Group Based Routing in Disconnected Ad Hoc Networks

Markose Thomas[1], Arobinda Gupta[2], and Srinivasan Keshav[3]

[1] Google R&D Center, Bangalore, India
[2] Dept. of Computer Science & Engineering
Indian Institute of Technology, Kharagpur
[3] School of Computer Science
University of Waterloo

Abstract. In this paper, we propose a routing protocol for disconnected ad hoc networks where most nodes tend to move about in groups. To the best of our knowledge, no routing protocol for disconnected ad hoc networks has been designed earlier keeping in mind possible group patterns formed by the movement of nodes. Our protocol works by identifying groups using an efficient distributed group membership protocol, and then routing at the group level, rather than at the node level. The protocol is designed so that existing concepts of routing in disconnected ad hoc networks can be extended to work at the group level. Initial simulations across a broad spectrum of parameters suggest that our protocol performs better in terms of delivery ratio and latency over traditional approaches like AODV [1], and also over disconnected routing approaches like the 2-Hop routing protocol [2].

1 Introduction

The existence of an end-to-end path is not guaranteed in many kinds of ad hoc networks because of various reasons, such as nodes switching off their radios or reducing their transmission range to conserve energy, node mobility, and application specific deployment. In such situations, traditional routing protocols for ad hoc networks such as [1, 3, 4] will fail in sending packets to destination nodes to which no path exists. An approach to solve this problem is to exploit the buffering capacities and mobility of the nodes participating in the network. If the nodes have sufficient mobility, then instead of waiting for a path to the destination, messages can be forwarded to intermediate nodes, which in turn would buffer these packets for some period of time and then forward them to other nodes. This process can be continued until some intermediate node eventually comes in contact with the destination node and delivers the message to it. Since such routing intrinsically relies on waiting for intermittently available paths, there are high latencies involved in message delivery. Applications which are able to tolerate such levels of latencies are referred to as *delay tolerant*.

Many of the delay tolerant routing protocols for disconnected networks do not assume any specific movement and location patterns of the nodes, and hence are unable to exploit any opportunity that these patterns may present. Several routing protocols have been proposed for disconnected ad hoc networks. The epidemic protocol [5] blindly floods each message to as many nodes as possible until the message reaches its destination. It is therefore very resource hungry with respect to energy consumption and buffer capacity. The 2-Hop protocol [2] showed that mobility can be used to keep throughput independent of the size of the network. However, it provides poor performance in terms of delivery ratios within practical time limits. Sushant Jain et. al. [6] formulate the general problem of routing in delay tolerant networks, and consider different levels of information availability in choosing a protocol. There has recently been research into protocols which use aggregated past information to predict future behavior ([7–9]), and thereby hope to make better routing decisions. The message ferrying approaches in [10] consider situations where dedicated *ferry nodes* are available to move around fixed routes, collect and relay packets.

An interesting class of applications within the delay tolerant disconnected ad hoc networking framework has to do with those in which nodes tend to move about in groups. This clustering of nodes leads to the formation of multiple groups of nodes (see [11–13]). Nodes are free to move about in their own groups, and also to occasionally relocate to another group. The nodes within each group will usually form a connected subnetwork, but the groups themselves would usually be in and out of communication range of each other. Further, the groups also have the ability to move about in the network, often mixing and merging with the other groups in the network and splitting into smaller groups. Some applications which work in such group environments include sensor networks deployed in wildlife [14], vehicular networks [15], and relief and military networks. None of the traditional routing protocols for disconnected environments mentioned earlier exploit the underlying group structure present in such applications.

By recognizing the presence of formation of such groups in the network, it becomes possible to combine the best ideas of both traditional ad hoc routing protocols as well as the ideas of routing in disconnected environments. For example, if members of a group are connected and if each node maintains routes to other members, it becomes possible to use traditional ad hoc routing to deliver messages and exchange routing information with nodes belonging to the same group quickly. Such a scheme would not be possible in any of the routing protocols proposed for disconnected networks. Moreover, if we treat each group as an entity in itself, much like an individual node in a normal disconnected network, then it becomes possible to apply concepts of routing for disconnected ad hoc networks to transfer messages across groups.

