AMRIS: A Multicast Protocol for Ad hoc Wireless Networks

C.W. Wu, Y.C. Tay
National University of Singapore
wuchunwei@alum.comp.nus.edu.sg,tay@acm.org

Abstract

This paper introduces AMRIS, a new multicast routing protocol for ad hoc wireless networks. AMRIS (Ad hoc Multicast Routing protocol utilizing Increasing id-numberS) is designed to operate independently of underlying unicast protocols.

The idea behind AMRIS is to dynamically assign every node (on demand) in a multicast session with an id-number. The ordering between id-numbers is used to direct the multicast flow, and the sparseness among them used for quick connectivity repair. A multicast delivery tree rooted at a special node called Sid joins up the nodes participating in the multicast session. The relationship between the id-numbers (and the nodes that own them) and Sid is that the id-numbers increase in numerical value as they radiate from Sid in the delivery tree. These id-numbers help the nodes dynamically leave and join a session, as well as adapt rapidly to changes in link connectivity (due to mobility etc). Messages to repair a link breakage are confined to the region where it occurs. AMRIS is simulated with PARSEC and the results reported.

This work was supported in part by National University of Singapore ARF Grant RP960683.

I. Introduction

Existing cellular wireless networks utilize fixed infrastructure, such as base stations, to provide wireless access to users. This form of wireless access has a single final hop, where users communicate wirelessly with the base station and have their data routed through some backbone connected to the base station. This is in contrast to multi-hop wireless networks (a.k.a. ad hoc networks) where no such infrastructure normally exists. Ad hoc networks have their roots in the DARPA packet radio networks[1][2] from the 1970s. Advances in mobile computing, including wireless technologies, have led to renewed interest in the use and deployment of these networks.

The dynamic nature of ad hoc networks means that existing routing protocols[3][4] that have been designed for fairly static networks are unlikely to operate well when deployed over ad hoc networks. Some multicast routing protocols designed for ad hoc networks can be found in [5][6]. However, they are dependent on an underlying unicast routing protocol.

In this paper, we propose a multicast routing protocol that is designed for ad hoc networks, and that is independent of the underlying unicast routing protocol. Section II presents a general overview of the protocol. Section III discusses the simulation model used to evaluate AMRIS, and the results are discussed in Section IV.

II. AMRIS

AMRIS is an on-demand protocol which constructs a shared delivery tree to support multiple senders and receivers within a multicast session. The key idea that differentiates AMRIS from other multicast routing protocols is that each participant in the multicast session has a session-specific multicast session member id (herein known as msm-id). The msm-id provides each node with an indication of its "logical height" in the multicast delivery tree. Each node except the root must have one parent that has a logical height (msm-id) that is smaller than it.

Each participant calculates its initial msm-id dynamically during the *Initialization* phase, which is initiated by a special node called Sid, who has the smallest msm-id. Sid is normally elected from among the set of senders if there is more than one. The relationship between the msm-id (and the node that owns it) and Sid (which is also the root of the tree) is that the msm-ids increase in numerical value as they radiate away from Sid. The msm-ids allow nodes that have broken off from the delivery tree (e.g. due to mobility, terrain) to rejoin the delivery tree in a localized fashion without causing permanent routing loops. Another key feature of AMRIS is that it does not depend on the unicast routing protocol to provide routing information to other nodes. AMRIS maintains *a Neighbour-Status* table which stores the list of existing neighbours and their msm-ids. Each node sends a periodic beacon to signal their presence to neighbouring nodes. The beacon contains the msm-ids that each node presently has.

