m^2, among 50 nodes, only 20 nodes (40%) relay receiving RREQs if DSP is used with GDSR while up to 45 nodes (90%) do this if GDSR works alone. In the same terrain, only 50 (18%) of 250 nodes relay RREQ, compared with 225 nodes (90%) if DSP is not applied. Obviously, when GDSR works together with DSP and the simulation terrain size is kept unchanged, the higher the density of nodes is, the smaller the percentage of nodes participating in relaying RREQ is. Fig. 11 depicts how much generated control overhead is reduced in GDSR using DSP.

Fig. 12 depicts the difference of delivery ratios between GDSR using and without using DSP. About 5% is the gap of these ratios when there are 250 nodes in the square of 1000x1000m^2. When the density of nodes increases, the collision caused by HELLO transmissions and control messages as well as data packets becomes higher. By using DSP, GDSR lessens the number of broadcasted RREQs, which leads to reduce the collision and increase delivery ratio.

7 Conclusions

We proposed a novel reactive and source routing protocol, GDSR, based on the clustering technique [5] and DSR [2]. Our protocol improves routing stability significantly over DSR by allowing a group of nodes to collaborate in delivering packets toward destination. The source label route is highly stable because it represents multiple paths between source and destination by a list of cluster labels. Actually, an effective route recovery mechanism is naturally constructed in its label route. Therefore, the control overhead caused by route rediscovery is decreased and delivery ratio is significantly improved. Data transmission does not fully depend on cluster heads; it only uses cluster head as the last resort if any other path is not available, preventing cluster heads from being congested. In addi-

tion, we proved GDSR that employs the proposed DSP algorithm improves overhead as well as reduces collision significantly.

Acknowledgments

This research was supported by the MIC (Ministry of Information and Communication), Korea, under the ITRC (Information Technology Research Center) support program supervised by the IITA (Institute of Information Technology Assessment) (IITA-2007_C1090-0701-0039)). This work was also supported by the Korea Research Foundation Grant Funded by the Korea Government (MOEHRD)" (KRF-2006-211-D00101).

References

1. Z. Haas, M. Pearlman, and P. Samar, "Zone routing protocol (ZRP)", IETF Internet Draft, draft-ietf-manet-zrp-04.txt, July 2002.
2. D. Johnson, D. Maltz, Y Hu and J Jetcheva, "The dynamic source routing protocol for mobile ad hoc networks," IETF MANET Internet Draft, Feb 2002.
3. M. Jiang, J. Li, and Y. C. Tay, "Cluster based routing protocol (CBRP) functional specification", IETF Internet Draft,draft-ietf-manet-cbrp-spec-01.txt, July 1999.
4. P. Jacquet, P. Muhlethaler and A. Qayyam, "Optimized link-state routing protocol," IETF MANET Internet Draft, Mar 2002 (work in progress).
5. V. Li, H. S. Park and H. Oh, "A cluster-label based mechanism for backbones on mobile ad hoc networks," Springer Verlag, LNCS 3970, pp. 26-36, May, 2006.
6. C. Perkins and P. Bhagwat, "Highly dynamic destination sequenced distance vector routing for mobile computers," In Proceedings of ACM SIGCOMM, 24(4), Oct 1994.
7. E. M. Royer and C. E. Perkins, "Ad-hoc on-demand distance vector routing," In 2nd IEEE Workshop on Mobile Computing Systems and Applications, pages 90–100, Feb 1999.
8. H. Oh and S. Y. Yun, "Distributed Self-Pruning (DSP) Algorithm for Bridges in Clustered Ad Hoc Networks", ICESS 2007: 699-707, May 2007.