Our main contribution in this paper is a routing protocol for the above type of intermittently connected ad hoc networks, which is able to exploit the underlying group structure formed by node locations to provide better delivery ratios at lower latencies.

The rest of this paper is organized as follows. Section 2 gives details of the proposed routing protocol. Section 3 discusses some implementation details. In Section 4 we demonstrate, via initial simulations across many parameters, that the proposed group based protocol works better than existing routing protocols in many real life scenarios.

2 Protocol Overview

We define a group to be a set of connected nodes which maintain their connectivity for a sufficiently large period of time. The proposed protocol attempts to recognize and exploit such group structures found in many disconnected ad hoc networks, in order to provide better delivery ratios and lower latencies. The main design goals of the protocol can be classified as follows:

1. Recognize groups and maintain group identities in a decentralized and efficient manner.
2. Leverage principles of existing ad hoc routing for intra-group communication.
3. Adapt concepts of routing in disconnected environments for inter-group routing.
4. Reduce possible complexities in inter-group communication by using group leaders.

The first part of our design goal is achieved by a distributed group membership protocol. Our focus is to develop an efficient protocol, and hence we do not place the requirement that all group members have consistent views of each other at all times. However, the algorithm is expected to converge to a consistent view for all group members given sufficient amount of time. In Section 2.1 we discuss the first two of our design goals. The third and fourth design goals are described in Section 2.2.

2.1 Group Membership

Let id denote a globally unique and comparable identifier of a node. Nodes belonging to the same group are tagged with a common identifier called the gid. Groups are created and maintained so that the gid of a group is equal to the value of the lowest id of any member in the group. Initially, each node belongs to its own group. As time progresses, nodes that remain together perform merge operations and form larger groups.

Within each group, a proactive routing algorithm is run. This algorithm not only maintains up to date paths to other group members, but it also helps in timely detection of unannounced disconnections. In this paper, we use the DSDV [3] protocol, although any other proactive routing protocol can be used. Each update message that is sent by DSDV now contains one additional field - gid. It's value is set to the gid of the node sending the update packet. This helps nodes in discarding update packets sent by members of other groups. Nodes maintain paths only to other group members, and hence each node can easily find out who the other group members are by looking at its routing table.

As time progresses, nodes of other groups may come within the communication range of one or more members of a group, or some existing members may move out of range from the group. The protocol should be able to adapt to these changing scenarios quickly. Our protocol handles the first case by an explicit merge operation. It handles the second case by implicitly removing the disconnected members from the original group, and by an explicit split operation on the nodes that have left the original group. The following two subsections describe how the proposed protocol handles these two scenarios (joining and leaving).

Group Merge A group merge is triggered explicitly by a node if it detects that one or more nodes belonging to another group have come within its communication range for a sufficiently long period of time, called the minimum group merge wait time T_M.

This minimum wait time ensures that groups are not merged accidentally when they come in contact only for a very short time in the course of their movements. Each node periodically sends beacon packets advertising its gid. A node receiving a beacon packet from another group registers a hit corresponding to the group from which the beacon was received. If a node stops receiving beacons from the other group for more than some time duration T_G, it resets the hit counter for that group. Once a node receives beacons from another group for a time period greater than T_M, a *MergeRequest* packet is sent to the node which sent the last beacon. The merge request contains the id, gid, and the DSDV routing table of the node. On receiving the merge request, a node takes a local decision to accept or reject the request. If it accepts the request, then it updates its gid to the lower value of both the gid's, and also adds the nodes in the routing table of the other group to its own routing table. It then propagates this information in the form of a new DSDV update packet to the other members of its group. The other members of the group, on receiving this update packet from a group member, lower their gid if the new gid has a smaller value, and update their routing tables. They also propagate a new DSDV update packet immediately since new nodes have been added to the group (as a result of the merge). This process continues till all the nodes of the group have added the new nodes to their routing tables. The node receiving the merge request also sends a *MergeReply* packet to the sender of the request. The reply packet contains the id, gid, $status$ (accept or reject), and the routing table of the node. Similar actions as above happen on receipt of the reply packet to update the gid if needed and the routing table of nodes in the other group. At the end of the merge operation, a new group with a common gid is formed.

