Reliable, Adaptive, Congestion-Controlled Ad hoc Multicast Transport Protocol: Combining Source-Based and Local Recovery

Venkatesh Rajendran[*], Yunjung Yi[†], Katia Obraczka[*], Sung-Ju Lee[‡], Ken Tang[§], and Mario Gerla[†]

[*] Computer Engineering Department, University of California, Santa Cruz

[†] Computer Science Department, University of California, Los Angeles

[‡] Internet Systems & Storage Laboratory, Hewlett-Packard Laboratories

[§] Scalable Networks Technologies

Abstract

This paper introduces the Reliable, Adaptive, Congestion controlled multicasT protocol, or ReACT, designed to provide reliable and timely multicast delivery for a wide range of mobile ad hoc network (MANET) scenarios. ReACT combines source-based congestion and error control with localized recovery. For congestion control, it uses simple stop-and-wait to adjust to congestion experienced by receivers. To recover from the different types of losses that may occur in MANETs, ReACT uses both source-based and local recovery. Ideally, while the latter will recover from localized losses caused by transmission errors or "local" congestion, the former will only be used when receivers experience "global" congestion losses. In this case, congestion control will also be triggered.

Through extensive simulations, we evaluate ReACT's performance and compare it against other reliable multicast mechanisms for a wide range of MANET conditions. Our results show that ReACT is the best performer in terms of reliability. Our results also showcase the effect of ReACT's local recovery mechanism which manages to prevent the source from reducing its rate unnecessarily, thus achieving acceptable throughput. In our experiments we run ReACT over a mesh- and a tree-based multicast routing protocols to show that its performance is fairly insensitive to the underlying routing structure. Finally, we also show that ReACT is fair to

competing TCP traffic.

Keywords: ad hoc networks, reliable multicast, local recovery, mobile networking, transport protocols.

Technical Area: Mobile Computing and Communication

1 Introduction

A mobile ad hoc network (MANET) [4] [6] [7] [11] operates without any fixed infrastructure or central administration. Each host communicates with each other through wireless packet radios. Because of the limited radio propagation range, routes can often be multihop. Hence, every host may act as a packet forwarder as well as a source or a destination. Because of its ease of deployment, a multihop (wireless) ad hoc network, or MANET, is an attractive choice for scenarios where the fixed network infrastructure is non-existent or unusable. Example applications include digital battlefield, search and rescue, disaster recovery, and covert military operations.

The types of scenarios targeted by MANETs make group-oriented services such as data dissemination and teleconferences a key application domain. Undoubtedly, multicast communication is an efficient means of supporting group-oriented applications. This is especially true in MANETs where nodes are energy- and bandwidth limited. In these resource-constrained environments, reliable point-to-point protocols (viable in wired networks) can get prohibitively expensive: the convergence of multiple requests to a single node typically causes intolerable congestion, violating the time constraints of a critical mission, and may drain the node's battery, cutting short the network's lifetime. Despite the fact that reliable multicasting is vital to the success of mission critical applications in MANETs, surprisingly very few efforts focus on reliable multicast.

In this paper, we propose the Reliable, Adaptive, Congestion-controlled multicasT (ReACT) transport protocol for reliable and timely multicast delivery in a wide range of MANET scenarios. One distinguishing feature of ReACT is that it combines source-based and local recovery. Source-based control includes a rate-based congestion control algorithm that guarantees data delivery to troubled receivers in a round-robin fashion. It adjusts the sending rate using a simple stop-and-wait mechanism based on receivers' feedback.

Wireless environments are prone to packet losses due to various factors other than congestion, including transmission errors, hidden terminal, and route breaks caused by mobility. Based on the observation that in MANETs congestion is not the sole reason for losses, ReACT's error control mechanism combines source-based and local recovery. Ideally, the local recovery mechanism will mend non-congestion losses, avoiding the multicast source to back-off its transmission rate unnecessarily. Local recovery also tries to catch losses due to "localized" congestion (e.g., caused by the hidden terminal effect). Besides preventing the source from unnecessarily slowing down, ReACT's local recovery mechanism also manages to reduce "global" congestion by suppressing feedback ultimately addressed to the source as well as source-originated retransmissions. Also, by recovering locally, timeliness is preserved.

Through extensive simulations, we evaluate ReACT's performance under a wide range of MANET conditions, comparing it against (multicast) UDP and Anonymous Gossip [3], which, to the best of our knowledge, is the only other end-to-end protocol for reliable multicasting in MANETs that has been proposed to date. AG's error control relies exclusively on local recovery.

One of our goals was to also analyze the performance of ReACT over different underlying multicast routing structures, namely tree and mesh. As shown in previous studies [9], mesh-based routing protocols typically exhibit very high packet delivery ratio at low traffic due to path redundancy. At high traffic however, redundant forwarding adds to network congestion and hence is less effective. Tree-based protocols, on the other hand, are known to achieve good packet delivery ratio efficiently in static MANET scenarios, but deliver poor performance in highly mobile environments it because of frequent tree breaks that require excessive control overhead for repairs. Our hypothesis was that its congestion and error control schemes would shield ReACT's performance from the nuances of the underlying routing fabric. In our performance analysis, we use On-Demand Multicast Routing Protocol (ODMRP) [8] as a representative of mesh-based protocols, and an improved version of Multicast Ad hoc On-Demand Distance Vector (MAODV) [13] [1] as a representative of tree-based protocols. Our simulation results show that ReACT performs comparably well with both ODMRP and MAODV.

[1] We describe in detail our changes to the original MAODV protocol in Section 4

We should point out that ReACT favors reliability over throughput. ReACT's congestion control mechanisms slows down the sender when congestion is detected so that packets that are already in the network can get through. Clearly, the side-effect is a decrease in throughput. Applications that are willing to trade throughput for reliability include military covert operations and search and rescue missions. However, as our simulation results demonstrate, local recovery manages to lessen the reliability-throughput tradeoff improving ReACT's overall throughput while still maintaining close to perfect reliability for a variety of scenarios.

The remainder of this paper is organized as follows. Section 2 overviews related work. ReACT is introduced in Section 3 which presents a detailed description of the source-based and local recovery mechanisms. Performance evaluation follows in Sections 4 and 5 which describe the experimental environments used and results obtained, respectively. Section 6 presents our concluding remarks and directions for future work.

2 Related Work

The Reliable Multicast Algorithm (RMA) [5] is a multicast routing protocol that offers reliable data delivery through the use of retransmissions at the source. Each source maintains a list of all members of the group and from this list ensures that all members have successfully received all transmitted packets (each members reply with ACK to source). Furthermore, rather than traditional hop count, the link's expected lifetime is used as routing metric to favor stable links. The drawback of RMA is the fact that each member must reply to the source with ACKs. This may lead to ACK implosion at the source and thus does not scale well. Furthermore, congestion control is not addressed and hence RMA would suffer under high network load.