AMRIS consists of two main mechanisms: *Tree Initialization* and *Tree Maintenance*. *Tree Initialization* is the mechanism by which a multicast session is created and advertised to nodes within the ad hoc network. Nodes that are interested in joining the multicast session (herein known as I-Nodes) then join in the *Initialization* phase. Nodes that are not interested in joining the multicast session are herein known as U-Nodes. It is important to note that U-Nodes may still become part of the multicast session subsequently when it is necessary for them to function as "intermediate" nodes within the delivery

0-7803-5538-5/99/$10.00 (c) 1999 IEEE

tree to forward multicast traffic. *Tree Maintenance* is the mechanism whereby nodes that become "detached" from the multicast delivery tree rejoin the tree to continue receiving multicast traffic, by executing a *Branch Reconstruction (BR)* routine. Nodes that did not join the multicast session during the initialization phase also make use of BR to join the tree. AMRIS uses a soft state beacon approach to determine if a link has broken between two neighbouring nodes.

A. Tree Initialization

Before *Tree Initialization* formally begins, it is necessary to determine which node will assume the role of Sid. In a single-sender, multiple-receiver session, Sid is normally the single-sender. In a multiple-sender, multiple-receiver environment, Sid may be elected from amongst the senders. The specifics of Sid election are beyond the specification for AMRIS.

Tree initialization begins with Sid broadcasting a *NEW-SESSION* message to its neighbours. The *NEW-SESSION* will contain, among other things, Sid's msm-id, multicast session id, and routing metrics. All nodes that receive the *NEW-SESSION* message generate their own msm-id by computing a value that is larger and not consecutive, so that there are gaps between the msm-ids of a sender and a receiver; these gaps are useful for quick local repair of the delivery tree. A receiver then replaces the msm-id in the message with their own, as well as various routing metrics, before broadcasting the message again. Information derived from the NEW-SESSION message is kept in the *Neighbour-Status* table for up to T1 seconds. (T1 is usually set as a multiple of the beacon interval; a suitable multiple is 3.) A random jitter is introduced between the receipt of a *NEW-SESSION* message and its subsequent rebroadcast to prevent broadcast storms. A node may receive multiple *NEW-SESSION* messages from different nodes. If it has not rebroadcast any messages yet, it will keep the message that has the best routing metrics and calculate its msm-id based on the values from that message. Otherwise the messages received are dropped.

A node X then joins the session by first determining from the *NEW-SESSION* and beacon messages received which neighbouring nodes have smaller msm-ids than X. These nodes form the set of potential parent nodes. A unicast *JOIN-REQ* is then sent to one of the potential parent nodes. When the potential parent Y receives a unicast *JOIN-REQ*, it checks if Y itself is already on the delivery tree. If so, Y will send *a JOIN-ACK* immediately back to X. Otherwise, Y too will try to locate a potential parent for itself and send a *JOIN-REQ* to it. This process is repeated until a node can satisfy the requirements of being a parent node. That node will send a *JOIN-ACK* which propagates back along the reverse path towards X, grafting a branch from the tree to X. Joining is first attempted through contacting a neighbouring node; if that fails, a localized broadcast is then used. If the immediate neighbouring nodes are already on the multicast tree, then this 1-hop 'peek' approach is very fast and efficient. The use of msm-ids helps a node identify a neighbor (who as a potential parent) provides a higher likelihood of a successful join. If a node is unable to find any potential parents, then the requesting node will execute the *Branch Reconstruction (BR)* process in its continued attempt to rejoin the tree.

B. Tree Maintenance

The tree maintenance mechanism operates continuously in the background to ensure that a node remains connected to the multicast session delivery tree. When a link between two nodes breaks, the node with the larger msm-id (a.k.a. the child node) is responsible for rejoining. A node attempts to rejoin the tree by executing the *Branch Reconstruction (BR)*, which has two main subroutines, BR1 and BR2. BR1 is executed when the node has neighbouring potential parent nodes that it can attempt to join to; BR2 is executed when the node does not have any neighbouring nodes that can be potential parents.

Basically, BR1 works as follows: The node X executing BR1 selects a potential parent node Y from its set of potential parents. It then sends a *JOIN-REQ* to Y; if Y is already a registered member on the multicast tree and has a smaller msm-id than X, it will send a *JOIN-ACK* back to X, acknowledging its request, and X has now successfully rejoined the tree. If Y is not yet a member on the tree, then it repeats the process of sending out its own *JOIN-REQ* to join the tree, provided it has at least one neighbouring potential parent node. Otherwise, it sends a *JOIN-NACK* back to X. If X receives *a JOIN-NACK* or timeouts on the reply, it will proceed to join with the next best potential parent node. If none are available, X executes the BR2 subroutine.