It is possible that multiple nodes of the same group may decide to merge with different nodes of another group at about the same time. In such a case, pairs of nodes (one node from each group) merge initially. The gid of each node in the pair will finally be the lower of the gid of the two groups. Each node in a pair will also initiate a fresh round of DSDV updates within its original group. Thus, the number of fresh DSDV updates that will take place due to a merge will be limited by the number of pairs of nodes that started the merge operation. Since each such DSDV update will have the same gid across all pairs, the routing tables of all nodes will eventually be updated consistently.

Group Split There are two main things to be done when members leave a group. Nodes still belonging to the group should realize quickly that such members have left the group. Also, since the gid of a group should reflect the value of the lowest id member in the group, the gid of the group which does not contain the lowest id node any more should be changed. However, nodes disconnect from a group in an unannounced fashion, with no prior information as to when they will leave the group. When a node leaves a group, the protocol depends on the underlying DSDV routing protocol to inform other members of its departure. If a node leaves a group, then eventually its neighbor would detect that it is no longer reachable and will remove it from its routing table. It will then send a DSDV update packet to inform the other group members of this. Thus, within a short time, the node which moved out from the group will be dropped from the routing tables of all remaining members of the group. Also, the part of the group which splits and does not contain the lowest id node, will eventually be classified as belonging to a separate group using the following mechanism. Among the nodes which belong to the above split part, the node having the lowest id will eventually discover that its id is not equal to its gid and it does not have a path to any member having id lower than it. This node then forms a new group with itself as the leader by sending an announcement to all other nodes to which it has a path in its routing table to change ther gid to its id. On receiving the announcement, all nodes which no longer have paths to their current gid (because they split from the old group) and which has the node sending the announcement as the lowest id member in its routing table, will change their gid to the id in the announcement. All these nodes also clear their existing routing tables, and build them from scratch (this is needed to ensure that these nodes do not accidentally consider the members of the old group to be part of their group).

2.2 Routing

Once the groups are identified, any standard routing algorithm for disconnected networks can be used to route between groups. Here, on the assumption that groups often move in predictable ways, we show how to adapt the PRoPHET routing protocol [7] to work in our group scenario. Like PRoPHET, we maintain probability measures of the *delivery predictability*; but instead of them being measures of successful delivery to nodes, they indicate the chances of successful delivery to other groups. These measures are stored at each node in a vector called $probTable$, which contains an entry for each group. In order for this scheme to work, it is also necessary to have up to date information regarding the group to which a destination node belongs. The protocol gathers this information in a proactive manner and stores it in a vector called the $nodeInfo$ vector, which contains an entry for each node. For the purpose of inter-group routing, the node with the lowest id is chosen as the group leader. The main responsibility of the leader is to consistently update and decay the $probTable$ vector of a group, and to disseminate the $probTable$ and $nodeInfo$ vector to the rest of the group.

Updating and Propagating Delivery Predictability and Group Information The method of updating the $probTable$ vector is the same as the approach followed by PRoPHET [7]. Let $P_{(A,B)}$ be the probability of group A being able to deliver a message to group B. Each group maintains, in its $probTable$ vector, the values $P_{(gid,B)}$, for all groups B. The group leader is responsible for carrying out the update of the $probTable$ vector. It does so when a group member informs it that a group has become newly adjacent to its group (this information is sent to the leader in the form of a $GroupUpdate$ packet). On receiving information that group B has become adjacent to its group, the leader of group A will update its group's delivery predictability value to group B as follows ($P_{init} \in (0, 1]$ is an initialization constant).