In [10], a reliable broadcast protocol was proposed to address the problem of reliable atomic delivery of messages. The protocol's operation is divided in two phases: *scattering* and *gathering*. During scattering, a source propagates the data to all members; ACKs are collected by the source in the gathering phase. While this protocol may work well in stable networks with low mobility and low failure rates, its performance will likely degrade in dynamic MANET scenarios where topology changes are frequent. In fact, simulation results reported in [10]

indicate that the protocol does not perform well in the presence of high node mobility.

Anonymous Gossip (AG) [3] recovers from losses by having pairs of multicast group participants exchange information on messages they have received or lost. AG uses solely local recovery for error control. AG's local recovery mechanism is based on the concept of gossip dissemination and works by having participating nodes recover lost packets from anonymous nearby members instead of sending their requests directly to the source. In AG, a node that has experienced packet losses periodically sends a gossip request message to an anonymously chosen neighbor node. The gossip request includes information on lost packets, sequence number of the next expected packet, the source and the group addresses. If the receiving node is a group member, it unicasts back a gossip reply to the initiator. If the receiving node is not a member, it randomly selects a neighbor and relays the gossip message. Periodic gossip messages can generate excessive control overhead. To reduce this overhead, gossiping to members farther away are performed less often than to nearby members. To implement this feature, each node is required to know the distance (in hop count) to group members. This information is obtained from the underlying multicast routing protocol. Since gossip-based approaches are known to converge slowly, one potential problem with the AG protocol is the delay it takes for nodes to recover from losses. Clearly, there is a tradeoff between message delivery delay and the overhead incurred by the protocol.

3 Reliable Adaptive Congestion Controlled Multicast Transport Protocol

ReACT is a reliable multicast transport protocol designed to adapt to the varying conditions of MANETs. The basic protocol implements a source-based recovery scheme using both congestion and error control to prevent and recover from packet losses due to congestion. ReACT also performs local recovery at intermediate nodes to recover from losses not necessarily due to (global) congestion. The remainder of this section describes both components of ReACT in detail.

3.1 Source-Based Recovery

ReACT congestion control mechanism uses feedback from the receivers to detect congestion and adjust the transmission rate. Each multicast source maintains a *Receiver List*, which is initially empty initially. When a source receives a NACK from a node that is not in the *Receiver List*, the node is added to the list. Nodes are removed from the list when the source receives an ACK from them. In addition, the source keeps track of the end-to-end latency between itself and each receiver that sent NACKs through a timestamp.

The source initially multicasts data packets at the rate specified by the application and continues to do so until a NACK is received. Whenever a receiver receives a new packet, it is added to the receiver buffer and delivered to the application in sequence. A receiver sends a NACK back to the source when it detects packet losses. More about detection of packet losses and feedback mechanism is described in Section 3.2. Upon reception of a NACK, the source adds the NACK sender to the *Receiver List* and enters the loss recovery phase. The source initiates loss recovery by selecting a receiver from the *Receiver List*, or the *feedback receiver*. The source multicasts a new packet or retransmits the lost packet requested by the feedback receiver. The packet header includes information instructing the feedback receiver to reply via "unicast" with an ACK indicating that all packets have been successfully received or specifying the sequence number(s) of packets that are still missing. All other receivers process the packet without replying to the source.

When there are lost packets, the feedback receiver sends an ACK to the source with the sequence number of lost packets one at a time until it has all up-to-date packets (i.e., send-and-wait). The design philosophy behind retransmitting one packet at a time is to slow down the source when congestion is detected. Since only the feedback receiver replies to the source, the ACK/NACK implosion problem is avoided. Moreover, to reduce the response overhead, NACK transmissions are rate limited to no more than once every κ seconds (κ value is five in our simulations).

By having only the feedback receiver respond to the source, we also avoid the inaccuracy of feedback suppression timers. This is particularly advantageous in a mobile environment where the majority of suppression

techniques will fail since they often rely on distances and end-to-end delay measurements. Such measurements are often in flux when mobility exists and are ineffective in suppressing feedback.

Note that retransmitted packets are *multicast* to the group since other receivers may also be missing these packets. The expectation is that as the source "visits" other receivers, those receivers would already have recovered the lost packets previously retransmitted by the source and thus no further retransmissions are required. Duplicate packets are discarded by the receivers. Once the feedback receiver obtains all the packets, it unicasts an ACK to the source indicating all the packets have been received. Upon reception of the ACK, the source removes the node from the *Receiver List*, chooses a new feedback receiver in a round robin fashion, and repeats this process until the *Receiver List* is empty. Feedback receiver selection is not restricted to the round robin approach, although we do so here for simplicity. Other alternatives include selecting the feedback receiver based on common loss packets to reduce the number of retransmissions and oldest packet lost to limit latency.

When the *Receiver List* is empty, the source reverts to the application sending rate. If, however, the source does not receive a NACK or ACK from the feedback receiver within the time interval given by the measured round-trip time from the source to the feedback receiver, the source backs off and tries again up to a maximum number of times (which is three in our simulations). Since we use a stop-and-wait approach, we also fix up the maximum back-off time to prevent the source from reducing the rate too much. If the source still does not hear from the feedback receiver after the maximum number of retransmissions, it removes the node from the *Receiver List* and moves on to the next node in the list. The removed receiver may later re-synchronize with the source with the normal NACK mechanism.

The round-robin send-and-wait approach does not require retransmissions of the same lost packets multiple times to each receiver. In the best-case scenario, lost packets are retransmitted only once by the source since retransmissions are multicast. For instance, if a set of receivers lost the same packet, it is retransmitted only once assuming the retransmitted packet is received by all the receivers. In the worst-case scenario, each receiver experiences different packet losses. In this case, all lost packets must be retransmitted to each receiver. Remember

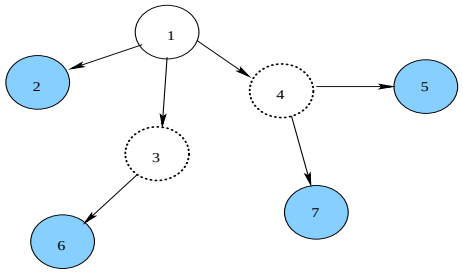


Figure 1: ReACT in operation.

also that when loss recovery mode is entered, the source first limits its rate and sends a new packet to the feedback receiver. This prevents the source from retransmitting packets multiple times.

