This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the ICC 2007 proceedings.

On Supporting Robust Voice Multicasting over Ad hoc Wireless Networks

G. Venkat Raju [a], T. Bheemarjuna Reddy [b], and C. Siva Ram Murthy [b]
[a] Yahoo! Software Development India Pvt. Ltd., Bangalore, India 560001
[b] Department of Computer Science and Engineering, Indian Institute of Technology Madras, India 600036
gvraju@yahoo-inc.com, arjun@cs.iitm.ernet.in, murthy@iitm.ac.in

Abstract— **In this paper, we address the problem of voice multicasting in ad hoc wireless networks. The unique characteristics of voice traffic (*viz.* small packet size, high packet rate, and soft real-time nature) make conventional multicasting protocols perform quite poorly, hence warranting application-centric approaches in order to increase robustness to packet losses and lower the overhead due to high packet rate. By exploiting the path diversity and the error resilience properties of Multiple Description Coding (MDC), we propose a Robust Voice Multicast Routing (RVMR) protocol. Our protocol uses a novel path based Steiner tree heuristic to reduce the number of forwarders in each tree, and constructs two trees in parallel with reduced number of common nodes among them. Moreover, unlike other on-demand multicast protocols, RVMR specifically attempts to reduce the periodic (non on-demand) control traffic. We propose several optimizations for reducing the overhead while transmitting high packet rate voice traffic in ad hoc networks. We extensively evaluate RVMR in the NS-2 simulation framework and show that it out-performs existing single-tree and two-tree multicasting protocols.**

I. Introduction

Ad hoc wireless networks (AWNs) are formed by a set of mobile nodes that communicate with each other over a wireless channel without the help of any pre-existing infrastructure. Nodes co-operate to forward packets for effecting communication between any two nodes that are not directly within the wireless transmission range of one another. Due to quick and cost effective way of deployment, these networks are attractive to numerous potential applications, ranging from emergency and rescue operations to real-time multimedia communications for disaster areas. Real-time multimedia applications can tolerate packet losses to some extent (up to 5% [1]) but are highly delay sensitive (typically for interactive voice communication, the *end-to-end delay* should be less than 200 *ms* [2]). In this paper, we concentrate on voice multicasting as it is a key application in many group-oriented scenarios. The unique characteristics of voice traffic, such as small payload size (20 bytes), high packet rate (40 packets/s to 100 packets/s), and soft real-time nature make voice multicasting a very challenging problem in AWNs.

Although there exists some video multicast routing protocols [3][4] besides several general multicast routing protocols [5] for AWNs, they fail to sustain high packet rate and hence not suitable for voice multicasting. This warrants development of application-centric multicasting protocols which increase the error resiliency (i.e., robustness to packet losses) and reduce the total data overhead for voice traffic. The recent advances in Multiple Description Coding (MDC) have made it highly suitable for multimedia applications in AWNs [6]. MDC offers a way wherein we can split a voice frame into two independent packets (also called descriptions), such that even if one of them is received the frame can be recovered although with degraded quality. This improves both robustness and continual flow of the voice frames. We can further increase the error resilience by employing path diversity (existence of two maximum node-disjoint paths between a source-receiver pair) and sending each description along an independent path to the receiver. By making these paths as node-disjoint as possible, we decrease the correlation of path breaks among them, hence increasing the probability of at least one description reaching the receiver.

In our recent work, by exploiting the error resilient properties of MDC and path diversity, a multiple tree video multicasting protocol, Robust Demand-driven Video Multicast Routing (RDVMR) protocol [7] was proposed and was shown to perform well over both ADMR [8] and MDTMR [3] protocols with minimum overhead. RDVMR uses a novel path based Steiner tree heuristic to reduce the number of forwarders in each tree, and constructs multiple (k) trees in parallel with reduced number of common nodes (i.e., nodes that are forwarders for more than one tree) to improve the robustness due to path breaks and correlated packet losses and it attempts to reduce the periodic (non on-demand) control traffic. Therefore, each receiver has k maximally node-disjoint paths to the source, along which different MDC descriptions are sent. However, under high packet rates the RDVMR protocol fails to performs well due to large overhead and thus limiting its application for voice multicasting. By adding several key optimizations (as explained in Section II-B) to RDVMR protocol, we propose RVMR protocol that sustains high packet rate (voice) traffic in AWNs with minimal overhead. The rest of the paper is organized as follows. Section II describes RVMR protocol in detail. In Section III we evaluate RVMR protocol through simulations, and finally in Section IV we conclude with possible future work.