$$P_{(A,B)} = P_{(A,B)_{old}} + (1 - P_{(A,B)_{old}}) \times P_{init} \tag{1}$$

Also, a group A can deliver a message to group C indirectly through another group B. To take care of this, the delivery predictability value can also be updated transitively using the following equation ($\beta \in [0, 1]$ is a scaling constant, deciding how large a role transitivity should play).

$$P_{(A,C)} = P_{(A,C)_{old}} + (1 - P_{(A,C)_{old}}) \times P_{(A,B)} \times P_{(B,C)} \times \beta \qquad (2)$$

If two groups do not meet each other for a while, they become less likely of being able to exchange messages in the future. Hence, it is necessary for the delivery predictability values to age or decay. This is done according to the following equation, where $\gamma \in (0, 1)$ is an aging constant and k is the time elapsed since the last decay operation was carried out.

$$P_{(A,B)} = P_{(A,B)_{old}} \times \gamma^k \qquad (3)$$

The group leader periodically sends out the updated values of the $probTable$ and $nodeInfo$ vectors to the entire group in a $LeaderUpdate$ packet. The $LeaderUpdate$ packet serves two purposes. First, each node belonging to the group of the leader updates its own $probTable$ and $nodeInfo$ vectors using the values present in the $LeaderUpdate$ packet. Second, these $LeaderUpdate$ packets will also be received by nodes of other groups which lie within communcation range of some member of the group of the leader. These nodes, on receiving a new $LeaderUpdate$ packet from an adjacent group, forward it to their own group leader using a $GroupUpdate$ packet, so that their group leader can update the $probTable$ vector using equations 1 and 2. Further, the group leader, on receiving a $GroupUpdate$ packet also updates the values in its $nodeInfo$ vector, using information from the $nodeInfo$ vector contained in the $GroupUpdate$ packet. Specifically, an entry in its $nodeInfo$ vector is updated, if either the corresponding entry in the received $nodeInfo$ vector has a higher sequence number ($seqNo$), or it has the same sequence number, but has a lower gid value.

The changes in the delivery predictability values when groups merge or split depend on the specific application scenario and the movement pattern of the new merged or split group. For example, when two groups merge, we may use the weighted average (according to size) of the $probTable$ values of the two groups, or we may take the minimum of the two values, etc. For simplicity, we use the $probTable$ values of the group having the lower gid. In the case of a split, the nodes which form a new group will start with an empty $probTable$ vector, whereas no change occurs for the remaining nodes.

Forwarding Strategy The $probTable$ and $nodeInfo$ vectors at each node are used to decide whether a message should be forwarded to an adjacent node belonging to another group. The protocol uses the following simple greedy strategy for forwarding a packet. A node uses the information present in its $nodeInfo$ variable to learn the group to which the destination node belongs. We call this group as the *destination group*. A node forwards a message to an adjacent node belonging to another group, only if the other group has a higher delivery predictability value to the destination group than the group to which the sending node belongs. Thus messages are transferred from group to group, until it reaches the destination group. Once the message reaches the destination group, the message is sent to the destination node using the DSDV routing table.

Table 1. Essential Node Structures

Name	Type	Fields
id	simple	-
gid	simple	-
$seqNo$	simple	-
$leaderSeqNo$	simple	-
$rTable$	compound	$dst, nextHop, seqNo$ $hopCount, ninfoTimer$
$groupInfo$	compound	$gid, active, ginfoTimer,$ $leaderSeqNo, processedSeqNo$
$nodeInfo$	compound	$id, gid, seqNo$
$probTable$	compound	$gid, deliveryPredictability$
$mergeReqTime$	compound	$gid, lastTime$

Table 2. Constants Used

Name	Description	Value
P_{init}	Initial delivery predictability value	0.5
β	Transitivity constant	0.5
γ	Aging constant	0.98
T_M	Merge wait period	50 sec
T_U	Interval for sending $Updates$	15 sec
T_L	Interval for sending $LeaderUpdate$	10 sec
T_N	Timeout period for $nodeInfo$ entry	45 sec
T_G	Timeout period for $groupInfo$ entry	15 sec
T_{MR}	Min span between two merge request's	5 sec