Figure 1 shows an example of ReACT's basic protocol operation. Node 1 is the source and nodes $2, 5, 6$ and 7 are the multicast receivers. Node 1 begins by multicasting packets at the application sending rate. Assume that after some duration, nodes 6 and 7 detect packet losses and each transmits a NACK to node 1. Upon receiving NACKs, node 1 adds nodes 6 and 7 to its *Receiver List.* Node 1 then selects a node from the list, say node 6 as the feedback receiver, and multicasts a new packet (if any) or retransmits the requested lost packet. This data packet will instruct node 6 to respond to the source. All other receivers, including node 7, are prohibited from sending any feedback, thus avoiding the feedback implosion. Node 1 waits for either node 6 to reply or a timeout to occur. If an ACK is received from node 6 requesting retransmissions, node 1 continues to multicast the lost packet indicated in the ACK. If a timeout occurs, the packet is retransmitted after a back-off. When an ACK is received without any request for retransmissions, node 1 removes node 6 from the *Receiver List* and chooses node 7 as the next feedback receiver to perform loss recovery. It is important to note that retransmitted packets are multicast. If node 7 missed the same packets as node 6 and node 7 received them when they were retransmitted, node 7 does not need to request retransmission of those packets in the ACK. Upon receipt of an ACK from node 7, node 1 removes node 7 from the list. Since the *Receiver List* is now empty, node 1 reverts to the original application sending rate.

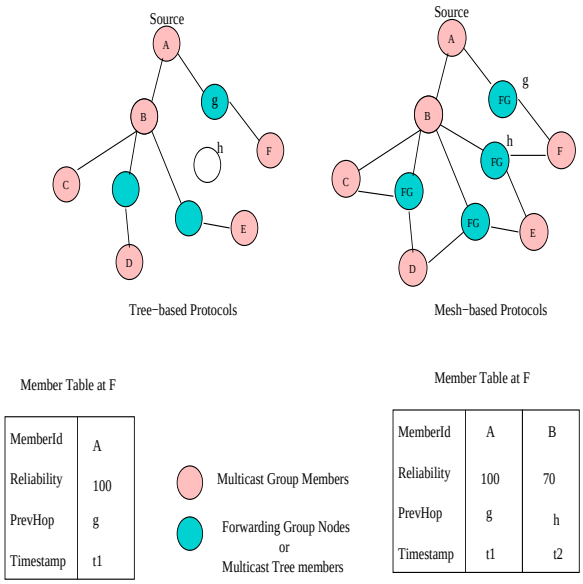


Figure 2: Member table maintained for local recovery.

3.2 Local Recovery

In wireless environments, losses may be caused by various factors. For the purpose of our congestion and error control mechanisms, we classify losses broadly into two categories: congestion- and transmission losses. Congestion losses include drops due to queue overflows and hidden terminal problems (we assume a contention-based MAC protocol IEEE 802.11 DCF [2]). Since the underlying multicast routing protocol (either tree- or mesh-based) uses broadcasting for data delivery, hidden terminal losses increase as traffic load increases. This is mainly due to the fact that in 802.11, channel access is performed as plain carrier sense multiple access (CSMA) with no ACKs for broadcast packets. Hidden terminal losses are also dependent on node density. Transmission losses include packet drops due to errors in the wireless channels (e.g., interference, noise) which can be caused by various factors including node mobility.

Recall that in the basic protocol, a receiver sends a feedback NACK to the source whenever it detects that packet losses are due to congestion using a fixed threshold for number of lost packets(three in our simulations). The sender then enters loss recovery mode and reduces its rate assuming network congestion. Since the losses are highly unpredictable in MANET environment, the nodes cannot effectively determine whether losses are due to congestion

or transmission errors. In either case, the source will try to recover the lost packets and will automatically slow down its rate. This may reduce throughput unnecessarily as reported losses may not be due to congestion. ReACT's local recovery mechanism tries to recover from transmission losses in the local neighborhood of the affected node, hence avoiding the source to unnecessarily reduce its sending rate.

Local recovery is also helpful in large, heterogeneous networks where different parts of the network might have different levels of "local" congestion. This eliminates the need for the source to restrict its rate just to satisfy a single member (or a small number thereof) located in a highly congested area. The affected member(s) can recover lost packets locally from nearby members. Recovering packets locally not only decreases the number of NACKs sent back to the source, hence avoiding unnecessary sending rate reductions, but also avoids unnecessary "global" retransmissions (as retransmissions from the source are multicast to the entire group) which may further increase incipient congestion.

Any local recovery scheme must obtain information of other group members as potential *recovery nodes*. In Anonymous Gossip [3], nodes get this information by adding additional fields to the underlying routing protocol's control packets. Our scheme gathers member information using the IP header, without relying on the underlying routing mechanism. Our local recovery scheme is thus routing-protocol independent. The main difference between our scheme and gossip style schemes is that we get information only about group members that are in the forwarding path from the source, whereas gossip tries to get information about all members.

Our mechanism of getting member information and selecting a recovery node works as follows:

- We add two fields in the the multicast data packet IP option field: <memberId, reliability>. These fields are updated as the packet is forwarded to the group. The memberId field contains the node-id of the multicast group member which forwarded the packet. Reliability field contains the current measure of reliability for the multicast source at that instant. Since node's reliability changes with time depending on the mobility and link conditions, we calculate reliability over a time window and take a moving average over the past. It is given by, $reliability_t = 0.3 \times reliability_{t-1} + 0.7 \times (A_r/E_r)$ where, A_r is the number of packets received

in the time window t and E_r is the number of packets that the node is supposed to receive in the window t. Note that we give more weight to the recent time window's reliability to address the dynamic nature of MANET's.

- Each member node maintains a member table of <memberId, prevHop, reliability, timestamp> for each multicast source and updates the table whenever it receives a multicast data packet using the IP option. It also updates the IP option fields with its own IP address as memberId and reliability based on the above formula. The prevHop field in the table identifies the node through which the multicast packet is received and the timestamp field denotes the time the entry was updated.

- Whenever a member decides to perform local recovery, it selects a member that has the highest reliability. The timestamp entry is used to timeout old entries and keep member information fresh. This is particularly important in MANETs where the topology can be very dynamic.

The above mechanism gathers upstream member (i.e., members that are on the path from the source to the node) information and is independent of the underlying multicast structure. Figure 2 shows a sample table maintained by a member node F both in mesh-based and tree-based protocols. In a mesh-based protocol we may have more than one upstream member because of path redundancy. As tree-based protocols also use broadcast for delivery, it is possible that a receiver might receive a packet from a node other than its parent node. Selecting a upstream recovery node based on its reliability increases the likelihood of successfully recovering packets locally, as nodes that are farther away from the source are less likely to have the requested packet(s).