BR2 is executed when a node X is unable to detect any neighbouring potential parent nodes. Instead of sending a unicast *JOIN-REQ* to a single potential parent node (as in BR1), X sends a broadcast *JOIN-REQ*. The broadcasted *JOIN-REQ* has a range field R that specifies only nodes within R hops of X are allowed to rebroadcast the *JOIN-REQ*. The purpose of the range field R is to localize as much as possible the effects of a BR routine without resorting to a network-wide broadcast in searching for new potential parent nodes.

When a node Y receives a broadcasted *JOIN-REQ*, it checks if it can satisfy the request. If so, Y sends a *JOIN-ACK* on the reverse path set up back to X. However, Y does not forward multicast traffic to X yet, since X may receive more than one *JOIN-ACK* in response to its broadcast *JOIN-REQ*. When X receives the *JOIN-ACKs* (it may receive more than one from different nodes), it will choose one of them to become the parent node and send a *JOIN-CONF* to that parent node. When the potential parent node receives the *JOIN-CONF*, it will now forward any multicast traffic it receives to its new child.

If a node does not have a valid msm-id and wishes to join, it first uses neighboring msm-ids to compute an msm-id for itself, then execute the BR routine to join the session.

III. Simulation Environment

We experimented with AMRIS using an ad hoc network simulator written in PARSEC[7], which is a discrete event simulation language developed at UCLA. The network consists of 100 mobile nodes moving about randomly (Brownian motion model) in a 1000x1000m two-dimensional space. Radio transmission range was set at 150m. The program simulated a CSMA MAC layer with a free space propagation model. Data rate was set at 2Mb/s.

Our preliminary goal for the simulation was to understand the protocol's routing behaviour and detect any major flaws it has. Therefore we used a relatively light traffic model to minimize congestion effects. Each data packet had a data portion (excluding headers) of 100 bytes and was generated at a rate of 1 per 100ms. The parameters varied were the beacon interval (from 500ms to 4000ms), the number of I-Nodes per multicast session (from 25 to 100, one of which was randomly chosen to be Sid), and the maximum movement speed (from 1 to 20m/s). The metrics measured were packet delivery ratio (pdr), routing overhead and end-to-end delay. Each run simulated 200 seconds of simulation time.

IV. Simulation Results

In the following, each sample point in each graph is an average from 20 simulation runs. Fig. 1 compares the packet delivery ratio with varying beacon intervals, membership sizes (membership size refers to the number of members in a multicast session.) and mobility rates. The packet delivery ratio is fairly good, with most figures in the upper quartile range. Generally, as mobility increases, the packet delivery ratio decreases for all cases. This is in part due to the soft state nature of the protocol, which uses timeouts to determine that a neighbouring member node (which may be a parent or a child node) is no longer around. Therefore, packet losses inevitably occur between the time a node is "broken off" from its neighbouring member nodes and the time that node actually "realizes" the breakage.

As larger beacon intervals are used, the packet delivery ratio drops significantly more at higher mobility rates. The timeout value that determines a neighbouring node is set as a multiple of the beacon interval (we set it at 3). Therefore, with a large beacon interval, a node takes significantly longer to realize that its neighbour parent node has moved away. This leads to a significantly higher number of packets not received at large