II. An Overview of RVMR

In order to make RVMR as demand-driven as possible (i.e., to minimize the non on-demand control traffic), we base RVMR on RDVMR protocol. RDVMR is a multi-tree based video multicasting protocol that exploits path diversity along with the error resilience properties of MDC to achieve an improved performance over single [8] and multi-tree protocols [3]. RDVMR is a receiver-initiated multicasting protocol, and it uses soft-state tree repair. Its salient features are: 1) There is no periodic control traffic like beaconing, or link state updates. 2) Most of its control information is piggybacked on data packets. 3) It is standalone, i.e., it does not require any underlying unicast routing protocol. Our

RVMR protocol builds two trees simultaneously, where each tree has lesser number of forwarders, and it employs several key optimizations (and thus minimal overhead (data+control)) to sustain high packet rate voice traffic compared to RDVMR protocol.

A. Data Structures

The following data structures are maintained by each node:

- ***TreeState(t,groupId)*** This stores the parent $parent$ (next hop to the source) for the tree t of the multicasting group $groupId$. It also stores the last sequence number $lastSeq$ heard from the source through $parent$, and the cost $cost$ to reach the source through $parent$. All nodes, including nodes that are not part of any multicasting tree, keep track of their parents.
- ***NodeState(d)*** This stores the unicast routing information for the destination d, namely the next hop $nxtHop$, the last sequence number heard $lastSeq$, and the cost $cost$ to reach node d through $nxtHop$. It is updated on hearing any packet originating from node d, similar to the backward routing protocol [9].

B. Protocol Overview

In this subsection we describe how RVMR builds and maintains two trees. RVMR is a tree-based receiver initiated multicasting protocol, where receivers join and leave on-demand. RVMR adapts each tree to the continuously changing network topology and to the changing group membership. In order to maintain two trees, each packet contains the $treeId$, identifying the tree it is meant for, in addition to the $groupId$. As an optimization, by maintaining the $treeId$ in each packet as a bit-vector of size two (named as $treeList$), a single control packet can be meant for two trees simultaneously. Note that, when we say that a node gets or sends a packet p along the tree t, we mean that $p.treeList$ has the t^{th} bit set. Hence by means of using a $treeList$ in each packet, RVMR is able to build and maintain two trees in parallel.

The source of the group $groupId$ multicasts data packets along the tree. It also periodically floods a data packet having a piggybacked header called $SrcJoinAdvt$, which eventually reaches every node in the network. It advertises a route to the source of $groupId$ to all nodes in the network, and helps to refresh the multicast tree. Since the $SrcJoinAdvt$ packet advertises a route to the source along all trees, it has its $treeList$ set to all ones, whereas the packets multicasted by the source advertise a route to it only along a particular tree t. On hearing such a route advertisement to the source of the group $groupId$, a node creates (or refreshes) the entry $TreeState(t, groupId)$. This way every node keeps track of its upstream node (parent) for the tree t to reach the source. The source also keeps track of the mean inter-packet time, ipt, which is piggybacked on every packet it sends. A tree node (i.e., a node that is a forwarder or a receiver) is said to be disconnected if it has not received any multicast packet before the expiry of $RepairTimer$[1]. When a receiver wishes to join the tree t, it unicasts a $joinReq$ packet to its parent for the tree t and sets a $JoinTimer$ based on ipt advertised by the source. If it does not have a parent or if the $JoinTimer$ expires, the receiver floods a $MulticastSolicitation$ packet for the tree t. When a tree node (i.e., a node that is a forwarder or a receiver for tree t) gets a $MulticastSolicitation$, it unicasts it up the tree through its parent to the source. On receiving a $MulticastSolicitation$, the source unicasts a $joinReply$ packet to the corresponding receiver. A node forwarding a $joinReply$ packet to the node d for the tree t, becomes a forwarder for that tree.

If a forwarder for tree t detects a disconnection, it multicasts a $repairNotification$ packet downstream. A $repairNotification$ serves to inform the downstream nodes of the disconnection, and avoids redundant repair attempts by them when they eventually detect the disconnection. Simultaneously the forwarder detecting the break, attempts to locally repair the tree by flooding a limited *time-to-live (ttl)* $LocalReconnect$ for the tree t. Similar to handling a $MulticastSolicitation$, a tree node unicasts a $LocalReconnect$ up the multicast tree t to the source. The source then unicasts a $LocalReconnectReply$ to the node that originated the local repair. A $LocalReconnectReply$ is processed exactly like a $joinReply$, $i.e.,$ any node forwarding it becomes a forwarder for the tree t. In order to avoid routing loops we only allow the source to respond to flooded packets like $MulticastSolicitation$ and $LocalReconnect$. If a receiver detects a disconnection, it attempts to rejoin the group after some time by flooding a $MulticastSolicitation$ in case the repair attempts of its upstream nodes ($i.e.$, nodes that are ancestors of this node along the multicast tree) fail. It can be clearly seen that the repair control overhead increases with mobility because of the increasing link breaks.