Recall that one of the main goals of local recovery is to avoid the source to enter congestion control backing off its sending rate when it is not necessary. In ReACT, this is accomplished by: (1) trying to differentiate losses that are due to congestion from those that are not and (2) suppress feedback reporting losses not due to congestion. In order to report different types of losses, the ReACT receiver generates three different types of feedback packets according to the number of missing sequence numbers as follows:

- The LR feedback packet type is generated when $(missing\_packets <= LR\_THRESH)$. LR packets are always suppressed by the recovery node assuming that when losses do not persist for some time, they are likely not caused by congestion.

- If $(LR\_THRESH < missing_packets <= REPAIR\_THRESH)$, feedback packets of type REPAIR are generated; REPAIR requests are forwarded to the source if $(unrecovered\_packets > LR\_THRESH)$. If a reasonable number of losses can be recovered locally, the REPAIR request is suppressed. In this case, the recovery node adds the sequence numbers of the packets that could not be recovered and the repair initiator id to its *missing packet* cache hoping that they could be recovered later. Whenever a new packet is received at a recovery node, it checks its missing packet cache and if there is a match, it will forward the packet to the request initiator and remove the entry. To avoid multiple retransmission, we forward the packet only if it is recovered locally from some other node than the request initiator.

- Finally, persistent losses are considered indication of congestion; in this case, feedback type generated are NACK (i.e., if $missing\_packets > REPAIR\_THRESH$ and are forwarded to the source if they cannot be fully recovered locally.)

Currently in our simulations, the values of LR_THRESH, $REPAIR\_THRESH$, and REQ_LEN are 3, 10, and 20, respectively. Note that REQ_LEN is the maximum number of sequence numbers a packet can carry. Similar to TCP, we set LR_THRESH to three; this means that a node needs to experience at least 3 losses before generating congestion indication. These parameters should be decided based on the environment under which the protocol is operating. Preliminary simulations showed that $REPAIR\_THRESH = 10$ yielded reasonable indication of persistent congestion. The complete pseudo-code description of the algorithm at the sender and the receiver or intermediate node is presented in figure 3 and figure 4 respectively.

```
Initialization:
RX_LIST = {} ; FEED_BACK_RX = NONE
If (FEED_BACK_RX = NONE) Then Goto Normal_Mode Else Goto Recovery_Mode
Normal_Mode:
If (QUEUE_EMPTY) Then Wait forPACKET from Application or Network
Else
   If (PACKET = DATA_FROM_APP) Then
      Transmit Data
      Add to send buffer
      Goto Normal_Mode
   Else If (PACKET = ACK) Then Goto Process_ACK
      Else Goto Process_NACK:
Process_ACK:
   If (ACK.SENDER ∉ RX_LIST) Then
      If (ACK.RETX_SEQ = 0) THEN GOTO NORMAL_MODE
      Else
            (RX_LIST = RX_LIST + ACK.SENDER)
            (FEED_BACK_RX = ACK.SENDER)
            Retransmit requested seq number with feedback flag set
            Increment RETX_COUNTER and Set BACKOFF_TIMER
            Goto Recovery_Mode
   Else
      FEED_BACK_RX = GET_NEXT_RX_FROM_RX_LIST
      If (FEED_BACK_RX = NONE) Then Goto Normal_Mode
      Else
            Transmit new data with feedback flag set
            Increment RETX_COUNTER and Set BACKOFF_TIMER
            Goto Recovery_Mode
Recovery_Mode:
Wait forPACKET from Application or Network or Timer_Event
If (PACKET = DATA_FROM_APP) Then
   Add to the application queue and Goto Recovery_Mode
Else (PACKET = ACK) Then Goto Process_ACK
Else Goto Process_NACK
Process_NACK:
If (NACK.SENDER ∉ RX_LIST) Then
   RX_LIST = RX_LIST + NACK.SENDER
   IF (FEED_BACK_RX = NONE) THEN
      FEED_BACK_RX = NACK.SENDER
      Transmit new data with feedback flag set
      Increment RETX_COUNTER and Set BACKOFF_TIMER
      Goto Recovery_Mode
Timer_Event:
If (RETX_COUNTER < MAX_RETX) Then
   Retransmit packet with feedback flag set
   Increment RETX_COUNTER and Set BACKOFF_TIMER
   Goto Recovery_Mode
Else
   Remove FEED_BACK_RX From List
   FEED_BACK_RX = GET_NEXT_RX_FROM_RX_LIST
   If (FEED_BACK_RX = NONE) Then Goto Normal_Mode
   Else
      Transmit new data with feedback flag set
      Increment RETX_COUNTER and Set BACKOFF_TIMER
      Goto Recovery_Mode
```

Figure 3: Algorithm at the sender.

4 Experimental Setup

As our simulation platform, we use the QualNet network simulator [1], GloMoSim's [14] commercial successor. We evaluate the performance of our protocol for two different multicast routing protocols, namely ODMRP [8] and MAODV [13] [2]. For all the experiments, 50 nodes are placed randomly in a $1500m \times 1500m$ area and a randomly chosen group of 10 nodes join the multicast group. All the nodes join at the start of the simulation with some jitter and stay subscribed to the group till the end of the simulation. Five randomly selected members continuously send CBR traffic throughout the whole duration of the simulation with data payload size of 512 bytes.

[2] Due to space limitations, we do not describe MAODV's and ODMRP's operation

```
Receive_Mode: Wait for PACKET from network
If PACKET = DATA Then Goto Process_Data
Else Goto Process_Repair
Process_Data:
If (OLD_DATA) Then
    Drop the PACKET
    Goto Receive_Mode
Else
    Add data to receive buffer
    Check LR_CACHE for possible locally requested retransmissions
    If (PACKET.FEEDBACKID = NODE_ID)
        Check for lost sequence numbers
        Send back ACK to the source with lost sequence numbers
        Goto Receive_Mode
    Else if (TIME – LAST_REQUEST_SENT > REPAIR_INTERVAL)
        Goto Receive_Mode
    Else
        Check for packet loss
        If (0 < NUM_LOST <= LR_THRES) Then
            Send LR request to the chosen member
        Else If (LR_THRES < NUM_LOST <= REPAIR_THRES) Then
            Send REPAIR request to the chosen member
        Else If (NUM_LOST > REPAIR_THRES) Then
            Send NACK request to the chosen member
Process_Repair:
Check for lost seq in the receiver buffer
Send the requested packets found immediately
If (REPAIR.TYPE = LR) Then
    Add unrecoverable seq number and source of request to LR_CACHE
    Goto Receive_Mode
Else (REPAIR.TYPE = REPAIR) Then
    If (UNRECOVERED_PKTS > LR_THRES) THEN
        Forward repair to the source
    Else
        Add unrecoverable seq number and source of request to LR_CACHE
        Goto Receive_Mode
Else (REPAIR.TYPE = NACK) Then
    If (UNRECOVERED_PKTS > 0) Then
    Forward repair to the source
```

Figure 4: Algorithm at the receiver or intermediate node.