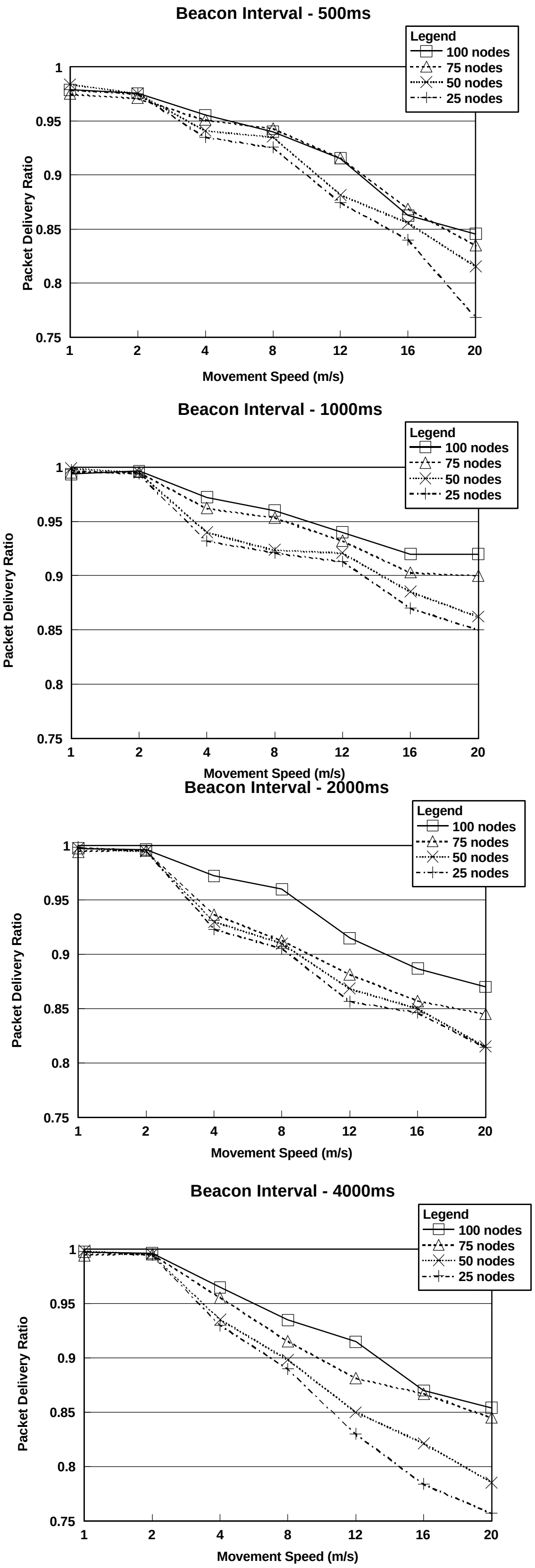

Fig. 1. Packet Delivery Ratio

beacon intervals. The drop is larger for small membership sizes because there is a smaller number of potential parent nodes around a node when it tries to rejoin the tree. For big membership sizes, when a node discovers the breakage, it can usually find a parent node nearby, and can rejoin the tree more quickly. Multicast sessions with more I-Nodes also generally perform better than those with less I-Nodes since nodes are better able to quickly find neighbors that are already registered on the tree. Nodes close to Sid have a higher pdr than those further away since they are usually within a single hop from Sid. Sessions that have more I-Nodes also have more nodes that are close to Sid, thus increasing the pdr.

It is interesting to note that with a small beacon interval of 500ms, the packet delivery ratio drops significantly at higher mobility rates compared to other beacon intervals. Further investigation shows that when the beacon interval is small, there is an increase in the number of link breakages being detected. A large number of link breakages are what we call micro-term breakages. As the nodes move about in a random fashion, they frequently move just out of range of each other for just a short while (micro-term) before moving back into range again. The effects of these micro-term breakages are more evident when the beacon interval is small. This causes the nodes to execute *Branch Reconstruction (BR)* to rejoin the tree. The increase in packets sent leads to increased packet collisions around those nodes, causing packets to be dropped and decreasing the packet delivery ratio.

The four graphs in Fig. 1 show that there is an optimum beacon period that should not be too small or too large. More studies need to be done to find out the relationship between this optimum and the node densities, movement speeds and traffic models.