RVMR does not have explicit $leave$ messages, instead forwarders prune themselves away if they detect that they do not have any downstream children. Each multicasted data packet carries a field containing the parent of the node that forwarded it. This field serves to passively acknowledge the parent of that node to continue forwarding data. However, since pure receivers (last-hop nodes in the multicast tree) do not forward data packets, they need to explicitly acknowledge their parents by periodically unicasting an acknowledgment packet. This way a forwarder not having any downstream receivers prunes itself away from the multicasting tree. RVMR has three routing packets responsible for building and maintaining two trees. They are $MulticastSolicitation$, $SrcJoinAdvt$, and $LocalReconnect$ packets originated by a receiver, source, or a forwarder for a particular group, respectively. To reduce the redundancy due to high packet rate voice traffic, we now propose a few optimizations.

C. Proposed Optimizations

1) Tackling Premature Timeouts: For high packet rate voice applications, the ipt value is usually very small (typically 20 ms). This causes a serious problem in both ADMR and RDVMR protocols and is explained as follows. In reply to a $SrcJoinAdvt$ packet, a receiver sends back a $joinReply$ packet and it expects a data packet before the expiry of $JoinTimer$[1]. Since for voice traffic the ipt value is very small, there are higher chances for the expiry of $JoinTimer$ before the data actually reaches along multiple hops. If the data

[1]The $JoinTimer$ and $RepairTimer$ in ADMR are specified in terms of the number of packets, k, where k is a small value such as 2.

packet sent by source has not reached the receiver before the expiry of $JoinTimer$, it falsely assumes that the previously sent $joinReply$ packet is lost in the network though it is actually delayed due to multiple hops present between source and receiver.

The receiver, however repeats the process of sending $joinReply$ packets two more times, before it finally gives up and sends $MulticastSolicitation$ message. Eventually, the source delivers a data packet to the receiver, and this causes another round of triple $joinReply$ messages followed by solicitations. Because solicitations are flooded and sent using broadcast at MAC layer, they cause severe congestion causing loss of large number of packets in the network. Since voice packets have delay bound, these packets are treated as lost packets if they reach the destination after their playout time (deadline). Note that, $RepairTimer$ also have the same problem as that of the $JoinTimer$, resulting in false disconnections and false initiation of multicast tree repair mechanisms causing enormous overhead in the network. To solve this problem, we need to adjust the $JoinTimer$ and $RepairTimer$ as explained below.

Optimization: We use $MaxWaitTime$, so that the calculated time for $JoinTimer$ and $RepairTimer$ can never be less than this value. By setting the $MaxWaitTime$ to a reasonably high time ($\approx$ 1,000 ms), we can ensure that the timer never fires too soon (this is equivalent to missing of 50 voice packets), but is fast enough to adapt to losses due to mobility in the network.

2) Tackling ACK Implosion: Another problem that is common in RVMR, ADMR, and RDVMR protocols is ACK implosion. Since pure receivers do not forward data packets, they need to explicitly acknowledge their parent nodes (upstream nodes) by unicasting an ACK packet for each data packet they received to maintain the forwarding state. This ACK packet overhead causes increased network traffic thus leading to collisions and reduction in packet delivery ratio. We propose the following two optimizations to reduce the ACK packet overhead to a minimum level.

Optimization I: Our solution to the ACK implosion is that for maintaining the forwarding state at the source (or forwarder), it is sufficient that a pure receiver sends one ACK packet for every k received data packets because ipt is very small for voice applications. This way the source (or forwarder) can maintain the forwarding state with minimum overhead. The source (or forwarder) sets its forwarding state timer to $AckWaitTime$ which is equal to $k * MaxAckTime$. This ensures that the source (or forwarder) does not time out its forwarding state unless it misses k ACKs in a row. This optimization is very important for high packet rate voice traffic since it reduces the ACK packet overhead substantially.

Optimization II: Since each ACK is broadcasted, each pure receiver sets an $AckTimer$ to a random value between zero and $MaxAckTime$. If the timer expires and the receiver has received voice packet during this interval then it sends an explicit ACK. However, if the receiver overhears an ACK sent by another pure receiver in its neighborhood during this interval, it cancels its timer and does not send ACK packet (refer Figures 1(a)-(c)). Similarly, if the receiver overhears the same voice packet sent by a tree node in its neighborhood during this interval, it cancels its timer and does not send ACK packet (refer Figures 2(a)-(b)). It then waits for the remainder of $MaxAckTime$ before it sets a new ACK timer.