Table 3. Type of Packets

Name	Members
Update	$id, gid, rTable$
LeaderUpdate	$id, leaderSeqNo, probTable, nodeInfo$
GroupUpdate	$gid, leaderSeqNo, probTable, nodeInfo$
MergeRequest	$id, gid, gidOther, rTable$
MergeReply	$id, gid, status, rTable$

3 Implementation Overview

The essential data structures maintained by each node is shown in Table 1. Some of these members (like $id, gid, seqNo, leaderSeqNo$) are simple data types, while the rest are multi-valued compound data structures. Not shown in the table are additional fields (like settling time etc.) in $rTable$ (the routing table) that will be necessary for DSDV. The data structure $groupInfo$ is maintained at each node to keep track of groups which are currently adjacent to it. Each entry has a field called $active$, which indicates whether the group corresponding to that entry is currently adjacent to the node. If an entry remains active for more than T_M time, then the node initiates a merge request (as explained in Section 2.1) with the group corresponding to that entry. The field $leaderSeqNo$ in the $groupInfo$ structure is used to discard old $LeaderUpdate$ packets received from an adjacent group. In a similar fashion, the field $processedSeqNo$ is used by a group leader to discard $GroupUpdate$ packets that it has already processed. Further, $mergeReqTime$ is a data structure that is used to limit the number of merge requests that are sent by a node to another group. A node does not send two consecutive merge requests to the same group within a span of T_{MR} seconds.

All packet types that are used by our protocol are listed in Table 3. The $Update$ packet corresponds to the DSDV update packet. The $LeaderUpdate$ packet is sent periodically by each group leader. The $GroupUpdate$ packet essentially has the same fields as a $LeaderUpdate$ packet, but it is sent by a node to its group leader. The $MergeRequest$ and $MergeReply$ packets are used for sending merge requests and replying to them. The $rTable$ member shown in the $Update$, $MergeRequest$, and $MergeReply$ packets is essentially the routing table of the sending node, with only the following fields - dst, $seqNo$, $hopCount$. Table 2 lists some constants that are used in the protocol. These values were chosen based on estimates from initial simulation results.

Since the $LeaderUpdate$ packet is sent periodically, it also acts as a beacon packet for detecting adjacent groups. Thus, when a node receives a $LeaderUpdate$ packet from another group, it sends its contents in a $GroupUpdate$ packet to its leader and also carries out the actions that were outlined in Section 2.1.

It is possible that, during a merge operation, the gid of the node receiving the merge request may change even before it processes the merge request (due to another merge operation initiated elsewhere). In that case, the merge reply sent back would confuse the node which initiated the merge request, since the gid of the replying group is different from what it is expecting. To circumvent this, the $MergeRequest$ packet contains a field called $gidOther$, which is the gid value of the adjacent group at the time the merge request is being sent. The node which sends the $MergeReply$ copies the $gidOther$ from the $MergeRequest$ packet into the gid field of the $MergeReply$ packet. The node receiving the $MergeReply$ is then able to easily find out the group it had originally sent the $MergeRequest$ to. The fact that the other group possibly has a different gid even after the merge operation is not significant, because as soon as the next round of DSDV updates are triggered, all nodes of the merged group will have a common gid.

Note that a node accepts a DSDV $Update$ packet from its neighbor only when either (i) the gid in the $Update$ packet has the same value as the node's own gid, or (ii) the neighbor is already present in the routing table of the receiving node, and the gid in the $Update$ packet has a lower value than the node's own gid. If the first condition is satisfied, then it means that the node has received an update packet from a neighbor which also belongs to its group. The second condition signifies that its group is currently taking part in a merge operation. The second condition allows a node to reject any $Update$ packet from a group member which has a higher gid than the node's gid. This condition is necessary to ensure that nodes which split to form new groups (they will have a higher gid, compared to the original group which they were a part of) do not incorporate any change received from the members of the old group. This condition might lead to an occasional loss of $Update$ packets (for example, when a node lowers its gid due to a merge operation, and then it immediately receives an $Update$ packet from a neighbor which is yet to lower its gid), but this loss of information is corrected in the next round of $Update$ packets sent.