Fig. 2 illustrates the results for routing overhead. Routing overhead is calculated as the ratio of control packets (e.g. *JOIN-REQs*, *JOIN-ACKs*; the beacons are not included as they are a constant overhead) sent versus all (data and control) packets sent. We count packets instead of bytes because the control packets are small in size (around 20 bytes) compared to the data packets (100 bytes). If the ratio of control bytes versus total bytes sent were measured, then the routing overhead would be very small. Furthermore, counting bytes would have ignored the cost of acquiring the medium to transmit a packet, which is relatively independent of packet size once the medium has been acquired. Counting packets therefore provides a clearer view of routing overhead.

When the beacon interval is small, there is noticeably higher routing overhead. This is due to more nodes superfluously initiating BR as a result of micro-term link breakages. The

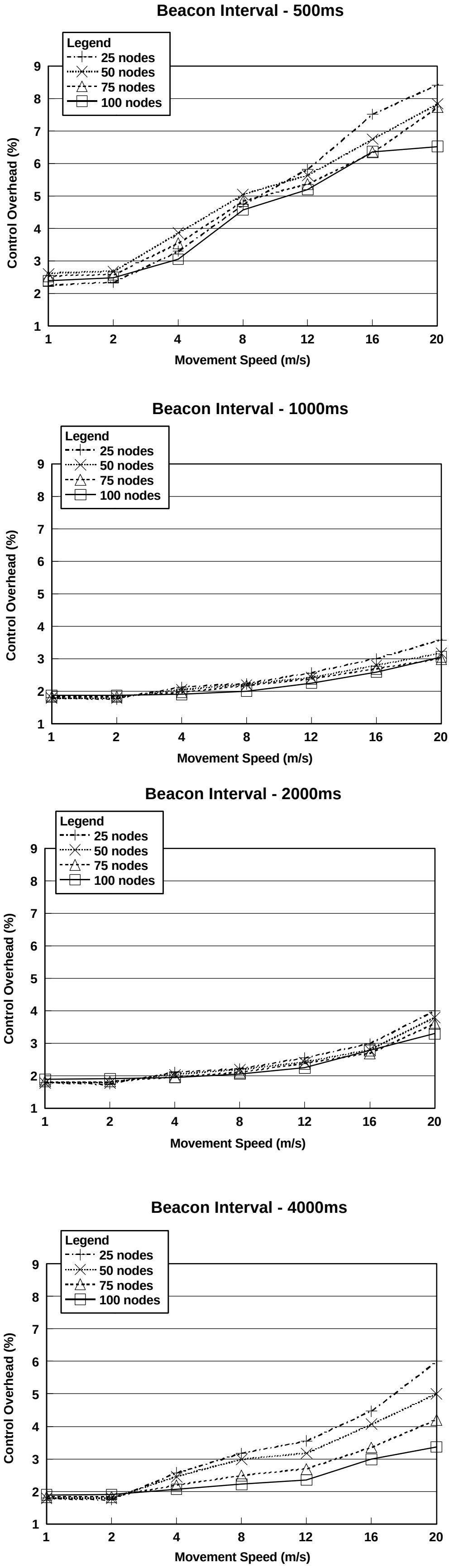

Fig. 2. Control Overhead

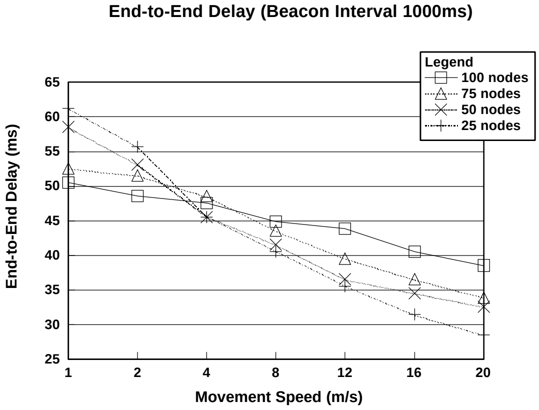

Fig. 3. End to End Delay

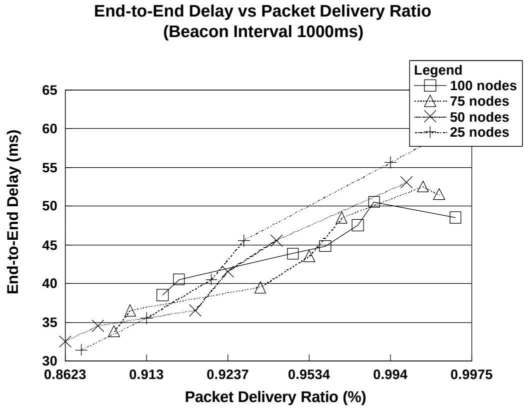

Fig. 4. End-to-End Delay vs PDR

increase in packet collisions result in dropped data packets which further increases the ratio of control packets to total packets. At high mobility rates, a large membership has lower routing overhead compared to a small membership. Again this is because of the localized repair feature of BR which queries neighbouring nodes rather than doing a localized n-hop broadcast. The routing overhead results again show that there is an optimum beacon period.

End-to-end delay is considered as the average time taken by a packet to reach an I-Node from the time it leaves the sender. In reality, nodes closer to the sources will usually have a smaller end-to-end delay than nodes further away. The metric is calculated as follows: When a data packet is first created by the source, it is tagged with a send time. Subsequently, each I-node that receives the packet calculates the end-to-end delay by subtracting the time the packet was received with the initial send time. An average is then taken from all I-nodes. The end-to-end delay is thus measured only for packets that are received. This is why, in Fig. 3, we see that, as the maximum movement speed is increased, the end-to-end delay actually drops.

Fig. 4 shows explicitly this relationship between end-to-end delay and packet delivery ratio. Interestingly, the curves for all four membership sizes are clustered together, thus showing that the relationship between the two metrics is robust with respect to membership.