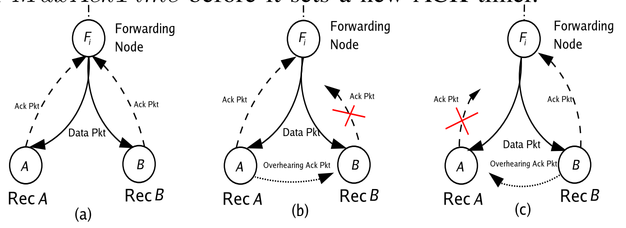

Fig. 1. Tackling ACK implosion problem - Optimization II(a).

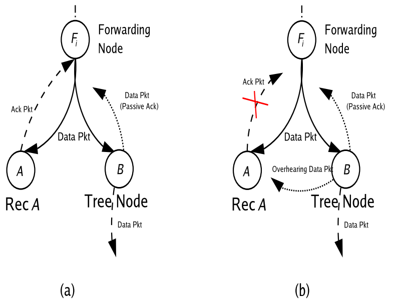

Fig. 2. Tackling ACK implosion problem - Optimization II(b).

D. Handling of Routing Packets

This subsection describes the handling of routing packets in RVMR. RVMR has three routing packets responsible for building and maintaining two trees. They are $MulticastSolicitation$, $SrcJoinAdvt$, and $LocalReconnect$ packets originated by a receiver, source, or a forwarder for a particular group, respectively. Each node in a multicast tree needs to learn of a loop free route to the source along that tree. RVMR employs backward routing. In backward routing, a node S floods a routing advertisement for itself (i.e., it sets $r.advertisedNode$ to S), and any node B hearing such an advertisement r from the node A, may choose A as the next hop to node S. A routing advertisement packet r contains a field $r.cost$ to store the cost of the path it has taken so far starting from its source (which is same as $r.advertisedNode$) and a field $r.prevHop$ which is set to the node that forwarded r. This is illustrated in Fig. 3(a).

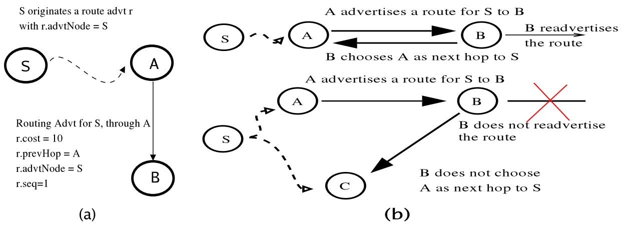
S originates a route advt r with r.advtNode = S
S
A
B
Routing Advt for S, through A
r.cost = 10
r.prevHop = A
r.advtNode = S
r.seq=1
(a)
A advertises a route for S to B
B readvertises the route
B chooses A as next hop to S
B does not readvertise the route
C
B does not choose A as next hop to S
(b)

Fig. 3. Handling of routing (a) Illustration of backward routing and (b) Node B drops node A's routing advertisement for S as it does not choose it as the next hop to the node S.

We can ensure loop free paths if we impose the following two conditions: 1) The cost in the routing advertisement packet monotonically increases and 2) A node is allowed to forward a routing advertisement r, only if it is received from it's next hop to the node $r.advertisedNode$ (shown in Fig. 3(b)). We can easily prove using contradiction that the above two conditions guarantee loop free paths. Assume that even in

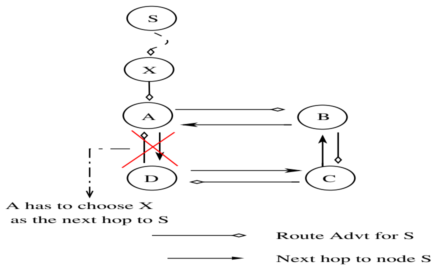

Fig. 4. Example scenario of a routing loop originating at node A.

the presence of these conditions routing loops can be formed. Such a scenario is illustrated in Figure 4. Without loss of generality let us assume that node S originates the routing advertisement which is further forwarded by nodes X, A, B, C, and D. Let us assume that a routing loop is formed at node A. Since node B forwarded the routing advertisement it obtained through node A, node B must have chosen A as its next hop to S, similarly node C chooses node B as its next hop to S and node D chooses node C. As the cost carried in the routing advertisement monotonically increases, node A receives a higher cost route to node S from node D than it received from node X. Hence it should have chosen node X and not node D as its next hop to node S. Hence we have proved that it is impossible to form a routing loop if our routing protocol follows the above two conditions.