For MAODV, we allow some time at the start of the simulation, i.e., before sending any data, for tree establishment. We followed MAODV's IETF draft [12] specification as close as possible. However, the current version of the MAODV draft leaves unspecified many issues that can drastically impact MAODV's performance in a negative way. Thus we had to change MAODV's "standard" implementation to address these issues so that we conduct a fair performance comparison. We describe the important changes to MAODV below.

While caching group hello information for checking duplicate group hellos, MAODV draft specifies to use a cache with <group address, sequence number> pair. This creates major problem when many nodes join the group at the same time. When multiple nodes join at the same time to form a new multicast group, it is highly possible that all of them declare themselves as leaders as there is no initial multicast tree and the nodes will not get a RREP. This creates number of small groups with independent leader, which have to build together as a single group if they are connected. To detect the existence of multiple groups with in reach and combine them, MAODV uses group hello messages for this detection. Hence just <group address, sequence number> is not enough as you can receive a new group hello from a different leader with same group address and sequence number pair which will

be ignored as a duplicate. So we included the leader information in the cache to effectively detect multiple trees with in reach.

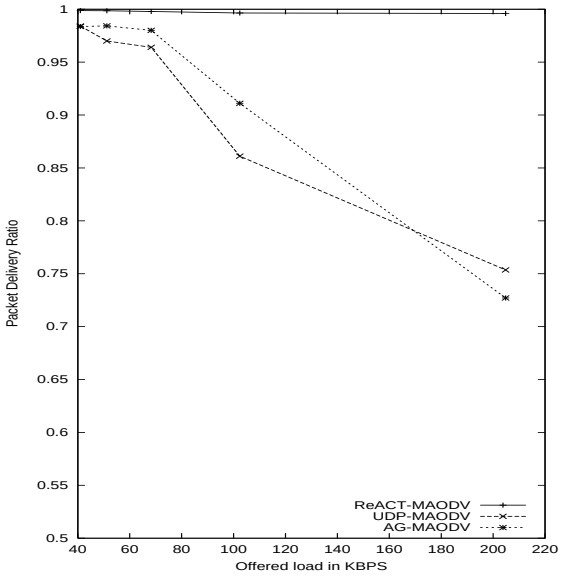
Another modification we made to MAODV is to maintain the tree effectively without any ambiguity between the upstream and downstream members. As per the MAODV specification, a link is assumed to be broken if you do not hear from your neighbor with in retransmit_ttime after multicast data. Only a node which is downstream to the link breakage initiates the repair and the upstream node removes the link from the nexthop. Since in a static network with high network density, hidden-terminal losses are more, a may detect its nexthop as a failed node erroneous. If the node which does this false detection of link failure is an UPSTREAM node and not the member of the multicast group, it may prune after a timeout if it is a leaf node. This will cause inconsistency in the tree as the downstream will still be thinking it is a part of the tree until it multicasts data. If it is just a receiver and not in the forwarding path from the source, then the detection is very difficult and may not detect this at all. To prevent this starvation, we added a new timer, which keeps track of the last heard time of the nexthop and eliminates this problem.

The values for MAODV and ODMRP parameters we use in our simulations are summarized in Table 1.

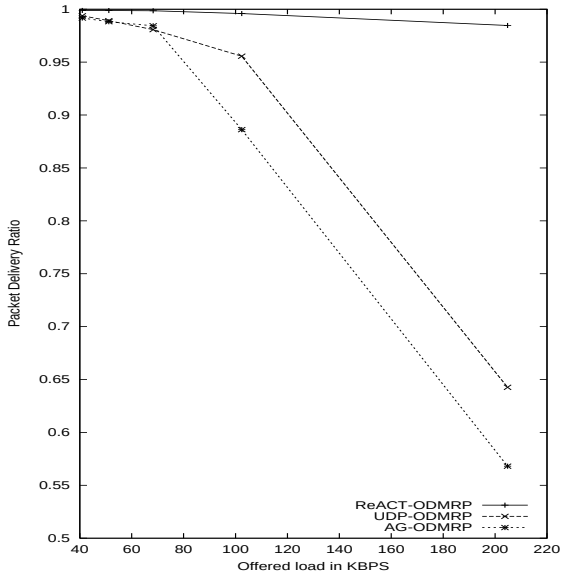
Table 1: Simulation parameters for MAODV and ODMRP

MAODV Parameter Name	Value
allowed_hello_loss	2
group_hello_interval	5 sec
hello_interval	1 sec
hello_life	3 sec
pkt_id_save	3 sec
prune_timeout	750 msec
rev_route_life	3 sec
rreq_retries	2
route_discovery_timeout	1 sec
retransmit_timer	750 ms
ODMRP Parameter Name	**Value**
JOIN DATA refresh interval	3 sec
Ackowledgement timeout for JOIN TABLE	25 ms
Maximum JOIN TABLE retransmission	3

In our study, we use the following performance metrics:



(a) Packet delivery ratio: MAODV



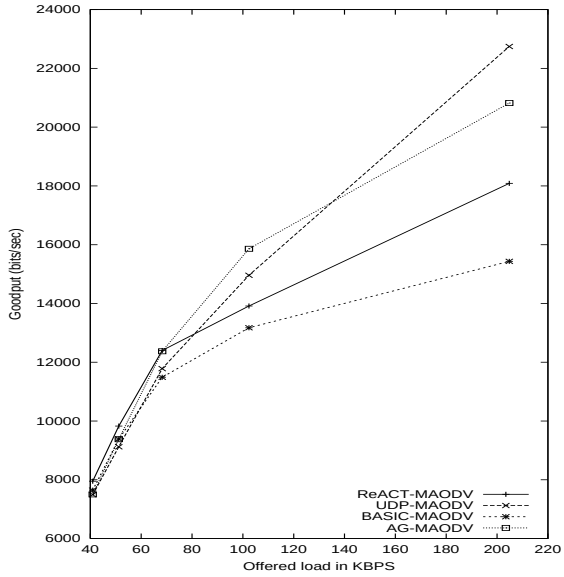
(b) Packet delivery ratio: ODMRP

Figure 5: Effect of congestion: Packet delivery ratio

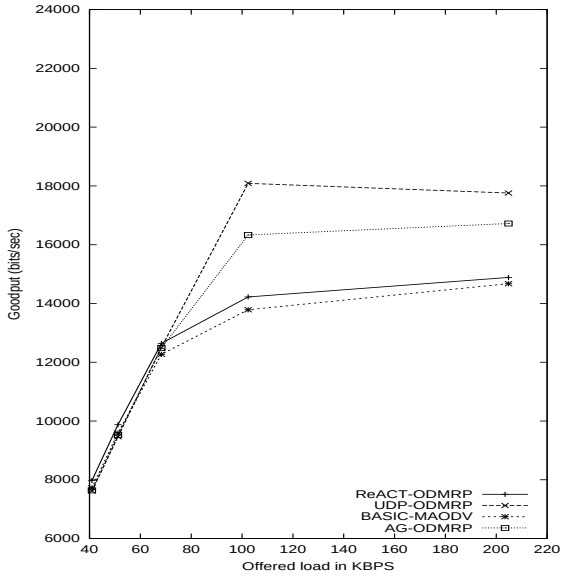
- **Reliability** is the ratio of the number of packets successfully (or reliably) delivered to each receiver over the number of packets supposed to be received by each member.
- **Goodput** measures throughput of packets reliably delivered, i.e., packets that are received by all members.
- **Overhead** measures the total number of packets sent (i.e., packets forwarded by the network layer). This includes control and data packets sent by the underlying multicast structure.