4 Simulation & Results

We verify the effectiveness of our protocol through simulations across a wide range of scenarios on the NS-2 simulator [16]. The input parameters varied included mobility pattern, communication range, and the size of the network. We compare the performance of our protocol with that of the 2-Hop protocol and AODV. Results for AODV are included in order to highlight the ineffectiveness of standard routing protocols in disconnected environments. The performance metrics used for the comparison are delivery ratio and delay in receiving messages. We first discuss the simulation setup and then present results of our simulations.

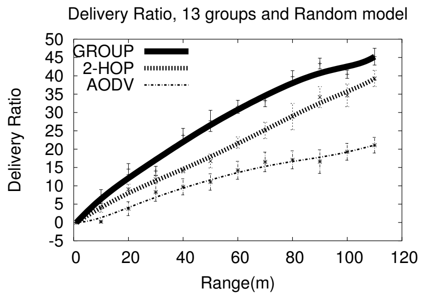
Three different mobility patterns are chosen in the simulations - *Random Waypoint Model* [11], *Inplace Model* [12], and *Community Model* [7]. In *Inplace Model*, the topology is divided into different grids. Each group is assigned a specific grid, which it never moves out of. Groups are able to interact when they meet along grid boundaries. In *Community Model*, the topology is again divided into grids. There are some designated grids called the *gathering grids*. Each group is assigned a *home grid* and also a *gathering grid*. The movement of groups is such that they tend to travel to and fro between the home and gathering grids, even though there is a slight chance that they may move to some other grids temporarily.

For the Community model, each grid has dimensions 200m$\times$200m. Each scenario has 3 randomly chosen gathering grids. If a group is in its home grid, it moves to its gathering grid with a probability of 0.90, while if it is in its gathering grid, it moves to its home grid with a probability of 0.95. For the Inplace model, each grid has dimensions 250m$\times$250m. Node assignments to groups are generated using the model in [13]. About 3% of the nodes are not placed in any group at the beginning of the simulation. In [13], nodes have to choose a new destination after reaching their current goal. The new destination can be within its group, within another group, or can also be outside of all groups. The probability of choosing these 3 types of destinations are set to 0.98, 0.01 and 0.01 respectively. For all scenarios, nodes belonging to a group move in a bounding box of 100m$\times$100m. Further, each group moves to its destination with a uniform random speed of (0,10] m/s. Likewise, each node within a group moved to its destination within the group with a uniform random speed (relative) of (0,5] m/s. The pause time for node movement within a group is set to 5 seconds, while the pause time for the group movement is set to 25 seconds.

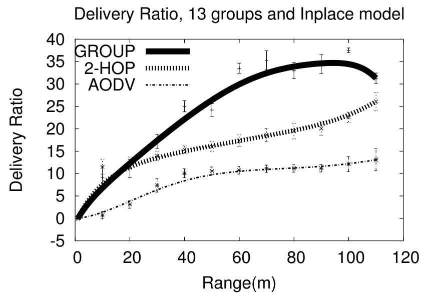
We tested 3 scenarios having different network sizes. The first scenario has 50 nodes, 7 groups, a 1000m$\times$1000m topology, 30 randomly established communicating pairs and a simulation time of 1000 seconds. The second scenario has 100 nodes, 13 groups, a 2000m$\times$2000m topology, 50 randomly established communicating pairs and a simulation time of 2000 seconds, while the third scenario has 200 nodes, 25 groups, a 3000m$\times$3000m topology, 70 randomly established communicating pairs and a simulation time of 3000 seconds. In all scenarios, each node has a buffer capacity of 1000 messages. All messages are generated as CBR traffic over a UDP connection. The underlying MAC protocol that used in our simulations is the IEEE 802.11 protocol. No measurements are taken during the first 50 seconds of the simulation to allow the protocols to stabilize.