In order to adapt to the changing network topology, each routing packet r, must carry a monotonically increasing unique sequence number seq, to differentiate between stale and new routing information. In order to construct Steiner like trees (i.e., trees with reduced number of forwarders), the route from the multicast source S to any node A in the multicast tree should not be the shortest path from S to A. Therefore RVMR not only needs to differentiate between stale and new routing advertisements, it must also make sure that it does not degenerate into finding shortest path trees. To achieve this, a node A on hearing a routing advertisement packet r for the node $r.advertisedNode$, may choose $r.prevHop$ as the next hop for $r.advertisedNode$, iff $(r.cost < NodeState(r.advertisedNode).cost)$ AND $(r.seq >= NodeState(r.advertisedNode).lastSeq)$. Essentially a node chooses a neighbor to be a next hop for a destination, only if it has received a routing advertisement for that destination from that neighbor which has a higher (hence newer) sequence number and a better cost. The cost field $r.cost$ is updated by the cost function (refer Algorithm 1).

E. Steiner Tree Heuristic

In this subsection we describe our novel path-based Steiner tree heuristic, which tries to reduce the number of additional forwarding nodes required by differentially costing each node along each tree. The details of the proposed heuristic are abstracted into a cost function $CostOfNode$, by means of which we are able to impose a routing gradient so as to heuristically reduce the number of forwarders leading to a decreased NPO. Each node adds its cost as returned by the function $CostOfNode$ to the cost fields in the routing packets (namely $SrcJoinAdvt$, $MulticastSolicitation$, and $LocalReconnect$). The following psuedocode describes the function $CostOfNode$. The

Algorithm 1 $CostOfNode$

```
$node \leftarrow$ Node that is calculating its cost
$groupId \leftarrow$ Multicast group address
$baseCost \leftarrow$ NON_PART_OF_TREE_COST
$treeId \leftarrow$ Tree along which cost is being calculated
**if** $node$ is a receiver for tree $treeId$ of multicast group $groupId$ **then**
    $baseCost = baseCost * \alpha$
**else if** $node$ is a forwarder for tree $treeId$ of multicast group $groupId$ **then**
    $ncr \leftarrow numReceiverDescendants(treeId, groupId)$
    $baseCost \leftarrow (baseCost * \beta)/ncr$
**end if**
**if** $numTreesPartOf(groupId) > 1$ **then**
    $disjointedCost \leftarrow$
    $\lambda * NON\_PART\_OF\_TREE\_COST$
    $*(numTreesPartOf(groupId) - 1)$
    return $\gamma * (disjointedCost) + (1 - \gamma) * baseCost$
**else**
    return $baseCost$
**end if**
```

function $numTreesPartOf(groupId)$ returns the number of trees this node is a forwarder of for the group $groupId$. The function $numReceiverDescendants(treeId, groupId)$ returns the number of receiver descendants in the subtree rooted at $node$ for the tree $treeId$ of the group $groupId$. We keep track of the number of receiver descendants under a node for a particular tree by piggybacking this information on join requests and passive acknowledgments for that particular tree. We set α to be much lower than β, because we want a receiver to have a much lower cost than a forwarder. Both α and β are less than one. The parameters γ and λ control the trade off between the node-disjointedness and the number of forwarders, as discussed earlier. The cost function is illustrated

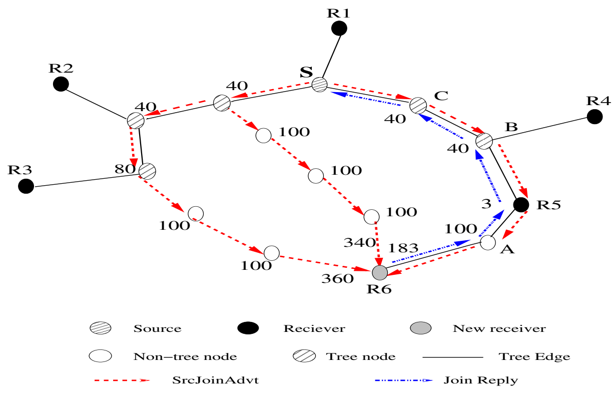

Fig. 5. An example to illustrate the cost function.

in Fig. 5. In this example, source S floods a join advertisement having a piggybacked cost. Node $R6$ is the new receiver. The join advertisements arrive at node $R6$ through three paths, of which the one through A and $R5$ is chosen as it has the least cost. In this example $NON\_PART\_OF\_TREE\_COST = 100$, $\alpha = 0.03$, $\beta = 0.8$, and $\gamma = 0$. The cost of each node and of the join advertisements received by node $R6$ along different paths are shown in the figure.