5 Results

We evaluate ReACT's performance when subject to range of network conditions. We are particularly interested in how ReACT performs under various offered loads, how it reacts to transmission losses, and what is the impact of node mobility. We also test ReACT TCP-friendliness.

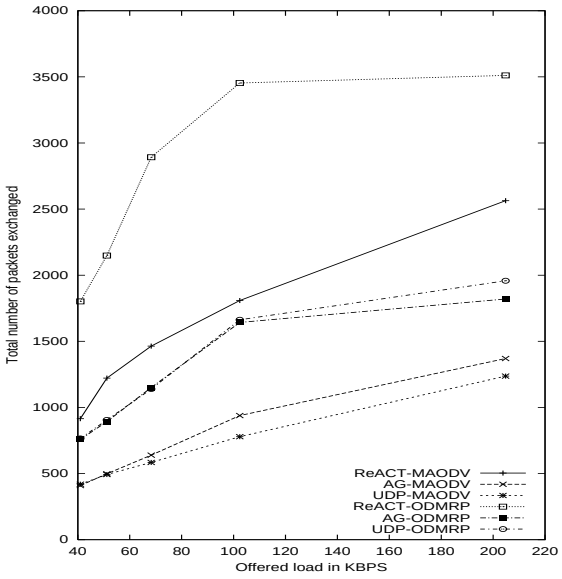


(a) Average goodput for each source over MAODV

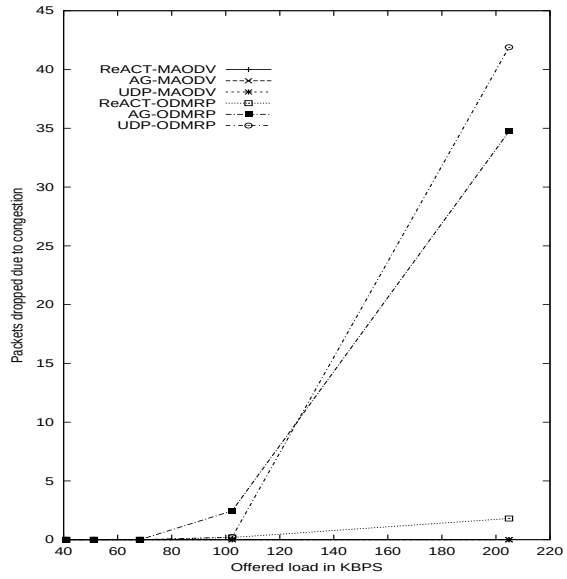


(b) Average goodput for each source over ODMRP

Figure 6: Effect of congestion: goodput



(a) Total packet transmission overhead



(b) Congestion Control

Figure 7: Effect of congestion: overhead and congestion control in action

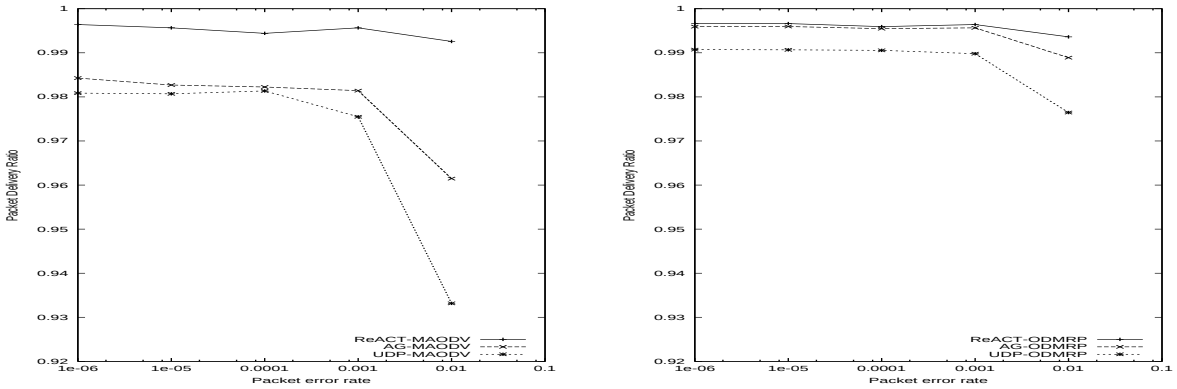
5.1 Effect of Congestion

MANETs are extremely sensitive to congestion which can greatly impact packet delivery ratio. To study this effect we varied the network offered load. Figure 5 shows the packet delivery ratio under different loads. Nodes are kept static (no movement is applied) and we measure packet delivery ratio, goodput and overhead for the different protocols over MAODV and ODMRP. As expected, packet delivery degrades as load increases. In particular ODMRP as a mesh-based protocol is more affected by congestion because of path redundancy in forwarding.

AG performs poorly as the load increases. Indeed AG does not improve the reliability of pure ODMRP. AG's poor performance is due to the fact that it does not perform congestion control and attest to the importance of congestion control in achieving reliable multicast in MANETs.

ReACT achieves near 100% reliability for all cases. The reason we do not achieve perfect reliability is because the application continuously sends packet and ReACT does not have a chance to recover all packets before the simulation ends.

Figure 6 illustrates the goodput achieved by the different protocols. As expected, ReACT's exhibits lower goodput than both AG and UDP as the latter do not employ congestion control. However, we observe that ReACT's local recovery mechanism is able to improve the protocol's overall goodput when compared to ReACT without local recovery (curve labeled "Basic"). This is because local recovery prevents the source from backing off its rate when packet losses are recovered locally. It also reduces the number of retransmissions at the source increasing its throughput. The price to pay is increased overhead due to its its aggressiveness in recovering packets without reducing the source sending rate (see Figure 7(a)). This effect is more pronounced in a mesh-based protocol than in a tree-based one. Nevertheless, as shown in Figure 7(b), which plots the number of packets dropped at the IP layer due to queue overflow, ReACT still minimizes network congestion by a large factor as shown in. The effect of congestion is more pronounced in ODMRP rather than MAODV. As expected, UDP exhibits the highest number of drops followed by AG since both protocols do not perform congestion control.