We present results of simulation for scenario 2 (100 node case) in Figures 1 and 2. The results for the other two scenarios look similar, and due to lack of space, we do not include them here. Figure 1 shows the delivery ratio v/s communication range for different mobility models. Each point on the plot is the average value taken over 5 simulation runs. The plots also show best fit bezier curves and 95% confidence intervals. Figure 2 gives the CDF of the message delivery delays, when the communication range is 100m.

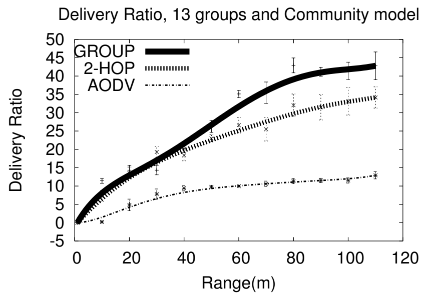
From Figure 1, we see that our protocol provides much better delivery ratios than the other two protocols for the Inplace model. Moreover, from Figure 2, we see that the delay our protocol incurs is significantly less than the 2-Hop protocol for the last set of



(a) Random Model



(b) Inplace Model

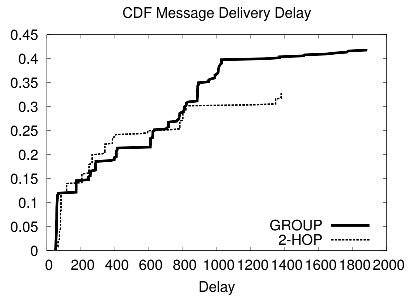


(c) Community Model

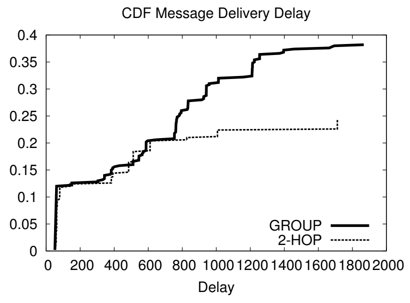
Fig. 1. Delivery Ratio % v/s Communication Range for 100 nodes, 13 groups and various mobility models

delivered messages. This is expected , as in the Inplace model, groups move in more or less predictable ways, and our protocol is able to exploit this property effectively.

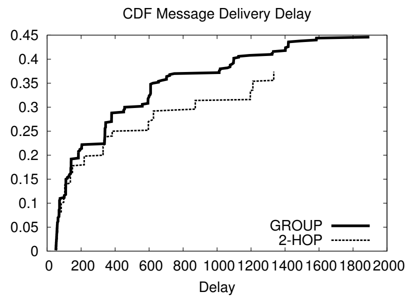
For the case of the Community model also, it is seen that our protocol performs better than the 2-Hop protocol, although the improvement is not as drastic as in the case of the Inplace model. One reason for this is that the movement of the groups is less predictable in the Community model than in the case of the Inplace model. For example, this can happen due to groups moving to grid locations which are neither their home grid nor their gathering grid. Another reason for this is that in the Community model, groups have more freedom to overlap and mingle with each other. This causes merging of some groups, even when they are not actually going to stay as a merged group in the near future. Due to this, there would be instances where groups merge, followed by a split very soon. This in turn, would cause otherwise unnecessary delays in rebuilding routing tables, and also a temporary loss of information regarding delivery predictability values.