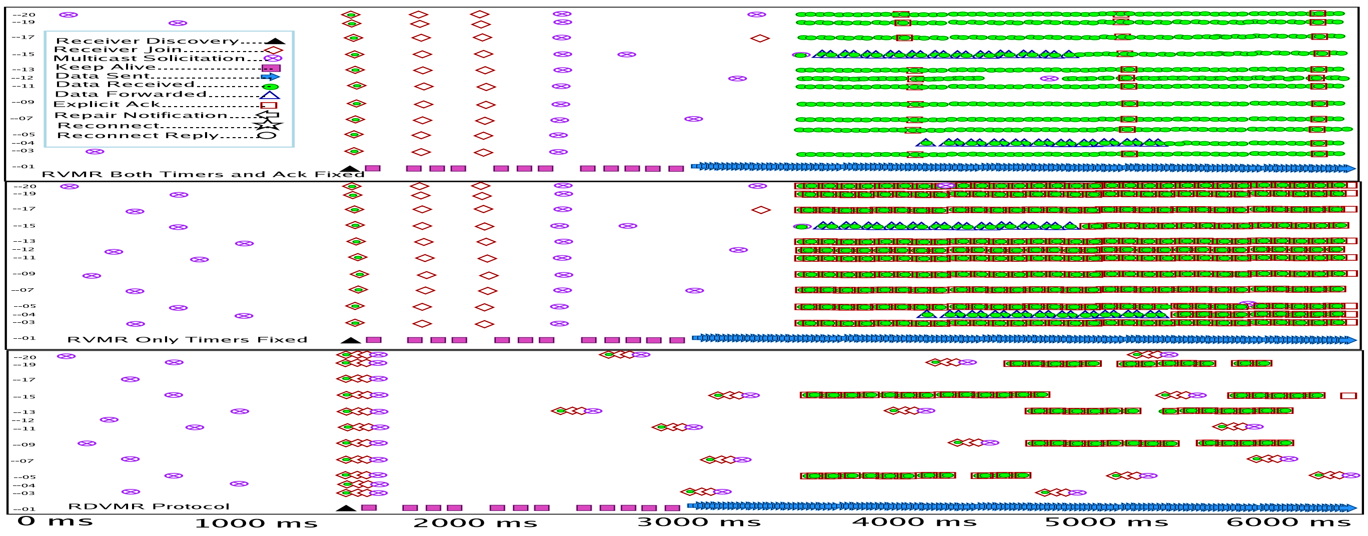

Fig. 6. Illustration of packet trace for RDVMR and RVMR protocols.

TABLE I

SIMULATION PARAMETERS

Parameter	Value	Parameter	Value
Terrain Area	1200 *m* x 800 *m*	Tx. Range	250 *m*
Channel Capacity	11 Mbps	# of Nodes	75
Mobility Model	*Random Way Point*	Sim. Duration	900 *s*
Source data rate	50 *pkts/s*	# of Receivers	25
MAC Protocol	802.11 DCF	Voice Frame Size	253 bits
Traffic Type	CBR	*ipt*	20 *ms*

III. SIMULATION STUDIES

A. Simulation Framework

We implemented RVMR in the NS-2 simulation framework [10]. We compare its performance with RDVMR and ADMR. We first compare our protocol for the single tree case with ADMR and RDVMR for various scenarios, and then using two trees we compare our performance with RDVMR. For all experiments we set the parameters α, β, γ, and λ in RVMR to be 0.03, 0.8, 0.2, and 2, respectively. These values were observed to give good results in our simulation setup. We evaluate the performance using the following metrics: 1) **Packet Delivery Ratio** (the ratio of the number of data packets received by each receiver over the number of data packets sent by the source), 2) **Normalized Packet Overhead** (the ratio of the total number of packets (control and data) exchanged over the total number of data packets received by all the receivers), and 3) **Measurement of Perceptual Evaluation Speech Quality Mean Opinion Score (PESQ-MOS)**. The PESQ-MOS is evaluated as follows. At each receiver, the voice frames are decoded and the wide band version of ITU perceptual measurement algorithm, PESQ-MOS reference software tool [11] is used to measure their perceived voice quality. The PESQ-MOS reference software tool compares the degraded speech with the reference speech and computes the objective MOS value in a 5-point score ranging from -0.5 (worst) to 4.5 (best). With respect to a original raw voice frame, the voice quality scores of different voice frames are evaluated using PESQ-MOS reference software tool. The evaluated voice quality scores of (a) raw voice frame, (b) decoded AMR-WB voice frame, and (c) decoded bits of AMR-WB voice frame that corresponds to basic quality are 4.5 (Ideal Quality), 3.818 (Optimal), and 2.86, respectively. The optimal quality score (3.818) corresponds to the decoding of AMR-WB (lossy encoded) voice frame assuming no losses in the network. We implemented RVMR in the NS-2 version 2.1b8. The simulation parameters are shown in Table I. The source sends data throughout the simulation period and 25 of the total nodes are randomly chosen to be receivers. Each of these receivers joins at a random time instant, chosen uniformly from $(4, 450)$ seconds. The receivers do not leave the multicast session. All the results presented in this paper were averaged over 30 simulation runs and all the results conform to 95% confidence levels. Each node moves with some constant speed (i.e., min speed is equal to max speed) with zero pause time. The playback deadline is 200 ms, if a packet is not received within its playback deadline it is considered lost. We use AMR-WB (Adaptive Multi-Rate Wide Band) speech codec with 12.65 Kbps bit rate with a sample size of 253 bits. For each AMR-WB voice frame, we generate two descriptions MDC-1 and MDC-2 of sizes 136 and 134 bits, respectively following the procedure given in [6]. In RVMR, the values of *AckWaitTime, MaxAckTime* are set to *3.3 s* and *66 ms*, respectively. That means, the receiver will send ACK packet for every 50 pkts (i.e., 1 sec). The *JoinTimer* and *RepairTimer* values were set to 500 *ms* to avoid premature timeouts. The packet trace of the above scenario for both RDVMR and RVMR protocols is shown in Fig. 6. It contains three sections corresponding to RDVMR original, RVMR with Timers fixed, and RVMR with both Ack and Timers fixed. X-axis represents the time duration and Y-axis indicates the traffic at receiver nodes. Node 1 is the sender for the group. The trace shows only the first 6 seconds of the network activity; the trends shown in the figure continue for the remaining duration. We can observe that when both Timers and Ack are fixed, the overhead in the network is at minimum.