It takes on average 5ms for each data packet to travel 1 hop. (The reason for the large 5ms per hop is because a random jitter with maximum of 50ms is introduced between data packet reception and retransmission) Therefore, with a maximum average end-to-end delay of 62ms for 25 receivers, we can estimate the average hop traversed by the data packet along the delivery tree.

V. Conclusion

We have proposed a new multicast routing protocol designed for use in ad hoc networks. AMRIS orders the nodes within the multicast delivery tree logically with a dynamically derived number. This ordering is used to direct multicast traffic, and the sparseness among these numbers facilitates quick local repair to the delivery tree. Our preliminary simulation results show that AMRIS has high delivery ratio and low overheads, and is thus feasible as a multicast routing protocol for ad hoc networks.

The simulations indicate that some improvements are possible. One of them is the criteria for selecting which potential parent node to send the *JOIN-REQ* if there is more than one to choose from. Presently, we pick the node with the smallest msm-id. However, from the simulation, we observe that this may not give a good route: a potential parent with the smallest msm-id may be further away, so the link may be weaker than if another nearby potential parent is selected.

References

[1] Leiner, B.M, Neison, D.L. and Tobagi, F.A, "Issues in Packet Radio Network Design". Proceedings of the IEEE Special issue on "Packet Radio Networks",, 75,1:6-20, 1987
[2] Jubin, J and Tornow, J.D, "The DARPA Packet Radio Network Protocols", In Proceedings of IEEE, volume 75, 1, pages 21-32, Jan. 1987
[3] Deering, S.E., Partridge, C., and Waitzman, D., "Distance vector multicast routing protocol", RFC 1075, Nov 1988
[4] Estrin, D., Farinacci, D., Helmy, A., Thaler, D., Deering, S., Handley, M., Jacobson, V., "Protocol Independent Multicast-Sparse Mode (PIM-SM): Protocol Specification", RFC2117, June, 1997
[5] C.-C. Chiang, M. Gerla, and L. Zhang, "Forwarding Group Multicast Protocol (FGMP) for Multihop, Mobile Wireless Networks", ACM-Baltzer Journal of Cluster Computing: Special Issue on Mobile Computing, vol. 1, no. 2, 1998.
[6] C.-C. Chiang, M. Gerla and L. Zhang, "Shared Tree Wireless Network Multicast", In Proceedings of IEEE IC3N '97.
[7] Meyer, R.A., PARSEC User Manual, August 1998.