(a) Random Model



(b) Inplace Model



(c) Community Model

Fig. 2. CDF of message delivery delay for 100 nodes, 13 groups, various mobility patterns and 100m communication range

For the case of the Random Waypoint model, we see that our protocol performs quite well in comparison to the 2-Hop protocol. However, in general, it is observed that the delivery ratio performance of our protocol varies and lies close to the delivery ratio curves for the 2-Hop protocol. This reasonably good performance of our protocol in the Random Waypoint model is slightly unexpected. This was also reported in the PRoPHET protocol simulations [7], and can be due to the fact that even in random motion, two groups that have met each other may not have moved far away from each other. In such cases, the delivery predictability values will still be useful.

5 Conclusion

In this paper, we proposed a routing protocol for disconnected networks where nodes tend to move in groups following particular mobility patterns. Initial simulations have shown that for the Inplace and Community model, our group protocol is able to deliver more messages than the 2-Hop protocol at comparable or better latencies.

References

1. Perkins, C.E., Belding-Royer, E.M.: Ad-hoc on-demand distance vector routing. In: WM-CSA. (1999) 90–100
2. Grossglauser, M., Tse, D.N.C.: Mobility increases the capacity of ad hoc wireless networks. IEEE/ACM Trans. Netw. **10** (2002) 477–486
3. Perkins, C., Bhagwat, P.: Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers. In: ACM SIGCOMM'94 Conference on Communications Architectures, Protocols and Applications. (1994) 234–244
4. Johnson, D.B., Maltz, D.A.: Dynamic source routing in ad hoc wireless networks. In: Mobile Computing. Volume 353. Kluwer Academic Publishers (1996)
5. Vahdat, A., Becker, D.: Epidemic routing for partially connected ad hoc networks. Technical Report CS-200006, Duke University (2000)
6. Jain, S., Fall, K., Patra, R.: Routing in a delay tolerant network. In: SIGCOMM '04: Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications, New York, NY, USA, ACM Press (2004) 145–158
7. Lindgren, A., Doria, A., Schelen, O.: Probabilistic routing in intermittently connected networks. In: SAPIR. (2004) 239–254
8. Musolesi, M., Hailes, S., Mascolo, C.: Adaptive Routing for Intermittently Connected Mobile Ad Hoc Networks. In: Proceedings of the IEEE 6th International Symposium on a World of Wireless, Mobile, and Multimedia Networks (WoWMoM 2005). Taormina, Italy., IEEE press (2005)
9. Jones, E.P.C., Li, L., Ward, P.A.S.: Practical routing in delay-tolerant networks. In: WDTN '05: Proceeding of the 2005 ACM SIGCOMM workshop on Delay-tolerant networking, New York, NY, USA, ACM Press (2005) 237–243
10. Shah, R.C., Roy, S., Jain, S., Brunette, W.: Data mules: modeling a three-tier architecture for sparse sensor networks. In: Proceedings of the First IEEE International Workshop on Sensor Network Protocols and Applications. (2003) 30–41
11. Camp, T., Boleng, J., Davies, V.: A survey of mobility models for ad hoc network research. Wireless Communications and Mobile Computing (WCMC): Special issue on Mobile Ad Hoc Networking: Research, Trends and Applications **2** (2002) 483–502
12. Hong, X., Gerla, M., Pei, G., Chiang, C.: A group mobility model for ad hoc wireless networks. In: Proceedings of ACM/IEEE MSWiM'99, Seattle, WA. (1999) 53–60
13. Musolesi, M., Hailes, S., Mascolo, C.: An ad hoc mobility model founded on social network theory. In: Proceedings of ACM/IEEE MSWiM '04, Venice, Italy, New York, NY, USA, ACM Press (2004) 20–24
14. Juang, P., Oki, H., Wang, Y., Martonosi, M., Peh, L., Rubenstein, D.: Energy-efficient computing for wildlife tracking: Design tradeoffs and early experiences with zebranet. In: ASPLOS, San Jose, CA. (2002)
15. Franz, W., Eberhardt, E., Luchenbach, T.: Fleetnet - Internet on the road. In: Proceedings of 8th World Congress on Intelligent Transport Systems. (2001)
16. McCanne, S., Floyd, S.: ns network simulator. http://www.isi.edu/nsnam/ns (2005)