(a) Packet delivery ratio over MAODV

(b) Packet delivery ratio over ODMRP

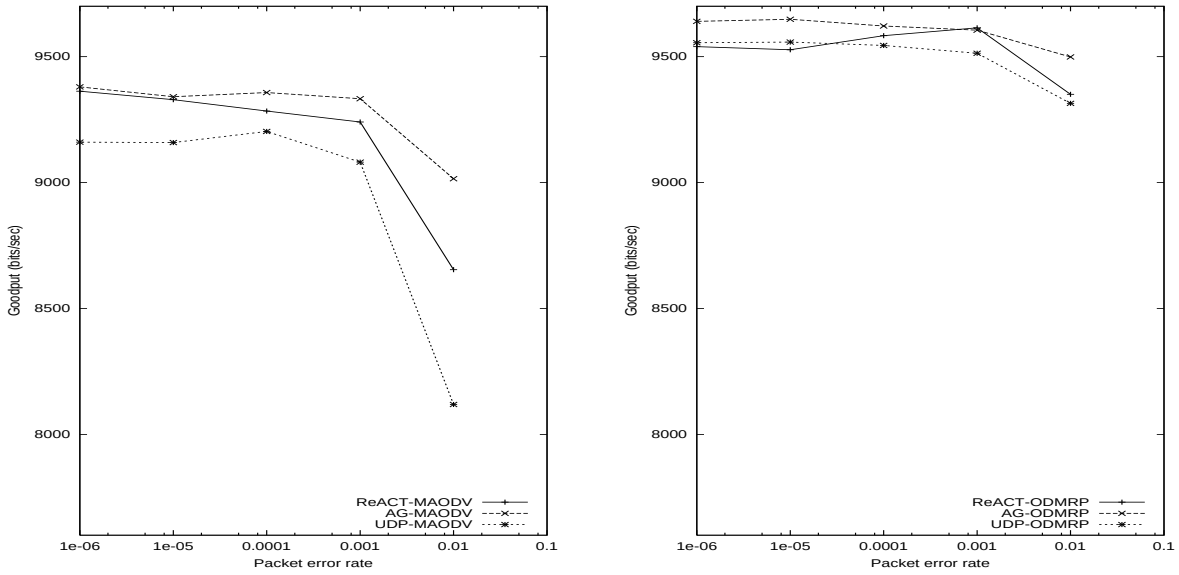
Figure 8: Effect of transmission losses: packet delivery ratio

5.2 Effect of Random Transmission Losses

In these experiments we focus on the performance of ReACT when subject to random link errors. We simulate link errors at the physical layer by generating random link losses according to a pre-defined error rate. Figure 8 shows the delivery ratio of the different protocols for different error rates. We can observe that without reliability mechanisms, the delivery ratio of plain UDP (which reflects the reliability that is delivered by the underlying routing protocol) goes down considerably as link error rates increase. In particular, the impact on tree-based protocols like MAODV is more severe as link errors may cause tree breaks and tree repairs are initiated. Whereas ReACT is able recover from these losses showcasing its local recovery feature. We should also point out that ReACT outperforms AG in terms of reliability under all conditions. Furthermore, because it uses local recovery, it is able to recover from errors without necessarily having the source reduce the sending rate as shown in Figure 9.

5.3 Effect of Node Mobility

In this section we study the effect of node mobility on performance. We use the random-way-point mobility model for mobility with 10*S* pause interval and we vary the average node velocity from 20m/s to 100m/s. As we can see from figure 10 packet delivery ratio of tree based protocol is affected a lot by mobility than a mesh based protocol. This is mainly because of the overhead in maintaining the tree with mobility. Frequent link breaks and tree repairs



(a) Average goodput over MAODV

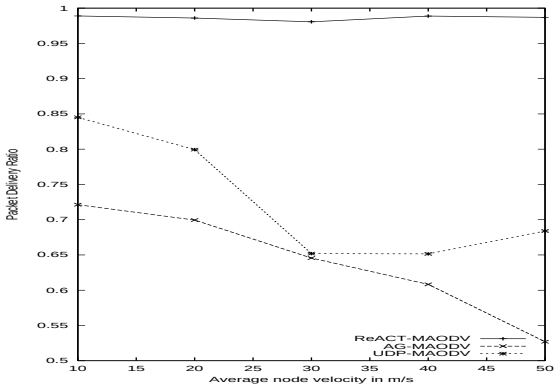
(b) Average goodput over ODMRP

Figure 9: Effect of transmission losses: Goodput

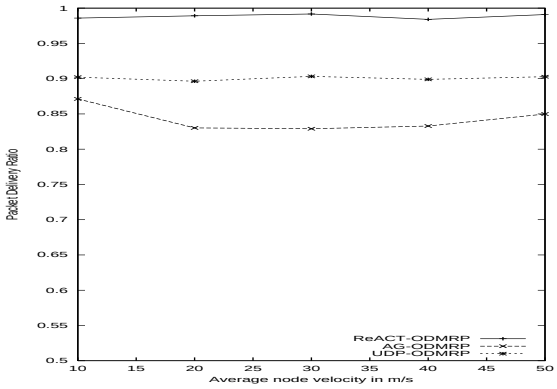
causes significant overhead. We can see that in a highly mobile scenario with a high load, AG fails to improve the performance. This is mainly due to the additional overhead incurred by the repair request and repair reply packets. ReACT is able to successfully recover the packets even with mobility and guarantees perfect reliability both over tree- and mesh-based protocols.

5.4 TCP Friendliness

In addition to analyzing ReACT's performance under a variety of MANET scenarios, we also needed to ensure that ReACT will compete fairly with other protocols, and in particular with TCP as it is very sensitive to congestion. The results presented in this section measure how fair ReACT is to TCP. We initiate a multicast group application with 10 receivers and 5 senders with sources generating traffic at the rate of 10 packets/second. In parallel, we run FTP and telnet traffic among 7 pairs of randomly chosen nodes. Table 2 shows the average throughput achieved by the TCP flows when competing with (multicast) UDP and ReACT. As can be observed, the throughput of TCP when competing with UDP is significantly lower than when ReACT is the competing protocol. This discrepancy