B. Simulation Results

1) Effect of High Packet Rate: We fix the periodicity of flooding $SrcJoinAdvt$ to be 30 seconds in all protocols for

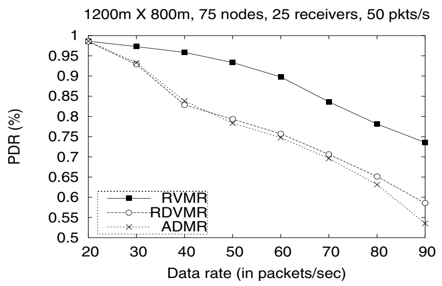

Fig. 7. PDR vs. Data rate.

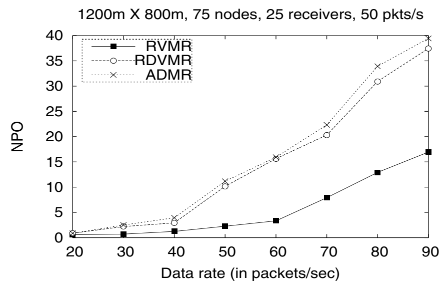

Fig. 8. NPO vs. Data rate.

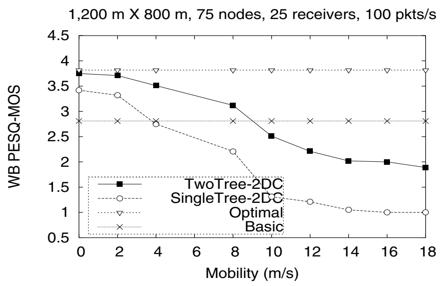

Fig. 9. WB PESQ-MOS vs. mobility.

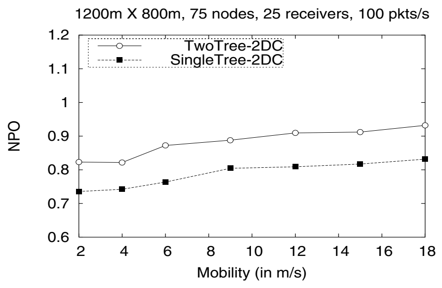

Fig. 10. NPO vs. Mobility.

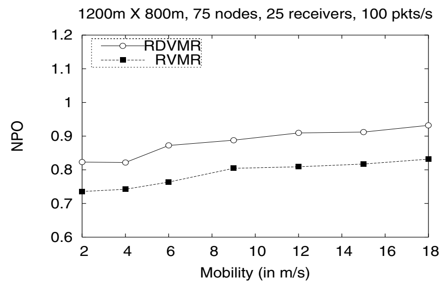

Fig. 11. NPO vs. Mobility.