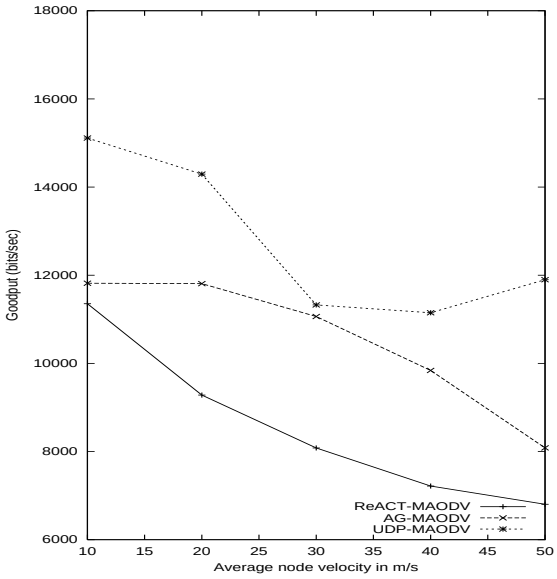


(a) Packet delivery ratio over MAODV

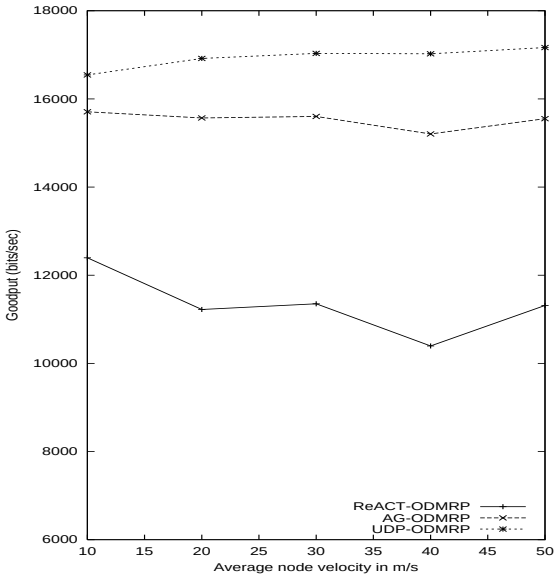


(b) Packet delivery ratio over ODMRP

Figure 10: Effect of mobility: packet delivery ratio



(a) Average goodput over MAODV



(b) Average goodput over ODMRP

Figure 11: Effect of mobility: goodput

Table 2: Average throughput (in Kbps) achieved by TCP sessions when competing with UDP and ReACT

Competing Transport Protocol	Over MAODV	Over ODMRP
UDP	83.48	49.26
ReACT	107.8	85.23

is even more accentuated when ODMRP is the underlying routing protocol as it is more sensitive to congestion.

The numbers shown in parenthesis in Table 2 denote the throughput of the protocol competing against TCP. In other words, in the first and second rows, these numbers refer to the throughput achieved by the competing UDP and ReACT session, respectively. They verify the hypothesis that by virtue of its congestion control mechanism, ReACT leaves a considerable portion of the bandwidth to competing TCP flows.

6 Conclusions

In this paper we presented the Reliable, Adaptive, Congestion-controlled multicasT (ReACT) transport protocol for reliable and timely multicast delivery in a wide range of MANET scenarios. One distinguishing feature of ReACT is that it combines source-based and local recovery. ReACT's source-based control includes a congestion control algorithm that adjusts the sending rate using a simple stop-and-wait mechanism based on receivers' feedback.

ReACT's error control mechanism combines source-based and local recovery. Ideally, the local recovery mechanism will mend no-congestion losses, avoiding the multicast source to back-off its transmission rate unnecessarily. Local recovery also tries to catch losses due to "localized" congestion (e.g., caused by the hidden terminal effect). Besides preventing the source from unnecessarily slowing down, ReACT's local recovery mechanism also manages to reduce "global" congestion by suppressing feedback ultimately addressed to the source as well as source-originated retransmissions. Also, by recovering locally, timeliness is preserved.

Through simulations, we evaluated ReACT's performance for a wide range of MANET scenarios using both mesh- and tree-based multicast routing protocols. We compare ReACT's performance against plain UDP and Anonymous Gossip. Our results show that ReACT is the best performer in terms of reliability, delivering close to

perfect reliability under a wide range of conditions. Our results also demonstrate the benefits of ReACT's local recovery mechanism which manages to prevent the source from reducing its rate unnecessarily, thus improving throughput considerably when compared to ReACT's throughput when local recovery is not employed. Finally we show that ReACT's performance is fairly insensitive to the underlying routing structure and that when competing for network resources, ReACT is fair to TCP.

As future work directions, we plan to investigate other congestion control schemes including window- as well as rate-based schemes. As a follow-on to our study of the interactions of ReACT with different routing protocols, we will also study the interactions of ReACT as a reliable transport protocol with MAC-layer protocols that provide link-level reliability.

References

[1] Scalable networks, http://www.scalble-solutions.com.

[2] Wireless lan medium access control (mac) and physical layer (phy) specifications.

[3] R. Chandra, V. Ramasubramanian, and K. Birman. Anonymous gossip: improving multicast reliability in mobile ad-hoc networks. *International Conference on Distributed Computing Systems*, pages 275–283, April 2001.

[4] M. Corson, J. Freebersyser, and A. S. (editors). *ACM/Baltzer Mobile Networks and Applications (MONET), special issue on Mobile Ad Hoc Networking*, October 1999.

[5] T. Gopalsamy, M. Singhal, D. Panda, and P. Sadayappan. A reliable multicast algorithm for mobile ad hoc networks. *Proceedings of ICDCS*, July 2002.

[6] Z. Haas, M. Gerla, D. Johnson, C. Perkins, M. Pursley, M. Steenstrup, and C.-K. T. (editors). *IEEE Journal on Selected Areas in Communications, special issue on Wireless Ad Hoc Networks*, (8), August 1999.

[7] P. Kermani and N. V. (editors). *IEEE Personal Communications Magazine, special issue on Advances in Mobile Ad Hoc Networking*, 8(1), February 2001.

[8] S.-J. Lee, M. Gerla, and C.-C. Chiang. On-demand multicast routing protocol. *Proceedings of IEEE WCNC*, 1999.

[9] S.-J. Lee, W. Su, J. Hsu, M. Gerla, and R. Bagrodia. A performance comparison study of ad hoc wireless multicast protocols. In *INFOCOM (2)*, pages 565–574, 2000.

[10] E. Pagani and G. Rossi. Reliable broadcast in mobile multihop packet networks. *Proceedings of ACM/IEEE MOBICOM'97*, pages 34–42, September 1997.

[11] C. E. Perkins. *Ad Hoc Networking*. Addison Wesley, 2001.

[12] E. M. Royer and C. E. Perkins. Multicast ad hoc on demand distance vector (MAODV) routing. *Internet Draft, draft-ietf-draft-maodv-00.txt*.

[13] E. M. Royer and C. E. Perkins. Multicast operation of the ad-hoc on-demand distance vector routing protocol. *ACM/IEEE MOBICOM*, 1999.

[14] UCLA. Glomosim: A scalable simulation environment for wireless and wired network systems.