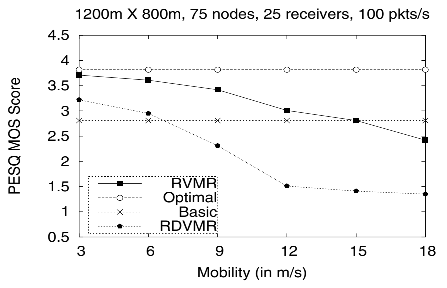

Fig. 12. WB PESQ-MOS vs. Mobility.

uniformity sake with a static scenario. Both RVMR and RDVMR protocols use single tree for transmitting voice packets. It can be seen from Fig. 7 that the PDR of both ADMR and RDVMR protocols decreases rapidly as the data rate increases beyond 30 pkts/s. Since RVMR controls premature timeouts and ACK implosion as mentioned earlier, it can cope up with high packet rate up to 60 pkts/s well. Since RVMR employs several key mechanisms, it reduces the overall network traffic causing minimal NPO as observed in Fig. 8.

2) One Tree vs. Two Tree MDC Voice Multicasting: In our subsequent experiments, we study the advantages and disadvantages of using multiple trees for MDC voice multicasting. In this experiment we compare two schemes carrying two MDC descriptions: *SingleTree-2DC* and *MultiTree-2DC*. The SingleTree-2DC scheme carries both MDC descriptions on a single tree with a total rate of 100 pkts/s. The MultiTree-2DC scheme uses two trees, each description is sent over a different tree with a per tree rate of 50 pkts/s. Thus in both cases, the total rate is equal to 100 pkts/s. It can be seen from Fig. 9 that the PESQ-MOS is significantly high in scheme MultiTree-2DC compared to scheme SingleTree-2DC though the NPO is slightly higher as observer in Fig. 10. Hence, sending each of the two descriptions on different independent trees is advantageous in AWNs.

3) Comparison of RVMR with RDVMR: We next compare RVMR's performance with that of RDVMR. Both RDVMR and RVMR send two MDC descriptions on two trees (i.e., one description per tree). Each description has a data rate of 50 pkts/s. The RDVMR uses a naive two-tree construction, hence it has a very high overhead at high packet rates which can be seen in Fig. 11. The measured WB PESQ-MOS at the destination for varying mobility is shown in Fig. 12. As observed in the figure, the RVMR protocol outperforms RDVMR due to its minimal overhead at high packet rates.

IV. Conclusion and Future work

In this paper we proposed an effective low overhead, two-tree based voice multicast routing protocol that exploits path diversity along with the error resilience properties of MDC to achieve an improved performance over single and two-tree protocols. Simulation results showed that RVMR outperformed RDVMR in terms of PESQ-MOS, PDR, and NPO. For our future work, we would like to explore novel ways of improving both PDR and voice quality by sending forward error protected data along with MDC descriptions.

References

[1] N. Jayant and S. W. Christensen, "Effects of Packet Losses in Waveform Coded Speech and Improvements Due to an Odd-Even Sample-Interpolation Procedure", *IEEE Transactions on Communications*, vol. 29, no. 2, pp. 101–109, February 1981.

[2] ITU-T Recommendation, G.114: "One Way Transmission Time", February 1996.

[3] W. Wei and A. Zakhor, "Multipath Unicast and Multicast Video Communication over Wireless Ad hoc Networks", *in Proc. of BROADNETS 2004*, pp. 496–505, October 2004.

[4] Shiwen Mao, Xiaolin Cheng, Y. Thomas Hou, and Hanif D. Sherali, "Multiple Description Video Multicast in Wireless Ad hoc Networks", *in Proc. of BROADNETS 2004*, pp. 671–680, October 2004.

[5] C. Siva Ram Murthy and B. S. Manoj, *Ad Hoc Wireless Networks: Architectures and Protocols*, Prentice Hall, New Jersey, USA, 2004.

[6] G. Venkat Raju, T. Bheemarjuna Reddy, Shyamnath Gollakota, and C. Siva Ram Murthy, "On Supporting Real-time Speech over Ad hoc Wireless Networks", *in Proc. of IEEE ICON*, pp. 421–426, September 2006.

[7] D. Agrawal, T. Bheemarjuna Reddy, and C. Siva Ram Murthy, "Robust Demand-Driven Video Multicast over Ad hoc Wireless Networks", *in Proc. of IEEE BROADNETS*, October 2006.

[8] J. G. Jetcheva and D. B. Johnson, "Adaptive Demand-driven Multicast Routing in Multi-hop Wireless Ad hoc Networks", *in Proc. of ACM MobiHoc*, pp. 33–44, October 2001.

[9] Yogen K. Dalal and Robert M. Metcalfe, "Reverse Path Forwarding of Broadcast Packets", *ACM Communications*, vol. 21, no. 12, pp. 1040–1048, December 1978.

[10] NS-2: Network Simulator, *http://www.isi.edu/nsnam/ns/*

[11] Proposed Modification to P.862 to Allow PESQ to be Used for Quality Assessment of Wideband Speech, *ITU-T SG12 Delayed Contribution COM-D007-E*, Feb 2001.